Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework

Version 1.05
June 15, 2020

Developed and Published by Members of:

Practical Software & Systems Measurement

Product No.
PSM-2020-06-001

National Defense Industrial Association

International Council on Systems Engineering

Product No.
INCOSE-TP-2020-001-06

Editors:

Cheryl L. Jones
US Army
cheryl.l.jones128.civ@mail.mil

Geoff Draper
L3Harris Technologies
geoff.draper@l3harris.com

Bill Golaz
Lockheed Martin
willliam.h.golaz@lmco.com

Paul Janusz
US Army
paul.e.janusz.civ@mail.mil

PSM Product Number: PSM-2020-06-001
INCOSE Product Number: INCOSE-TP-2020-001-06

Contents

List of Figures

LIST OF TABLES

Executive Summary

This report provides recommendations for the measurement of continuous iterative developments (CID). It includes a series of diagrams and an ontology to describe the development approaches and terminology used. The report includes a Practical Software and Systems Measurement (PSM) CID measurement framework detailing common information needs and measures that are effective for evaluating CID approaches. This is documented in the “Information Category-Measurable Concept-Measures” (ICM) Table. The information needs address the team, product, and enterprise perspectives to provide insight and drive decision-making. This is documented in the ICM table described in Section 7. The framework also identifies an initial set of measures that have been identified as being practical measures to address these information needs. For the highest priority measures, sample measurement specifications have been developed that detail the identified measures. These are included in Section 8. Additional potential measures will be added in future releases, as described in Section 6.

A successful measurement program depends on establishing a clear context and operational definitions for the measures to be collected. Definitions can sometimes vary depending on the references and how measures are applied. The diagrams and definitions that follow provide the terminology used in this PSM CID measurement framework, in order to establish a common understanding, so that measures can be implemented and used consistently with community consensus.

This report is intended to be methodology and approach-agnostic and is written so that it may be adapted to organizational needs. Different methodologies and tools may use different terminology than defined in this report. The ontology in Section 3 provides synonyms for commonly used terms.

1. Front Material

The following sections provide overview information.

1.1 Background

A collaborative working group was established between Practical Software and Systems Measurement (PSM), the National Defense Industrial Association (NDIA) Systems Engineering Division, and the International Council on Systems Engineering (INCOSE) to develop a PSM measurement framework for Continuous Iterative Development (CID) in response to recommendations of the Defense Science Board (DSB) and Defense Innovation Board (DIB) studies.

Additionally, the U.S. Department of Defense (DoD) is making a transformational change in acquisition policy by redesigning the Defense Acquisition System, including the addition of a new Software Acquisition Pathway (Software Acquisition Pathway Interim Policy and Procedures, 2020). The general guidelines for this new acquisition policy are established in Section 800 of the 2020 National Defense Authorization Act. The pathway promotes Agile and DevSecOps and allow for upgradeable and timely fielding of software in a way that aligns with this CID approach. The measurement recommendations in this report provide a methodology to measure the Execution Phase of the Software Acquisition Pathway. These CID measures also apply to other non-DoD domains.

The most critical information needs and measures have been prioritized, based on a series of surveys with members of relevant NDIA, PSM, and INCOSE working groups. Additional measures will be specified, and revisions to the information needs will be included, as additional input is provided. This framework will be improved over time. We welcome your recommendations and comments.

1.2 CONTRIBUTORS

Table 1: PSM CID Measurement Framework Editors

Editors	Organization
Cheryl Jones	Army Futures Command – CCDC Armament Center
Geoff Draper	L3Harris Technologies / NDIA Systems Engineering Division
Bill Golaz	Lockheed Martin Corporation
Paul Janusz	Army Futures Command – CCDC Armament Center

Table 2: Core Team Contributors and their Organization

Core Team	Organization
Steve Cox	Telecote Research
Firas Glaiel	Raytheon Company
Stephen Henry	Defense Acquisition University
Suzette Johnson	Northrop Grumman Corporation
Jonathan Kiser	The Boeing Company
Jason McDonald	L3Harris Technologies
Greg Niemann	Lockheed Martin
Carmela Rice	Office of the Undersecretary of Defense, Acquisition and Sustainment (OUSD A&S)
Garry Roedler	Lockheed Martin Corporation / INCOSE
David Rosenfeld	L3Harris Technologies
Larri Rosser	Raytheon Company
Robert Simmons	Raytheon Company
Robin Yeman	Lockheed Martin Corporation

Additional thanks go to the many additional colleagues who contributed to the development of the guide thorough participation in meetings, workshops and reviews.

Table 3: Additional Contributors to the Report

Additional Contributors	Organization
Elizabeth Ashwood	Quantech Services
Dr. Barry Boehm	University of Southern California / Systems Engineering Research Center (SERC)
Dr. Jeff Boleng	Office of the Undersecretary of Defense, Acquisition and Sustainment (OUSD A&S)
Katherine Bradshaw	US Air Force AFCAA
Cherrie Brown	US Navy
Connie Bustillo	Lockheed Martin Corporation
Kevin Chapman	L3Harris Technologies
Dr. Robert Charette	ITABHI Corporation
David Chesebrough	NDIA
Dr. Bradford Clark	Software Metrics Inc.
Christopher Costello	Army Futures Command – CCDC Armament Center
Victoria Cuff	Office of the Undersecretary of Defense, Acquisition and Sustainment (OUSD A&S)
Kyle Davis	Quantech Services
James Doswell	US Army DASA-CE
Rick Dove	Paradigm Shift / INCOSE Agile Systems and Systems Engineering Working Group
Kim Elliott	Raytheon Company
Joseph Elm	NDIA Systems Engineering Division
Esma Elmazaj	L3Harris Technologies
Trevor Enos	US Air Force
Will Hayes	Software Engineering Institute, Carnegie Mellon University
Diane Juhas	Raytheon Company
Matt Kennedy	US Department of Treasury
Jessica Li	Lockheed Martin Corporation
Lindsay Migala	US Air Force
William J. Nichols	Software Engineering Institute, Carnegie Mellon University
Victoria Perez	US Air Force
Bernard Reger	Army Futures Command – CCDC Armament Center
Gene Rosenbluth	Northrop Grumman Corporation
Ranjit Singh	Lockheed Martin Corporation
Roz Singh	Raytheon Company
Dan Strickland	Missile Defense Agency (MDA)
Steven Verga	L3Harris Technologies
Marilyn Vickers	US Air Force

2. Major Concepts

This PSM CID measurement framework provides guidance on information needs and measures from three perspectives: team, product, and enterprise. In many cases, the same base measures may be used, although aggregated to higher levels for product or enterprise needs. In other cases, different base measures may be used. The measurement specifications provide initial guidance on tailoring measures and indicators for these different perspectives and aggregation levels.

For CID, stakeholders include actual users of the system and software, as well as the development team, customer, and enterprise managers. The measures need to provide value to all stakeholders and inform diverse information needs.

One of the major issues with measures is ensuring that they provide information needed to support decision making and that they are used. A small set of measures should be tailored for each program and organization, focused on those needed for fact-based decision making. The measures should be regularly reviewed to ensure they are being used. If not, other measures may be required, or additional training may be required for decision makers on how the measures can be utilized.

2.1 CID Work Decomposition

Figure 1 contains a sample work decomposition approach for CID. This terminology will be used throughout this report and the associated ICM Table and measurement specifications.

Mission Requirements or Capabilities are the top level of user requirements. They are often documented in a roadmap. The roadmap is a top-level view of capabilities, which evolves over time as the CID process is performed. For DOD systems, the mission requirements may begin in the Joint Capabilities Integration and Development System (JCIDS), Capability Needs Statement (CNS), or an equivalent document. Capabilities are then decomposed into features which are then decomposed into stories, which may be decomposed into tasks.

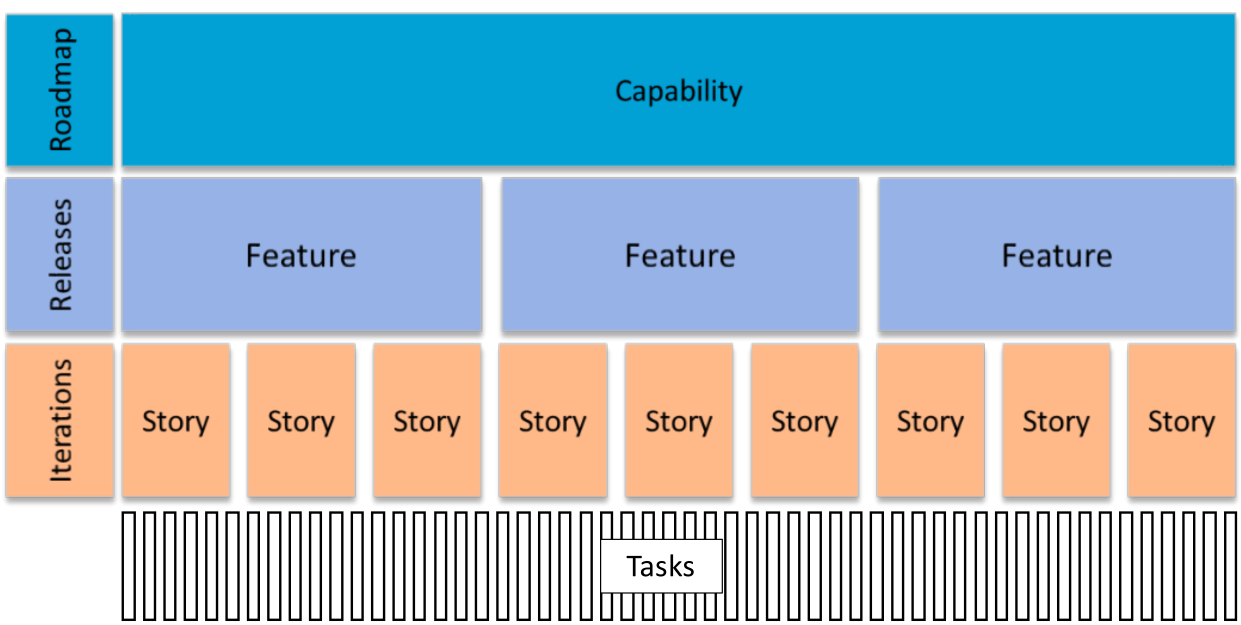


Figure 1: CID Work Decomposition

2.2 Measurement Context Diagram

Figure 2 illustrates the context for common measures of continuous iterative development as they are defined and applied in the PSM CID measurement framework and measurement specifications. The diagram should be interpreted as a model supporting multiple iterations throughout development and operations. Although intended to be broadly applicable across a range of domains, adopters of the framework should further interpret, tailor, and apply these measures as applicable to their own business context.

Measurement may occur in each of many potential stakeholder environments. Not all organizations will have all of these environments, as distinct entities. Different levels of sophistication of these environments may be used by different teams, for different levels of evaluation. Possible environments include:

- Development/Integration Environment(s)
- Production Representative Environment
- Operationally Relevant Environment
- Operational Environment

The enterprise generally focuses on actual measures from the operational environment. The team or product measures may begin in earlier environments, and focus on ensuring objectives will be met as the system is developed and sustained. Similar activities may be performed in different environments, with separate measures of effectiveness.

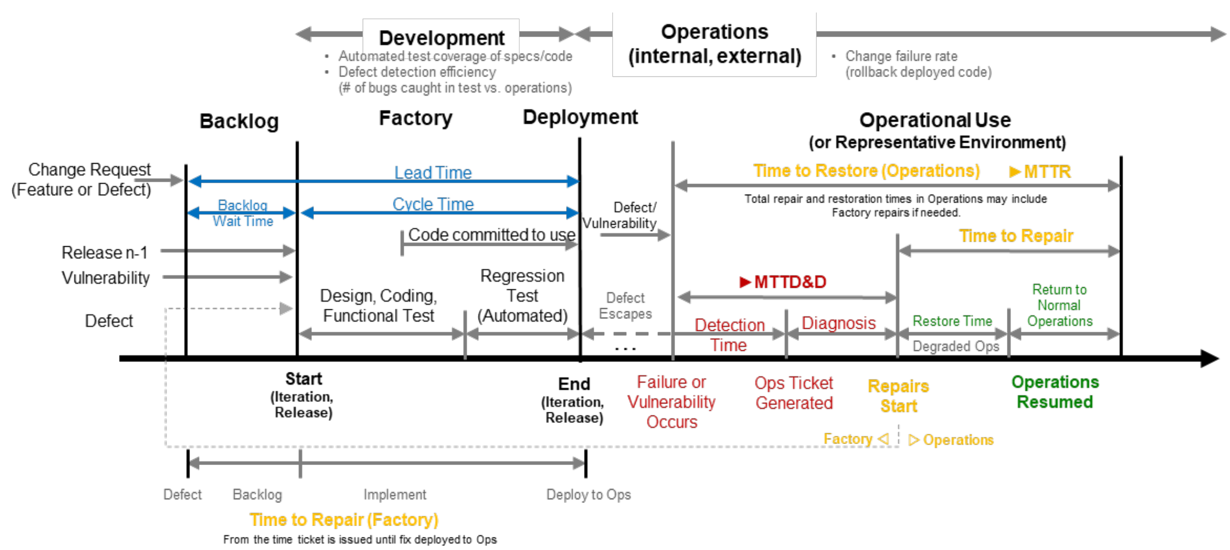


Figure 2: Measurement Context Diagram

Adapted from: https://limblecmms.com/blog/mttr-mtbf-mttf-guide-to-failure-metrics/

Major elements of this diagram for interpreting the context for candidate measures in the PSM CID measurement framework, emphasized by the bolded text labels, are described below. Additional details on individual measures are provided in the measurement specifications.

- **Backlog**: A collection of proposed work items to be implemented (see Section 3 for full description). Work items may include user needs (new or unfilled items) or defects from prior releases. Work proceeds for only those requests that are prioritized and accepted for implementation (committed work).

- **Factory**: Development proceeds through the Factory processes (requirements, design, implementation, test) for committed work and culminates with deployment. Work is planned and implemented iteratively (a recurring series of iterations and releases).
- **Operations**: Completed work from the Factory is Deployed in a new release to internal or external Operations, which may include a developer integration/test environment, end use Operations, or other intermediate operationally representative environments (e.g., operational test bed). The measures shown may be relevant to any or all of these environments. See Figure 3 for additional details on internal and external operations.
- **Rework:** The release(s) deployed may need to be updated to account for defects, security vulnerabilities, or other anomalies that affect the delivery of deployed services. Defects (e.g., trouble tickets) are issued for these requested changes. Operations may be able to continue in a degraded mode (e.g., workarounds, redundant paths) until full service is restored. Restoration time (Time to Restore) includes the time to detect and diagnose the error (MTTD), and to implement and deploy repairs (MTTR). Some repairs may be possible directly in Operations (e.g., network issues, configuration changes, restarting COTS software); others may need to be routed to the Backlog for prioritization. The colors (Red, Yellow, Green) in this figure indicate the transition from observation of the issue, to initiation of repairs, and to restoration of normal operations.

2.3 Defect Terminology

Defect terminology may also change from one methodology or company to another. Defect terminology used in this PSM CID measurement framework is defined in the ontology in Section 3, consistent with Figure 3. Operationally representative environments can be either internal or external.

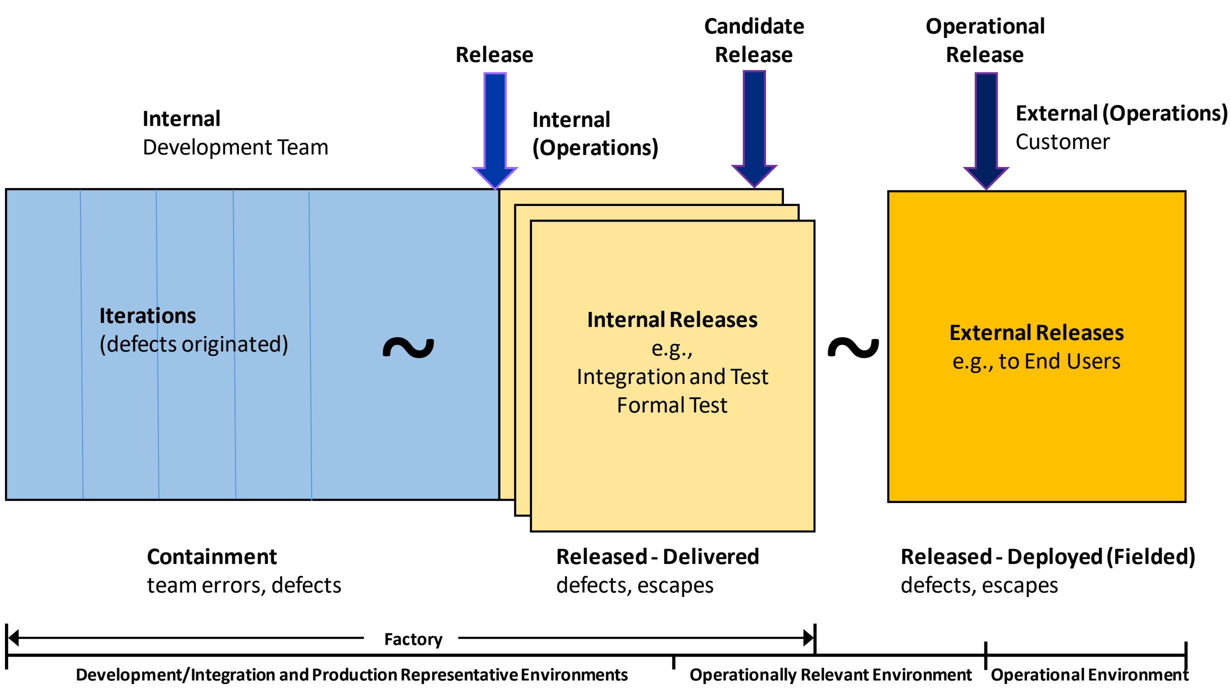


Figure 3: Defect Terminology

2.4 CID PROCESS

Figure 4 provides a conceptual depiction of the base measures that are collected for iterative releases and deployments to operations. There may be many iterations that are produced for internal use and continued development (for example v0.n, v1.n, v2.n in Figure 4). A subset of these are candidate releases that are available for external use (for example Release 1.0 in the figure), with a subset of these actually released for operational use (for example Release 2.0 in Figure 4). Some of these releases are assigned conceptual terms (MVP, NVP, MVCR) indicating the maturity of the product capability for early operational use; refer to Section 3 for descriptions.

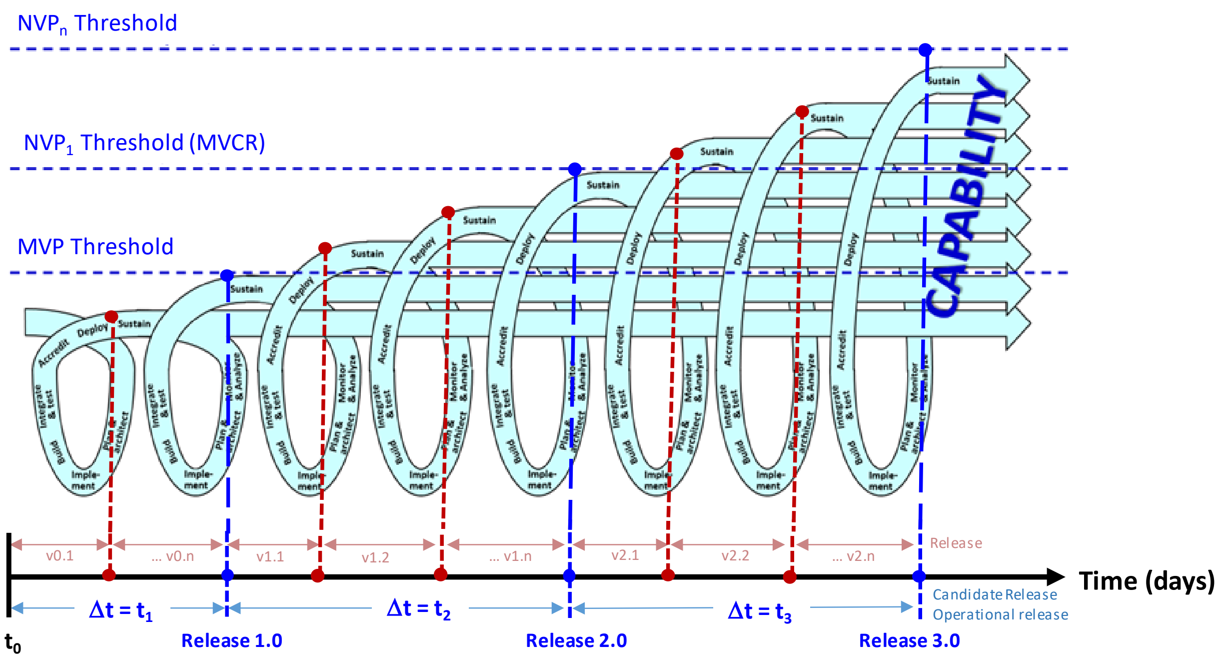


Figure 4: Continuous Iterative Development Process

3. Ontology and Definitions

The terms in Table 4 are used in the PSM CID measurement framework and specifications. Related terms are illustrated in figures 1, 2, and 3, and are grouped together in this section. The terms and definitions used here are drawn from several sources, including common industry best practices (defense and commercial), inputs from subject matter experts, DoD Software Acquisition Pathway policy and guidance, and DSB/DIB software acquisition reports. (See Bibliography for references.)

Table 4: PSM CID Measurement Framework and Specifications Terms

Term	Synonyms	Description
Continuous Iterative Development (CID)	Agile, DevOps, DevSecOps, SAFe	A method of managing development, testing, and release of software, or systems, to continually, or iteratively, provide working functional systems of increasing capability to internal and external customers.
Roadmap		A high-level visual summary that maps out the vision and direction of product offerings over time. It describes the goals and capabilities of external releases.
Capability	Epic, Mission Requirements	Higher-level solutions typically spanning multiple releases. For DoD, these may be reflected by a Capability Needs Statement (CNS) or JCIDS capabilities. Capabilities are made up of multiple Features to facilitate implementation.
Feature		A service or distinguishing characteristic of a software item (e.g., performance, portability, or functionality) that fulfills a stakeholder need and includes benefit and acceptance criteria within one release. Features are used to complete capabilities and are comprised of multiple Stories (or tasks, use cases, etc.).
Story	Use cases	**User Story.** A small desired behavior of the system based on a user scenario that can be implemented and demonstrated in one iteration. A story is comprised of one or more tasks. In software development and product management, a user story is an informal, natural language description of one or more features of a software system. User stories are written from the perspective of an end user or user of a system. **Use Case.** In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a user and a system (or between software elements), to achieve a goal. Use cases can be used in addition to or in lieu of user stories.
Story Points		A subjective value assigned by the developing team to a story to provide a relative measure of effort and complexity. Story points are a unit-less value: they are a scalar indicator of relevant complexity. Story points are generally not comparable across teams.
Task		Steps to be completed to satisfy a Story.

Term	Synonyms	Description
Cycle Time		The elapsed time from when work is put into progress until the time work has been completed.
Lead Time		The elapsed time from when work is identified, and a request is provided to the time the request has been satisfied. Note: The time the request has been satisfied is usually the same time the associated work is completed.
Backlog	Program Backlog Release Backlog	**Product backlogs** identify detailed user needs in prioritized lists. The backlogs allow for dynamic reallocation of scope and priority of current and planned software releases. The backlog contains new capabilities/features, changes to existing capabilities/features, defect fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome. Issues, errors, and defects identified during development and operations should also be captured in the product backlog to address in future iterations and releases. The development team works with the user community to decompose and prioritize the roadmap capabilities into product backlog entries. An **iteration backlog** is a list of the new stories, changes to existing stories, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome, within a near term iteration cadence. The iteration backlog contains a decomposition of product backlog entries into lower level items, for those prioritized for near-term implementation.
Problem Report	Defect Report, Discrepancy Report, Trouble Ticket	Identified issue with the product. Once approved for implementation, a Change Request, or Story, may be created, or the Problem Report may be used to track implementation. Service incidents in Operations are typically recorded in trouble tickets or equivalent.
Defect	Errors, Issues	A defect is a condition in a (software, system, hardware) product which does not meet its requirements or end-user expectation, causes it to malfunction or to produce incorrect/unexpected results, or causes it to behave in unintended ways. Defects may be documented in problem reports (or trouble tickets), or they may be added to the backlog for consideration in future iterations. • **Escaped Defects** are defects detected, or resolved, after release of the product and version containing the defect. Defects are generally tracked separately for internal and external releases • **Contained Defects,** also known as Saves, are defects detected and resolved before internal or external release of the product and version containing the defect.
Change		Revision that adds, removes, or modifies any aspect of the product. Note: Identified changes may be documented using Stories or Features.

Term	Synonyms	Description
Change Request	CR	Requested change to the product. Some organizations may use Problem Reports instead of separate Change Requests to track issues.
Release	Build, Increment	A grouping of Capabilities and/or Features that can be used for demonstration, evaluation, or delivery. A release may be internal for integration, testing, or demonstration; or external, to system test or as user delivery. A release may be based on a time block or on product maturity.
Internal release		A release that is ready for internal use outside of the development team. It may be used for integration, testing, or demonstration.
Candidate Release	External Release	A release that has been through the pipeline and system test, and is ready for transition to the user.
Operational Release	Deployment Release	A release that has been approved for operational use.
Iteration	Sprint	A small internal time block in which the development team develops and demonstrates a set of Stories. An iteration is a full development cycle that can result in a Release. In some methodologies, an iteration is called a Sprint.
MVP / MVCR / NVP		**Minimum Viable Product (MVP):** An early version of the software to deliver or field basic capabilities to users for evaluation and feedback. Insights from MVPs help shape scope, requirements, and design of future product releases. **Minimum Viable Capability Release (MVCR)**: A set of features suitable to be fielded to an operational environment that provides value and capability to the end user/warfighter in a rapid timeline. The MVCR delivers initial warfighting capabilities to enhance some mission outcome(s). The MVCR, used in DOD software policy, is analogous to a Minimum Marketable Product (MMP) in commercial industry. **Next Viable Product (NVP)**: The next set of features in the succeeding product delivery.
Release Style		There are three types of release styles: Cadenced (e.g., Quarterly), Feature-based (e.g., Minimum Viable Product), and Continuous Deployment. Continuous Deployment takes significant discipline, and therefore requires more maturity. Most programs will do some form of cadenced release/iteration schedule, with specific time blocks.

4. Mapping Data to Measurement Specifications

In the PSM methodology, the information model links the data that can be measured to a specified information need, as illustrated in Figure 5. More detail on the discussions in this section can be found in Practical Software and Systems Measurement (John McGarry (Author), 2001).

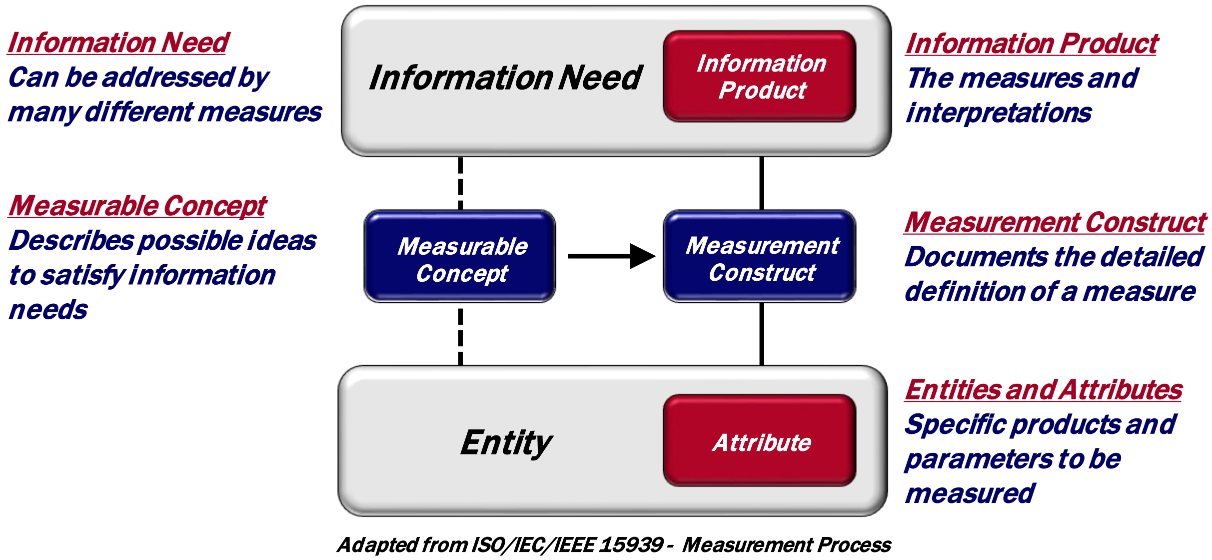


Figure 5: Information Model - High-Level View

The things that can actually be measured include specific attributes of the systems and software processes and products, such as size, effort, and number of defects. The measurement construct describes how the relevant attributes are quantified and converted to indicators that provide a basis for decision making. A single measurement construct may involve three types, or levels, of measures; base measures, derived measures, and indicators. The measurement planner needs to specify the details of the measurement constructs to be used in the measurement plan, as well as the procedures for data collection, analysis, and reporting.

At each of the three levels of measures - base measures, derived measures, and indicators - additional information content is added in the form of rules, models, and decision criteria. Figure 6 illustrates the structure of a measurement construct in more detail. This figure depicts how the base measures collected are dependent on the information needed by management. It also shows how the data is combined into an indicator and analysis model to form the information product provided to management.

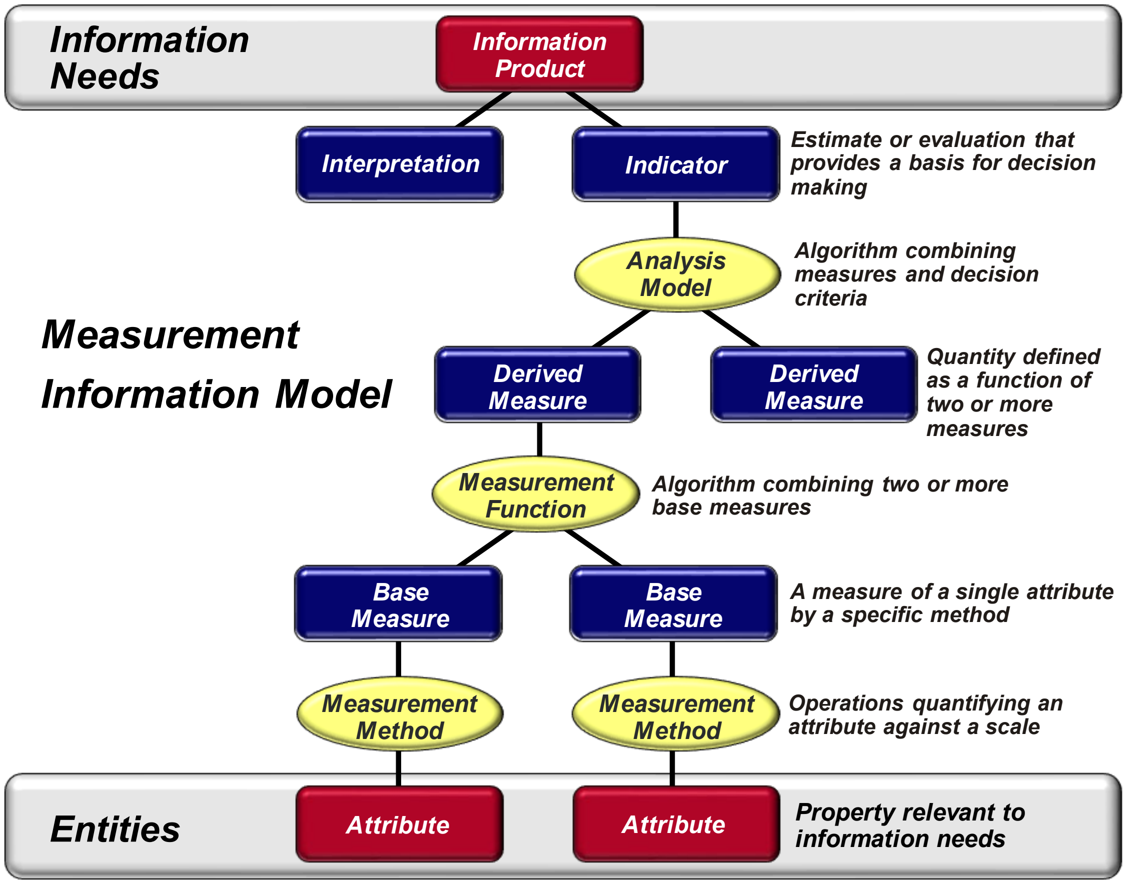


Figure 6: Measurement Information Model

Figure 7 contains a specific example of this, for the defect detection measure that is specified in Section 8.6. The measurement specifications in Section 8 detail the information needs, base measures, derived measures, and analysis models for each proposed measure.

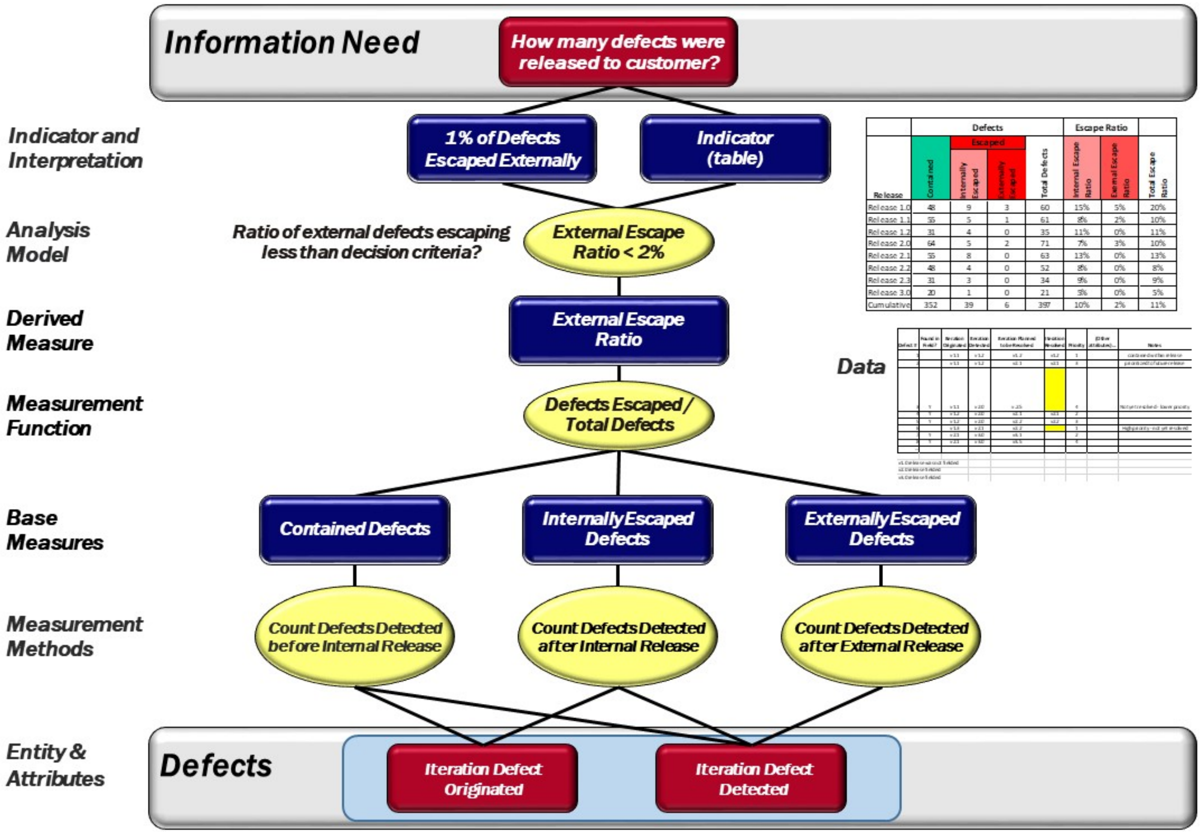


Figure 7: Mapping Data to Measures

5. Measurement Principles

The "Information Category-Measurable Concept-Measures" (ICM) Table provides the PSM CID measurement framework detailing common information needs and measures that are effective for CID approaches. The information needs address team, product, and enterprise perspectives. These different perspectives have different information needs and concerns. In some cases, the same base measures may be aggregated to address high-level information needs. In other cases, unique measures are required. The ICM Table also identifies a set of measures that have been identified as being practical measures to address these information needs, based on practical experience from the working group members. The ICM table is included in Section 7.

Some key principles for these information needs and measures include:

- The set of measures included in the ICM Table are sample measures identified through survey and subject matter expert (SME) review as being important in selected circumstances and at various levels.
- Team, product, and enterprise measures are included: not all can be aggregated.
- A minimum practical set of measures should be selected and tailored based on organizational and program circumstances, tools, and processes. Often organizations or programs will select a subset of these measures to emphasize for implementation and decision-making.
- The selected measures should have an identified stakeholder, inform decisions or answer key programmatic questions, and drive actions. They allow early visibility into the issues so that timely corrective action can be taken.
- The set of measures are process agnostic, but they were specifically developed for continuous iterative development. Other PSM materials represent a broader set of materials and processes.
- The collection of measures should be automated to the extent practical and integrated with business workflows.
- A balance between speed and quality needs to be maintained, as illustrated in Figure 8. There is often a 'sweet spot' tradeoff between speed and quality that delivers a best value solution based on project objectives. Quality needs to be monitored, in addition to speed, to ensure that these measures are appropriately balanced. An over-emphasis on speed can be at the expense of product quality. An over-emphasis on quality can slow the speed of delivery.

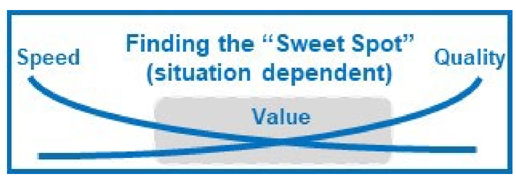


Figure 8: Speed - Quality Sweet Spot

For the highest priority measures, sample measurement specifications have been developed that detail the identified measures. Measurement specifications have been developed for:

- Automated Test Coverage
- Burndown
- Committed vs. Completed Progress
- Cumulative Flow
- Cycle Time / Lead Time
- Defect Detection
- Defect Resolution
- Mean Time to Restore (MTTR) / Mean Time to Detect (MTTD)
- Release Frequency
- Team Velocity

See Section 8 for these specifications. The ICM table and the sample measurement specifications can also be found at http://www.psmsc.com/CIDMeasurement.asp

6. Next Steps

This version of the PSM CID measurement framework is an initial set of measures that have proven to be useful in practice. Additional measures will be considered and added in future releases. One of the most critical missing elements is a measure of user value. This is a measure of the value of a particular capability or feature to the end user in the operational environment. There is also a separate measure of business value for items that are important to the program, but not of interest to the end user. Another critical missing element is how to count size for estimating.

Known future additions include:

- Value assessment (from end user, acquirer, supplier, and business perspectives)
- Technical Debt
- Security
- Size measures for estimating
- Additional focus on enterprise measures

7. ICM Table

Table 5: Issues, Categories, and Measures

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
Schedule and Progress	Work Unit Progress (team, product) Milestone Completion (enterprise)	Are story points delivered as committed? Are we still on track to deliver all story points per roadmap? (on plan)	Are features/capabilities delivered as committed? Are we still on track to deliver all features/ capabilities per roadmap? (on plan) What are the features/capabilities at risk of not being completed as scheduled? Are all capabilities/ requirements assigned to releases?	Are capabilities delivered as committed? Are we still on track to deliver all capabilities per roadmap? (on plan) What are the capabilities at risk of not being completed as scheduled?	Burndown Committed vs. Completed Velocity
	Work Unit Progress		Did we deliver expected capabilities / features? Is the roadmap still valid?	Is the user satisfied with the delivered products? Do they provide the desired functionality when needed?	Feature or Capability Implementation
	Work Unit Progress		Is the integration and test progress proceeding as planned?		Test Progress
	Work Unit Progress		Is the flow of work moving forward through the process work flow states?		Cumulative Flow
	Work Backlog		How much outstanding technical or mission debt exists?		Feature or Capability Backlog Technical Debt

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
Resources and Cost	Financial Performance		What is the cost to release? (capability development through deployment)	What is the cost to release? (capability development through deployment)	Cost ($) Effort
	Financial Performance		What is the estimated cost and schedule for a new CID product or release? What is the estimated cost and schedule per feature or capability?	What is the estimated cost and schedule for a reference feature or capability? (historical reference)	Estimate vs. Actual Cost/Effort Estimate vs. Actual Effort Estimate vs. Actual Schedule Earned Value
	Financial Performance		Are the feature level estimates accurate and feasible?	How accurate are the estimates across the set of enterprise programs?	Estimation Accuracy
	Personnel Effort	Do we have the appropriate team members for each identified role (skills and skill levels) with appropriate training?			Staff Experience
	Personnel Effort		How much turnover is occurring on the teams and as a whole?	How much turnover is occurring on the programs?	Team Turnover Rates Program Turnover Rates
	Personnel Effort	What is the satisfaction of the workforce?	What is the satisfaction of the workforce?	What is the satisfaction of the workforce?	Net Promoter Score (NPS)
	Facilities and Support Resources			How quickly can a new tool chain or environment be deployed?	Time to Deploy

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
Size and Stability	Functional Size and Stability Physical Size and Stability	How much work must be done?	How much work must be done?	How much work must be done?	Committed vs. Completed Requirements SLOC (Function Points?)
	Functional Size and Stability		How volatile are capabilities or features? Are we adding more features? What is the ability to accommodate changes in user needs?	How volatile are capabilities or requirements? What is the ability to accommodate changes in user needs?	Feature Volatility Capability Volatility Backlog Volatility
	Functional Size and Stability	How much of the product is newly developed vs. reused from other sources?			Reuse of Artifacts
	Functional Size and Stability		What value is being provided?	What value is being provided?	User/Warfighter Value Mission Effectiveness Business Value
Product Quality	Functional Correctness	Do features/stories work as expected?	Do features/capabilities work as expected?	Do capabilities work as expected? Is rework identified and managed?	Acceptance of Completed Work (Stories, Features, Capabilities) Rework Stories Enhancement Stories Defect Detection Defect Resolution
	Functional Correctness	Do changes break previous functionality?	Do changes break previous functionality?	Do changes break previous functionality?	Rework Defects Rework Hours Rework Stories Change Failure Rate Rollback Defect Density

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
	Functional Correctness	How many defects were contained (discovered) prior to internal release? How many defects were released (escaped) to an internal customer (e.g., Integration and Test, Formal Test)?	How many defects were released (escaped) to an internal customer (e.g., Integration and Test, Formal Test) or released (escaped) to an external customer (e.g., end users)?	How many defects were released (escaped) to an external customer (e.g., end users)?	Defect Detection
	Functional Correctness	What is the product quality delivered from the development team?	What is the product quality delivered to the field?	What is the product quality delivered to the field?	Defect Detection Defect Resolution
	Value	Do features/stories work as expected?	Does the delivered product meet the operational need?	Does the delivered product meet the mission need?	Value Assessment
	Security - Safety		How secure is the product		Vulnerabilities
	Supportability - Maintainability Dependability - Reliability		What is the reliability and availability of operational capabilities? How long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?	What is the reliability and availability of operational capabilities? How long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?	Mean Time to Restore (MTTR) Mean Time to Detect (MTTD)
	Supportability - Maintainability Dependability - Reliability		What is the reliability and availability of the environment (e.g., people, process, infrastructure)?	What is the reliability and availability of the environment (e.g., people, process, infrastructure)?	Environment Reliability

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
Process Performance (Process Effectiveness)	Process Efficiency - Speed Security - Safety		How quickly can new security vulnerabilities be resolved and deployed to fielded products?	Is the system cyber-resilient?	Security Vulnerability Lead Time Mean Time to Restore (MTTR)
	Security - Safety		Have all safety-critical items been resolved?	Is the system safe to operate?	Safety Assessment Status
	Process Efficiency - Speed Security - Safety		How long does it take to successfully complete cybersecurity audit/penetration testing? Are security vulnerabilities identified and addressed proactively?	Is the system cyber-resilient?	Cybersecurity Test Duration
	Process Efficiency - SpeedSecurity - Safety			How long does it take to receive ATO approval for new releases?	Time to Certification and Authority to Operate (ATO)
	Process Efficiency - Speed	Is the flow of work (stories) moving forward through the value stream? Is the flow of work as efficient and predictable as needed?	Is the flow of work (features, capabilities) moving forward through the value stream? Is the flow of work as efficient and predictable as needed?	Are the evolving stakeholder needs being met when needed?	Committed vs. Completed Cumulative Flow Capacity
	Process Efficiency - Speed	Is the team performing as expected? How much work can be accomplished by a	n/a	n/a	Team Velocity Acceleration

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
		team in a future iteration?			
	Process Efficiency - Speed		How long does it take to deploy an identified feature/capability?	How responsive is the program to change?	Cycle Time / Lead Time Release Frequency
	Process Efficiency - Speed		What is the cadence of product release or deployment? How long does it take to release a minimum viable product?	What is the cadence of product release or deployment? How long does it take to release a minimum viable product?	Release Frequency MVP Release Duration
	Process Efficiency - Speed		How much time does it take to conduct a full regression test? How much time for the automated regression test?		Test Duration Automated Test Duration
	Process Effectiveness		How much of the testing is automated? How often do we perform automated testing? How much capability is tested in an automated fashion?	How much of the system testing is automated? How much of user test is automated? How often do we perform automated testing? How much of system automated test is credited for user test?	Automated Test Coverage Automated Test Frequency
	Process Effectiveness - Value		What is the product value (normalized feature/capability delivered by effort)?	What is the product value (normalized feature/capability delivered by effort)?	Acceleration

Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures
			Is productivity improving over time?	Is productivity improving over time?	
	Process Effectiveness	Is the work in progress being managed appropriately?	Is the work in progress and product backlog being managed appropriately? Are there queues or delays in our process workflows that prevent us from optimizing throughput?	Are there (major) queues or delays in our process workflows that prevent us from optimizing throughput?	Cumulative Flow Defect Resolution Backlog Readiness
Customer Satisfaction	Customer Support			Is the user satisfied with the delivered products? Do they provide the desired functionality when needed?	Value Assessment

8. Measurement Specifications

8.1 Automated Test Coverage (Product or Enterprise Measure)

Measure Introduction	
Description	In an iterative development approach, it is important not only to efficiently verify new features but to ensure prior functionality is not impacted. Doing so manually can be time-consuming. Typically, code coverage is verified primarily in structural (white box) testing at the unit level, and requirements are verified primarily in functional/system test. Efficiency and throughput can be enabled by automated test suites executed at multiple levels (unit level, functional level, regression testing). The extent to which automated testing is implemented is a business decision depending on objectives and constraints, such as velocity, quality, and cost vs. benefit. It may not be feasible or desirable to automate all testing. Projects may set planned test automation objectives, such as 70%-80% coverage based on their cost benefit analysis. Often these automated test suites are integrated directly in the code pipeline and invoked upon each code commit and build, or in nightly regression test batch jobs. (Refer to Figure 2 for context.) Test results (tests passed, tests failed) can be distributed automatically in email so anomalies impacting the code quality and pipeline can be quickly identified and resolved.
Relevant Terminology	Functional Testing: Testing against the requirements or function of the software, without considering the internal implementation. Sometimes termed black box testing. Structural Testing: Testing the internal structure, design, implementation, or logic of software, such as paths, conditionals, or branches through the code. Sometime termed white box testing.

Information Need and Measure Description	
Information Need	How much of the testing is automated? How many tests have been validated and approved? How much credit is given in formal test (e.g., DT/OT) for automated test?
Base Measure 1	Total Requirements *[integer > 0]*
Base Measure 2	Requirements Tested *[integer ≥ 0]*
Base Measure 3	Requirements Tested Through Automation *[integer ≥ 0]*
Base Measure 4	Requirements Tested Manually *[integer ≥ 0]*
Base Measure 5	Code Constructs (e.g., classes, conditionals, files, lines, packages) *[integer > 0]*
Base Measure 6	Code Constructs Tested by Automated Test *[integer ≥ 0]*
Base Measure 7	Automated Test Cases Passed *[integer ≥ 0]*
Base Measure 8	Automate Test Cases Failed *[integer ≥ 0]*
Derived Measure 1	Requirements Not Tested = (Total Requirements) – (Requirements Tested Through Automation) – (Requirements Tested Manually) *[integer ≥ 0]*
Derived Measure 2	Percentage Requirements Tested Through Automation = (Requirements Tested Through Automation) / (Total Requirements) * 100 *[percentage]*
Derived Measure 3	Percentage Requirements Tested Manually = (Requirements Tested Manually) / (total requirements) * 100 *[percentage]*
Derived Measure 4	Percentage Requirements Not Tested = (Requirements Tested Not Tested) / (total requirements) * 100 *[percentage]*

Derived Measure 5	Percentage Code Constructs Tested = (Code Constructs Tested by Automated Test) / (Code Constructs) * 100 [percentage] *(for each code construct) [percentage]*

<table>
<tr><th colspan="2">Indicator Specification</th></tr>
<tr><td>Indicator Description and Sample</td><td>
Figure 9 depicts the percentage of project requirements that are verified by automated vs. manual testing over time. In this example, the project set a planned objective for 70% automation, and ultimately met and exceeded that objective. Percentages are used rather than absolute values to facilitate comparisons across projects, but with the total number of requirements plotted on the secondary axis for considering scale and complexity of the test automation effort. Note also that the Automated Test Coverage will change over time as new requirements are added, and tradeoff decisions can be made on the benefit of investing further program effort to develop new automated test cases to increase coverage. This may include estimating the net impact on program throughput, quality, or cost.

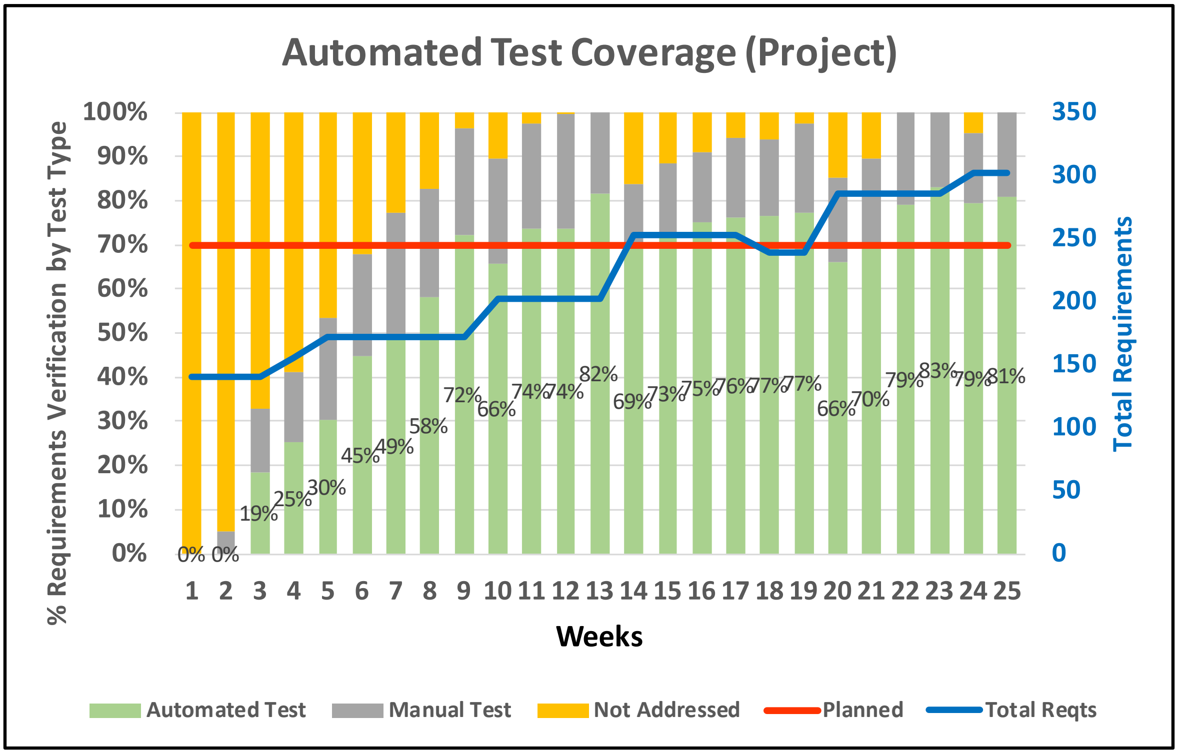
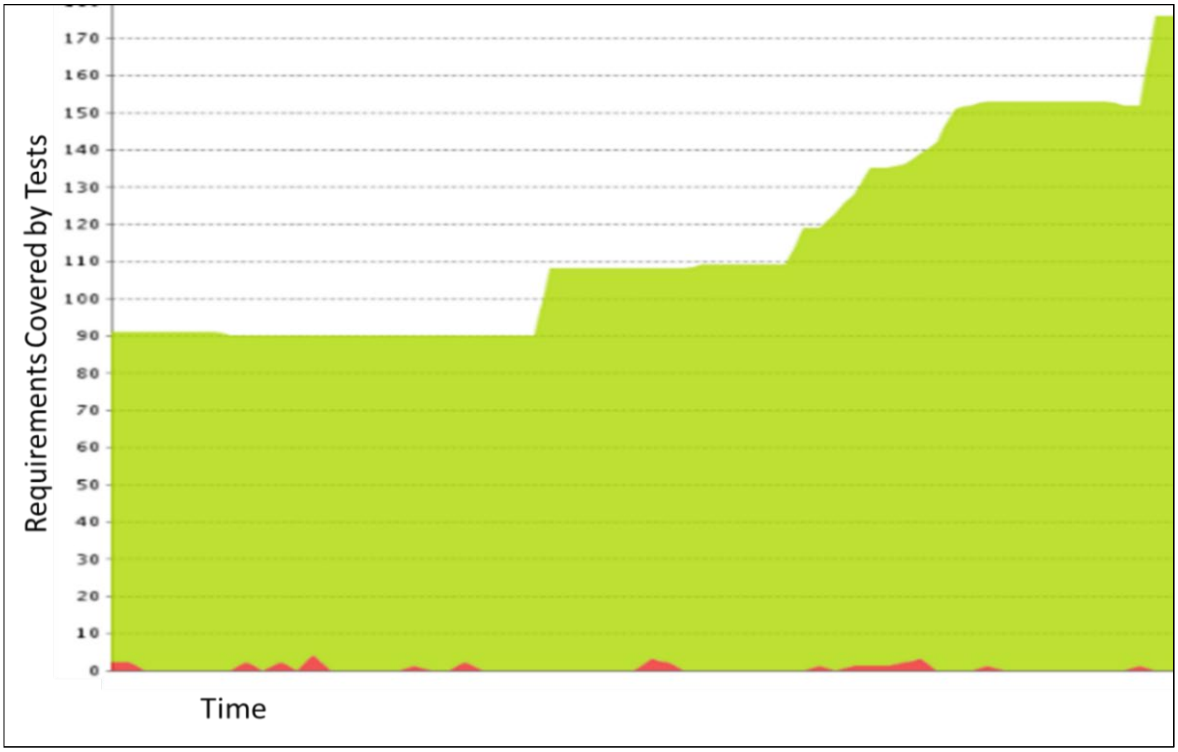


Figure 9: Automated Test Coverage (Project Level)

At project startup an initial requirement set is established that evolves iteratively (with additions, modifications, deletions) across the project life based on collaboration with the product owner and other stakeholders. Test cases (automated and manual) are developed to verify requirements as they are implemented. By iteration 9, the automated test suite is verifying over 70% of requirements, supplemented by manual test cases that verify nearly all project requirements. In iteration 18, the product owner deleted a capability from the backlog and requirements count was reduced. Over time, additional automated tests are developed that increase automated coverage while reducing the dependence on manual testing, although both are supplemented regularly as new requirements are added. The project has sustained its automated test suite to generally meet the project objective of 70%-80% automated test coverage.
</td></tr>
</table>

	Effectiveness of automated testing should be monitored. The pass/fail success status of automated tests is often available from automated test tools, as illustrated below in Figure 10, so anomalies breaking the code pipeline can be quickly detected and resolved. The quantity of requirements covered in automating testing is depicted in the amplitude. Requirements that failed an automated functional test are shown in red, indicating quality of the pipeline over time. Some tools may also provide additional information, such as Yellow for the requirements that were skipped, or the requirements with no automated test.  **Figure 10: Automated Test Pass/Fail Status** This automated report from the program test tool indicates a low number of requirements (<5) over time that failed automated testing. All test failures are investigated. Some of the test failures are due to enhancing the automated test scripts to verify new requirements as they are added, others are the result of regression test failures where baseline product functionality was impacted by new enhancements, but this quickly stabilizes as the product development baseline matures.

The extent of code structural coverage from automated (white box) testing can increase confidence in development baseline quality. In Figure 11 test coverage is collected for each build and depicted by trends for % coverage of structural code constructs (classes, conditionals, files, lines, packages). The extent of coverage can indicate the risk or confidence in code quality, suggest a need for additional testing, or the potential risk of incurring defect escapes.

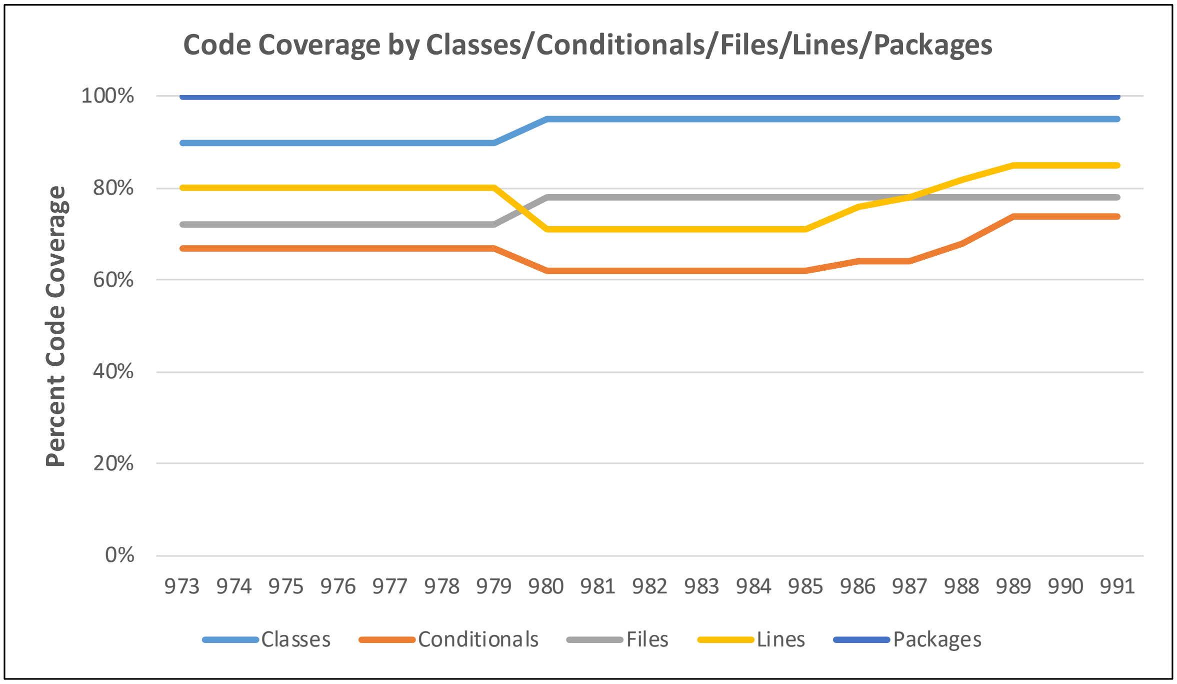


Figure 11: Code Coverage from Automated Testing

100% of packages and 95% of classes are addressed by automated tests. 85% of the code (lines of code) and 75% of branches are currently exercised; coverage dropped in iteration 980 (to 70% of code, 65% of branches) as new functionality was added, but has continued to grow in subsequent releases as the automated test suite was expanded to address these enhancements. The project has set a target for ≥ 80% of code and branches exercised in automated testing, so the test suite is being enhanced for additional logic test cases focusing on the most risky or complex modules.

Indicator Description and Sample (continued)

At the enterprise level, the extent of automated testing utilized across projects can be monitored, as reflected in Figure 12. The enterprise may set business objectives for the extent of automated testing across projects (e.g., 70%), subject to project-specific characteristics and constraints.

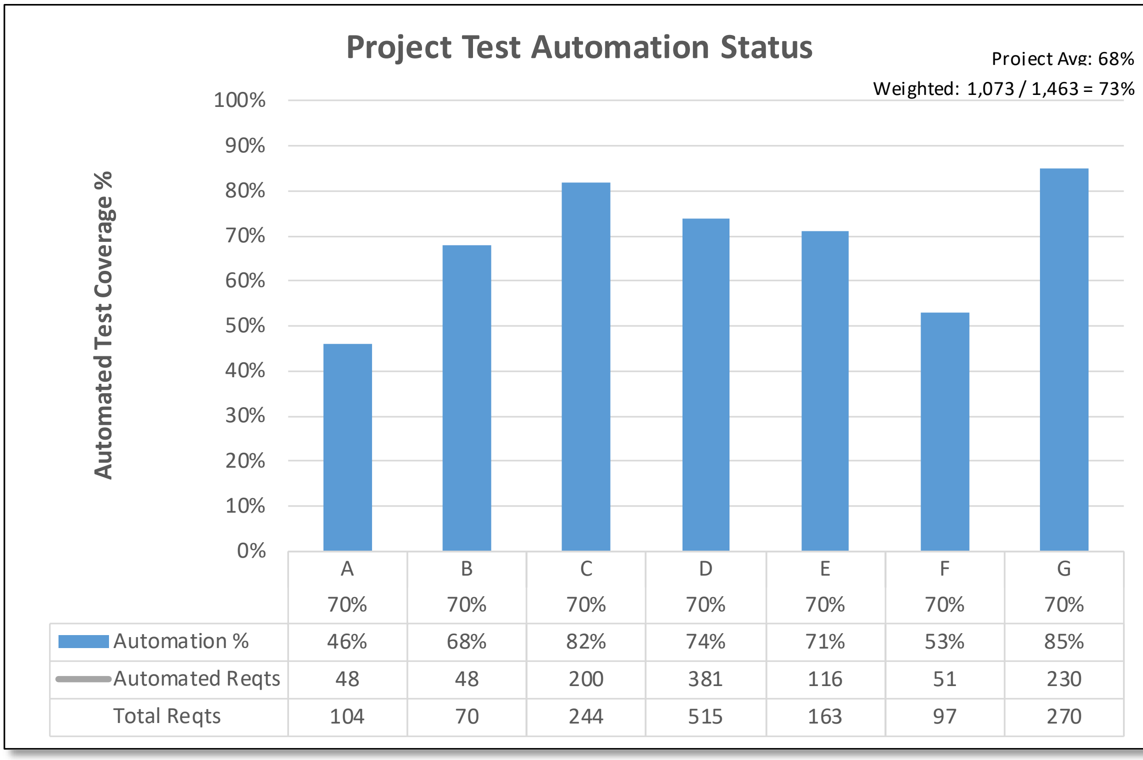


Figure 12: Automated Test Coverage (Enterprise Level)

Automated test coverage percentages are collected from projects and aggregated at the enterprise level to monitor the success of implementing automated testing. Measures are displayed for each project in both relative (%) and absolute terms (Requirements Verified). Absolute values are used for context in evaluating the overall impact of the project automated test coverage; larger projects may have greater challenges in scope but also more resources available to realize the benefits of automation. Some projects are early in their development cycle and development of automated test cases are still in work. Overall, the project average is 68% automation, but when weighted by the number of requirements verified the coverage is 73% due to the higher impact from larger projects. Analysis and actions at the organizational level will depend on the characteristics of the individual projects, the extent to which performance and quality measures are impacting objectives, and the extent to which they may be positively impacted by investing in additional automation.

Analysis Model	Automated Test Coverage (Project Level): • What percentage of functional requirements are verified with automated testing? Is each requirement fully covered by the automated testing, or are some aspects not verified? • Any requirements not verified automatically must be verified manually, which can impact productivity, schedule, and resources. Apply decision tradeoffs for the cost vs. performance benefit of investing effort to expand the extent of automated test coverage. Automated Test Pass/Fail Status: • Are automated tests completing successfully, or are there anomalies impacting the code pipeline that should be investigated? • Automated tests are typically conducted regularly as part of the code and unit tests in the code development pipeline, such as upon each code commit or in nightly regression tests. Summary test reports can be automatically generated and distributed by the automated test tools. 100% success of automated tests passing is often a criterion for advancing the code baseline to production. Discrepancies could be in the code, or in the test cases themselves, but either should be investigated. Code Coverage from Automated Testing: • How much of the code structure is covered by the automated test suite? Which parts of the code are not covered (e.g., any safety critical code, interfaces, interoperability requirements)? • Code coverage is a tradeoff between investment, risk, and return; although 100% coverage may be desirable, that might not be practical within available environments, resources, interfaces, and constraints. Automated Test Coverage (Enterprise Level): • What is the extent of automated testing conducted across the organization's projects? What benefits to organizational performance (e.g., cycle time, quality, throughput) are enabled by effective automated testing? Automated testing is a primary enabler for achieving efficiency, quality, and cost savings at both the project and organizational levels. Organizations should monitor automated test measures in relation to achievement of their desired performance objectives.
Decision Criteria	Automated test coverage alone is not an objective; it is the associated gains in accelerating performance and improving product quality at the project and organizational levels that make investments in automation worthwhile. Automation measures should be evaluated in the context of other performance measures, such as those defined elsewhere in the PSM CID measurement framework. Industry experience suggests that automation in the range of 70%-80% is often beneficial in producing improved performance outcomes, but this may vary by domain or application. If automation measures are lower than planned, or if there are process effectiveness or product quality issues that are impacting objectives, consider root cause analysis and decision tradeoffs to assess the impact and determine if they can be improved by further investments in test automation.

Additional Information	
Additional Analysis Guidance	Test automation and coverage are key elements of achieving faster and more comprehensive releases with higher code quality. These should be used in conjunction with quality measures to ensure the adequacy of testing and achieve acceptable, inherent quality levels. A reasonable goal is to achieve near instantaneous automated test results with acceptable quality. Testing efficiency and speed are closely related to achieving other performance measurement objectives such as lead time, cycle time, and release frequency. Robustness of the testing conducted should also be considered (e.g., stress testing, boundary conditions on valid data inputs). Additional project performance measures, such as effort, schedule, and cost, can be correlated with automated test coverage measures to evaluate the performance benefits (e.g., cost savings, productivity, quality) achieved through automated testing.

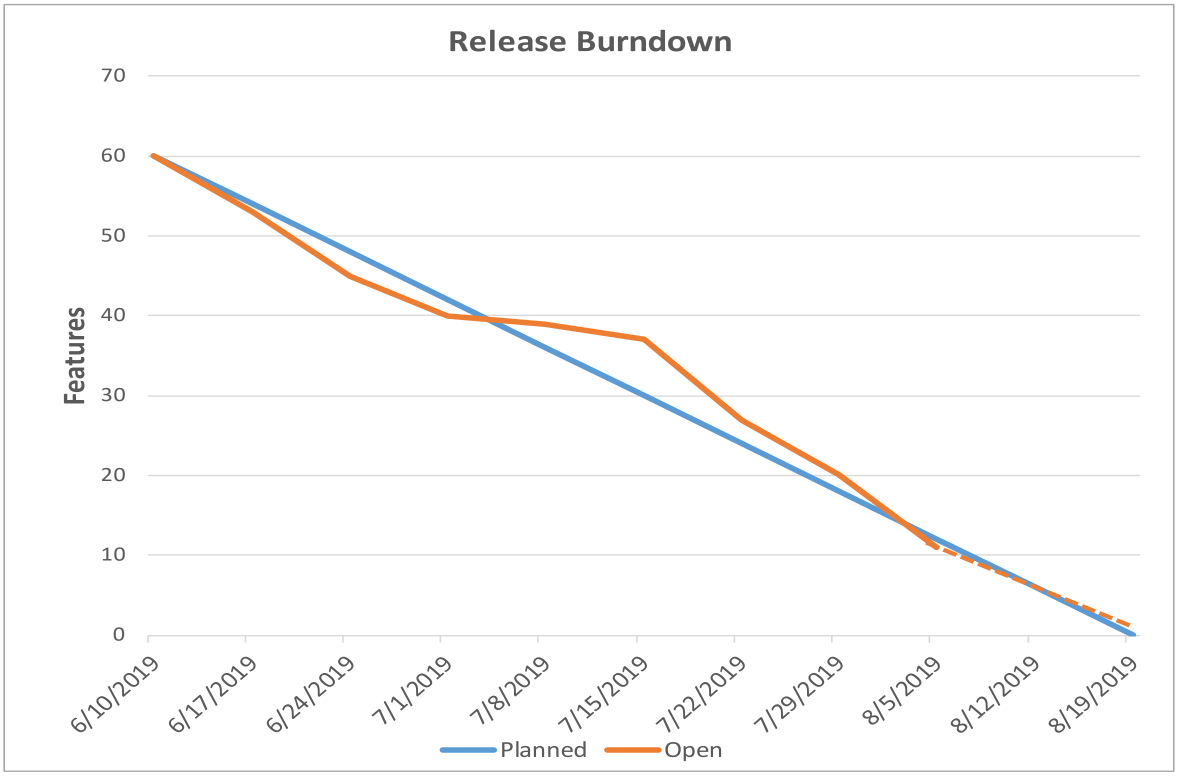
Implementation Considerations	Measures for code coverage and requirements coverage are directly available from many automated development tools commonly integrated across the tool chain. However, the emphasis should be on thorough testing sufficient to ensure product quality rather than achieving high code coverage numbers. Code coverage is an important factor, but by itself, is not sufficient to ensure product quality. Automated test cases could focus on areas of high risk, complexity, or dependencies where repeatability or regression testing are important factors, especially in the near term. Relying solely on automated test tools and scripts may not be wholly sufficient to exercise all functionality needed (e.g., user interfaces, databases). It may be necessary to supplement automated test scripts with manual effort to execute additional test cases and validate that the automated test is sufficiently representative of the overall functionality. Automated testing may be conducted at various or multiple points in the workflow, for instance before or after the baseline merge. A best practice is to execute automated test suites nightly or as part of the pipeline following each code commit. For existing systems, the enterprise will need to make a business decision as to whether it is worth the investment to develop automated tests. This will be dependent on the necessary infrastructure to support automated test, the expected lifecycle of the system, the level of updates/regression test typically required, etc. Automated test scripts are a valuable work asset that should be sustained in a manner similar to source code. Test scripts may need to be enhanced or refactored as the product evolves.

Additional Specification Information	
Information Category	Process Performance (Process Effectiveness)
Measurable Concept	Process Effectiveness
Relevant Entities	System, Test cases
Attributes	Amount tested, amount automated tested
Data Collection Procedure	Data is typically collected by automated tools upon execution of test scripts as part of standard pipeline workflows. Results are recorded in team tracking tools. Summaries of test results and coverage can often be provided automatically nightly or upon completion.
Data Analysis Procedure	Data is reviewed and analyzed to ensure adequate quality for each candidate product. Discrepancies in process effectiveness, product quality, or test coverage not meeting threshold targets may indicate updates to code or test scripts are necessary.

8.2 Burndown (Team, Product, or Enterprise Measure)

Measure Introduction	
Description	Burndown is used to monitor completed work items (e.g., stories, features, capabilities) vs. planned work items for an iteration, release, or capability. Work items may include design, code, test and all supporting activities (e.g., requirements development, configuration management and quality engineering). Progress toward completing planned work is depicted graphically to provide an indicator of the likelihood of meeting planned goals.
Relevant Terminology	See Section 3: Ontology and Definitions.

Information Need and Measure Description	
Information Need	What is the status of the iteration, release, or capability? Will all the remaining committed work be completed as planned? What are the features/capabilities at risk of not being completed as scheduled? What are the trends in execution relative to plan?
Base Measure 1	Planned Work (integer scale) (e.g., Story Points/Features/Capabilities)
Base Measure 2	Completed Work (integer scale) (e.g., Story Points/Features/Capabilities)
Derived Measure 1	Open Work = Planned Work - Completed Work (e.g., Story Points/Features/Capabilities)

Indicator Specification	
Indicator Description and Sample	In Figure 13, the orange line represents the number of open features over time, while the blue line indicates the planned burndown.  **Figure 13: Release Burndown** At release planning, work items representing 60 features were committed. While little progress was made during the first week to a planned training event, the teams recovered and is still projected to complete the planned work by the end of the release.
Analysis Model	At the team level, the focus is generally on stories or story points open through the iteration. Is the team completing the committed work items? Are they significantly behind or ahead of the burndown plan? Are items blocked? What is the likelihood of meeting the commitment on time? Can additional backlog stories be brought into the iteration? Are teams improving execution over time? At the product level, the focus turns to features or capabilities across releases. At the enterprise level, the focus is generally on capabilities for external releases.
Decision Criteria	At the team level, lack of progress (e.g., not reducing open story points at all over several days) and variances from the plan (e.g., 5%) should be reviewed for action by the team. Data is generally not shared externally to the team. At the product level, variances of over 10% are reviewed for causes of roadblocks and consideration of replanning.

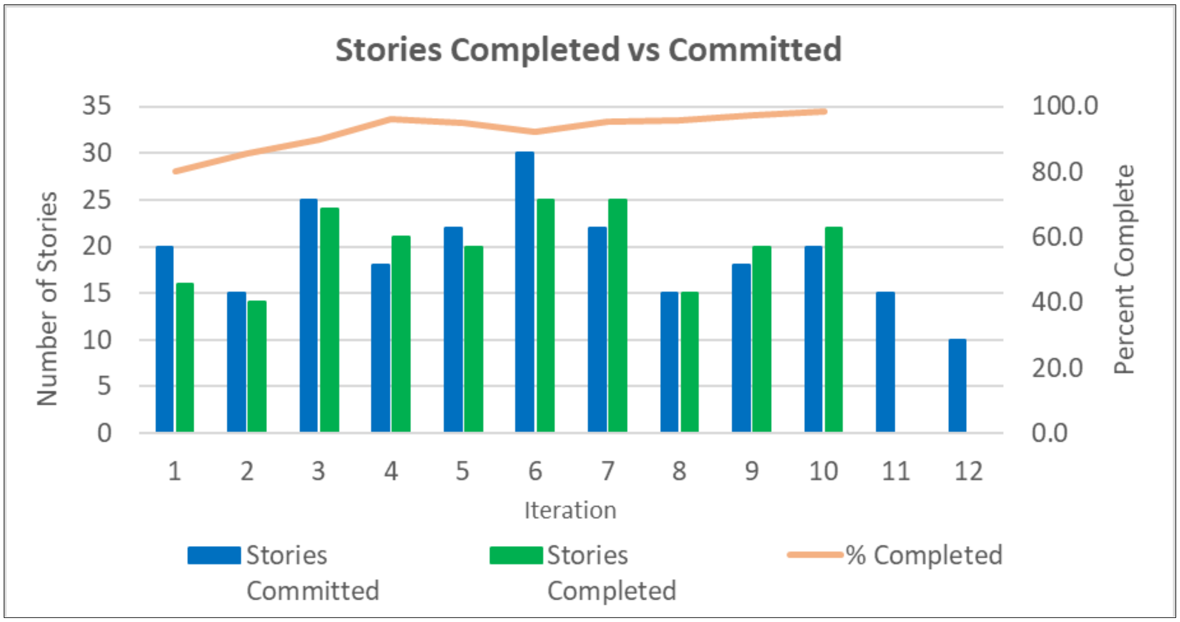
Additional Information	
Additional Analysis Guidance	Use this metric with the velocity metric and other work unit progress metrics (e.g., test progress, cumulative flow). The velocity metric supports the planned story points for each iteration. The actual completed story points from the iteration is an input to the velocity metric. Review with other work unit progress metrics may support an assessment of overall risk and may impact prioritization of work for future iterations. Consider bounds of estimated burndown based on historical performance, e.g., best case, worst case, Monte Carlo analysis.
Implementation Considerations	Some teams may use hours instead of story points (or may map story points to hours).

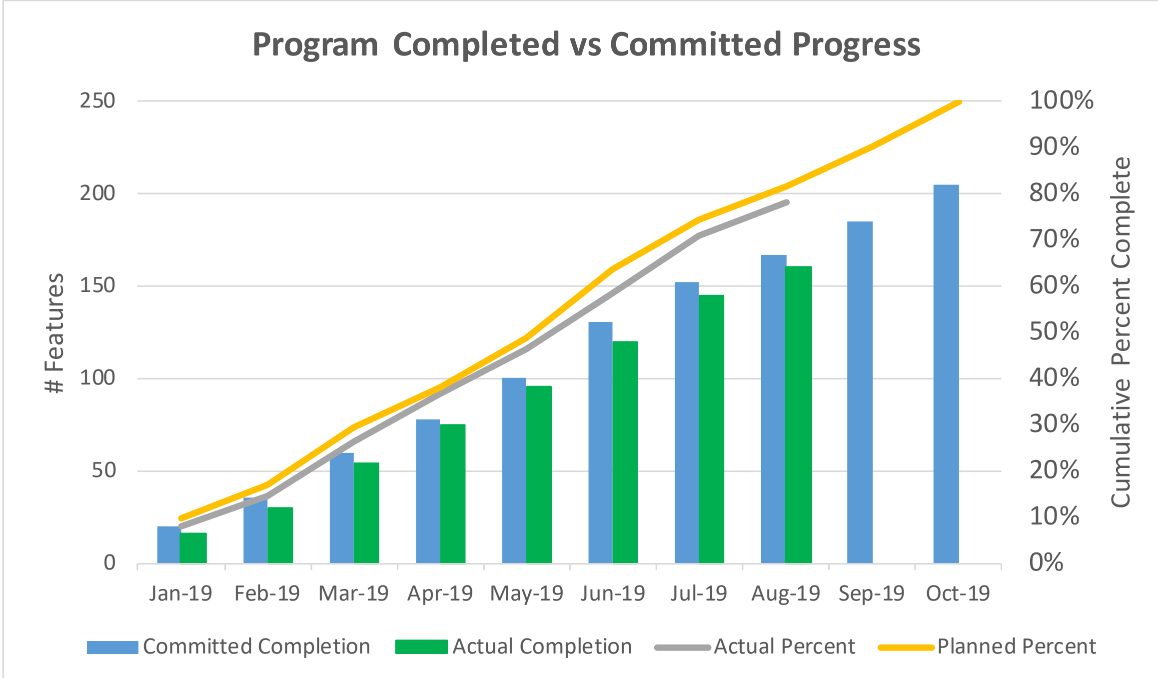
Additional Specification Information	
Information Category	Schedule and Progress
Measurable Concept	Work Unit Progress
Relevant Entities	Product
Attributes	Story Points, Features, Capabilities
Data Collection Procedure	At the team level, story points committed for each iteration are determined at the iteration planning meeting. This value is determined from the velocity metric. Based on the average velocity and other factors (e.g., vacations), the team commits to a number of story points for the next iteration. Work items (e.g., stories, tasks) are selected to match this commitment. Work items are closed when completed and meet their evaluation criteria, and burndown progress is updated daily. At the product level, the features and capabilities committed for each release are determined during release planning. Commitments may be replanned as work is completed and priorities change.
Data Analysis Procedure	For the team, Burndown is analyzed daily for progress/risk and at the end of each iteration to determine if the story points were delivered as committed. The final story points completed value is an input to the velocity metric. For the project, Burndown is analyzed periodically (e.g., monthly, quarterly, by release). For the enterprise, Burndown of capabilities for major events is analyzed.

8.3 Committed vs Completed (Team, Product, or Enterprise Measure)

Measure Introduction		
Description	Committed vs Completed is a measure of progress toward completing planned, or expected, features and capabilities. At the team level it may be used to measure progress of each iteration. At the program or organizational level, it can be used to measure overall progress toward a release and completing product development. It may also be used to measure quality of the product by indicating product readiness with respect to expected capability, or functionality.	
Relevant Terminology	Stories Committed	Stories the team has committed to complete within an iteration.
	Features, or Capabilities, or Committed	Features and capabilities committed to the customer by the program to be included in the product.
	Completed Stories, Features, or Capabilities	Stories that have completed their level of verification and validation and have been proven to work as expected.

Information Need and Measure Description	
Information Need	Are Stories, Features, or Capabilities delivered as committed? What are the Stories/Features/Capabilities at risk of not being completed as scheduled?
Base Measure 1	Work Items Committed Each Iteration (integer) (e.g., stories, story points)
Base Measure 2	Work Items Completed Each Iteration (integer) (e.g., stories, story points)
Base Measure 5	Work Items Committed Each Release (integer) (e.g., features, capabilities)
Base Measure 6	Work Items Completed Each Release (integer) (e.g., features, capabilities)
Derived Measure 1	Percent Work Items Completed = (Sum of All Work Items Completed) * 100 / (Sum of All Work Items Committed) for a desired iteration, release, or program (e.g., stories, story points, features, capabilities)

Indicator Specification	
Indicator Description and Sample	In Figure 14, Stories Committed is graphed as a column for each iteration [blue bar] and stories Completed as a column for each iteration [green bar]. Cumulative Percent Stories Completed are also graphed as a line chart across iterations (secondary axis). The indicator may be aggregated for a release, set of features, capability, or a complete project to provide progress toward product completion.  **Figure 14: Stories Completed versus Committed** Iterations 1, 2, 3, 5 and 6 did not complete expected stories. During iterations 1 and 2, the team was forming and learning to work together. Iteration 3 completed close to expected stories. Iterations 4, 7, and 9 completed above expected stories. The team was working together and attempting to catch up on the backlog of stories. This could also reflect rework that was being identified and resolved. Current percent complete does not indicate a need for a re-plan but progress and velocity should be watched to ensure the team can complete the remaining backlog over the next two minor iterations.
	Figure 15 shows a product level view of completion, for Features Committed. Monthly data is graphed, along with a cumulative percentage complete.

	 **Figure 15: Program Completed versus Committed** The month to month cumulative view of the same data shows the project is not completing committed features creating a backlog of work. The gap between planned percent complete and actual percent complete is increasing slightly and is behind target to complete all features by project end. Some corrective actions may be needed.
Analysis Model	Is the team and project completing the assigned work? Will they deliver required features within allocated project schedule? Teams may not complete all Stories for each iteration, so this indicator provides information about any backlog of features growing as you progress through the release or program.
Decision Criteria	If the gap between committed and actual completion is more than 5%, than the team should investigate causes of lack of completions. If any team is more than 10% behind commitments, than the project management should investigate and consider corrective action. If the product completion is more than 10% behind commitments, than alternative courses of action (e.g., adding additional teams or changing commitments) should be considered.

Additional Information	
Additional Analysis Guidance	Use this with the Committed Backlog, Burndown, and Velocity to ensure project will release identified features (or capabilities) as scheduled. The project may want to use different levels of aggregation to view the progress at different levels to expose any adverse trends. If a story is not completed within its expected iteration, it will be placed back on the backlog and re-prioritized for a future iteration. If a team completes assigned stories for an iteration with additional time to work, they should select additional stories from the backlog. Stories, Features, or Capabilities may be weighted by complexity to give a more complete view of program completion.
Implementation Considerations	In general, Committed vs Completed Stories is specific to a team since story point size may vary from team to team. An aggregate measure at the Feature or Capability level can be compiled across teams and compared to capability roadmap to see if project is completing multi-team capabilities within project expectations.

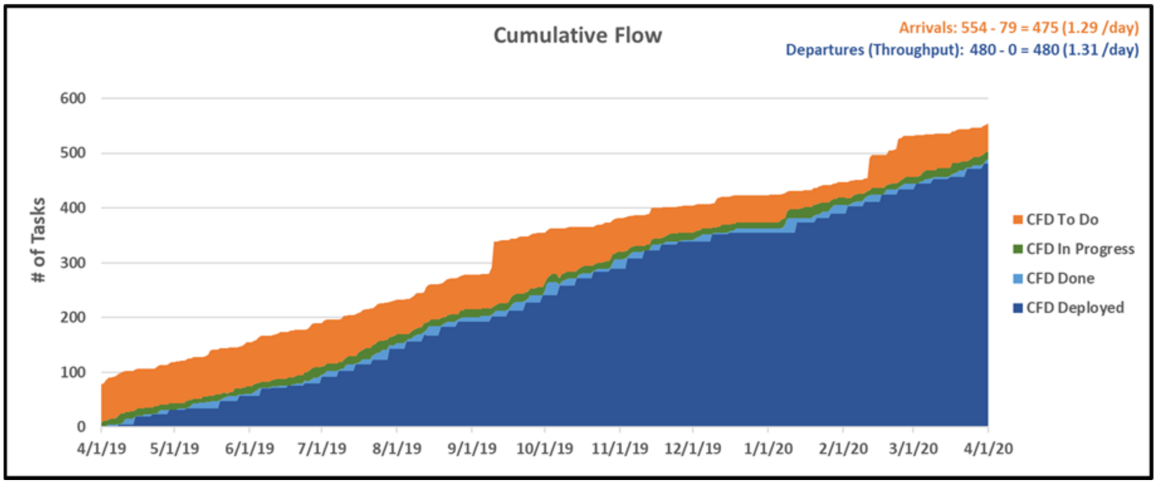
Additional Specification Information	
Information Category	Schedule and Progress
Measurable Concept	Work Unit Progress
Relevant Entities	Stories, Features, or Capabilities
Attributes	Story Points (estimated size), Iteration Committed, Iteration Completed for each entity
Data Collection Procedure	For team measure, data is collected at the end of each iteration by the team lead from the team tracking tool. Story Points must be tested and satisfy "Done" criteria, with no open defects to be counted as completed. If a Story does not satisfy "Done" criteria, then it is not considered "Complete" and its Story Points are not included in the total of Completed Story Points. For product or enterprise measures, data is collected periodically (e.g., monthly, quarterly, end of each iteration or release).
Data Analysis Procedure	Data is analyzed at the end of each iteration by the team during the iteration review and considered during the planning session for the follow-on iteration. The data is also aggregated and analyzed at summary levels across iterations or releases to ensure the program is completing its committed capabilities.

8.4 Cumulative Flow (Team, Product, or Enterprise Measure)

Measure Introduction	
Description	Cumulative flow is a tool to visualize work in progress, cycle time and throughput. In this specification, the indicator (Cumulative Flow Diagram) is described, with base and derived measures that duplicate other measures listed above. Continuous iterative development (CID) methods are focused on the delivery of capabilities/features achieved by managing the flow and throughput of work through a process. Understanding and managing flow is fundamental to achieving stable processes with predictable performance and the efficient use of resources.  Flow is visualized and represented graphically in a Cumulative Flow Diagram (CFD) depicting the total quantity and transition of work items in each workflow state over a time period. It is generally desirable that the amount of work distributed across each process workflow state is in balance (new work is equivalent to the completion of work in each workflow state). This can be visualized on a CFD as roughly parallel upper and lower bounds of the cumulative work through each state. Failure to match departures and arrivals for each state can result in queues, backlogs, or inefficiencies in the progress of work completion or utilization of resources. Adherence to effective processes ensuring standard CFD assumptions, rules, and constraints, can help teams achieve predictable performance. Reference: Actionable Agile Metrics for Predictability (Vacanti, 2015)
Relevant Terminology	Cumulative Flow Diagram: A tool used in queuing theory showing whether the flow of work is consistent; visually points out shortages and bottlenecks. Throughput: The number of work items completed per unit time. Work in Progress (WIP): The number of work units in progress between workflow steps in a process. Work Items: Item that indicates the type of work and what needs to be done (e.g., tasks, stories, features, capabilities). It may include the target date for completion.

Information Need and Measure Description	
Information Need	Is the flow of work moving forward through the value stream (through the process work flow states)? Is the throughput of work predictable? Are there queues or delays in our process workflows that prevent us from optimizing throughput?
Base Measure 1..N	Base Measures 1-N: The number of work items in each of N workflow states. Collected using counts or times. Note: These states vary by project, organization, or defined process. For the example indicators below, the workflow states used include: • To Do: Work items from the product backlog that have been approved/accepted for implementation (committed to), but not yet started. They generally have been assigned to an iteration or release. The product backlog may also include items that are never implemented. To best depict flow, CFDs do not typically include Backlog work items. • In Progress: Work items that have been approved/accepted for implementation (committed to) and have started development. • Done: Work items have completed all development activities in an iteration and are ready for internal release. • Deployed: Work items have completed all development activities defined by the process, including integration and test activities, and are deployed in an internal or external release.

Derived Measure 1	Approximate Average Cycle Time = average duration for all completed work items Note: The duration is an approximate based on the set of completed work items for a given time range. It is not based on an average of individual work item durations. See Cycle Time / Lead Time specification for a measure based on individual work item durations. - Other derived measures for transitions between workflow states can be calculated similarly.
Derived Measure 2	Throughput = average of Work Items Done per unit time
Derived Measure 3	Work in Progress = average of Work Items in Progress per unit time

Indicator Specification	
Indicator Description and Sample	Flow is commonly depicted in a Cumulative Flow Diagram (CFD), Figure 16, depicting the stacked cumulative quantity of process arrivals, departures, and WIP in bands for process workflow states over time, as illustrated in the example below. The amplitude of the CFD chart indicates the amount of work in each workflow state.  **Figure 16: Cumulative Flow Diagram** This example CFD indicates a project workflow with a team capacity that is well balanced with demand. The number of tasks in each workflow state (height of the bands, or vertical distance between lines) is holding fairly steady and narrow, with relatively parallel lines (slopes) indicating a balance of work arrivals (added to the top orange, To Do, Band) transitioning smoothly into subsequent work flow states culminating in the bottom dark blue, Deployed, band. There are no notable queues, delays, or backlogs (widening CFD bands), except for the arrival of new needs and objectives from the customer in September and March. These are reflected in the Release Backlog (increases in the height of the orange To Do band). These were steadily worked off and implemented by the project team at its consistent rate and capacity (indicated by maintaining fairly stable slopes of the In Progress, Done, and Deployed lines). Throughput rate is steady with no significant changes, except for a short flattening of the progress curves over the December holiday period, that resumed quickly when the team returned to full staffing in January. This workflow balance over the year shown is sustantiated further by an average task departure rate (1.31 tasks/day), well matched to demand reflected in the average arrival rate (1.29 tasks/day).

Indicator Description and Sample (cont.)

For projects adhering to standards for collection and reporting of CFD data, derived measures for average WIP, average Throughput, and approximate average Cycle Time are related by Little's Law (as discussed in *Actionable Agile Metrics for Predictability*). Generally, these summary cumulative measures can be derived and visualized for a given time range from a CFD diagram as in the abstraction shown in Figure 17. The figure below further illustrates these relationships

Little's Law:
CT = WIP/TH
TH = WIP / CT
WIP = CT * TH

Avg WIP: Tasks in Work (cum) / duration
Avg Throughput: Tasks Done (cum) / dur
Avg Cycle Time: WIP / Throughput (dur. / task)

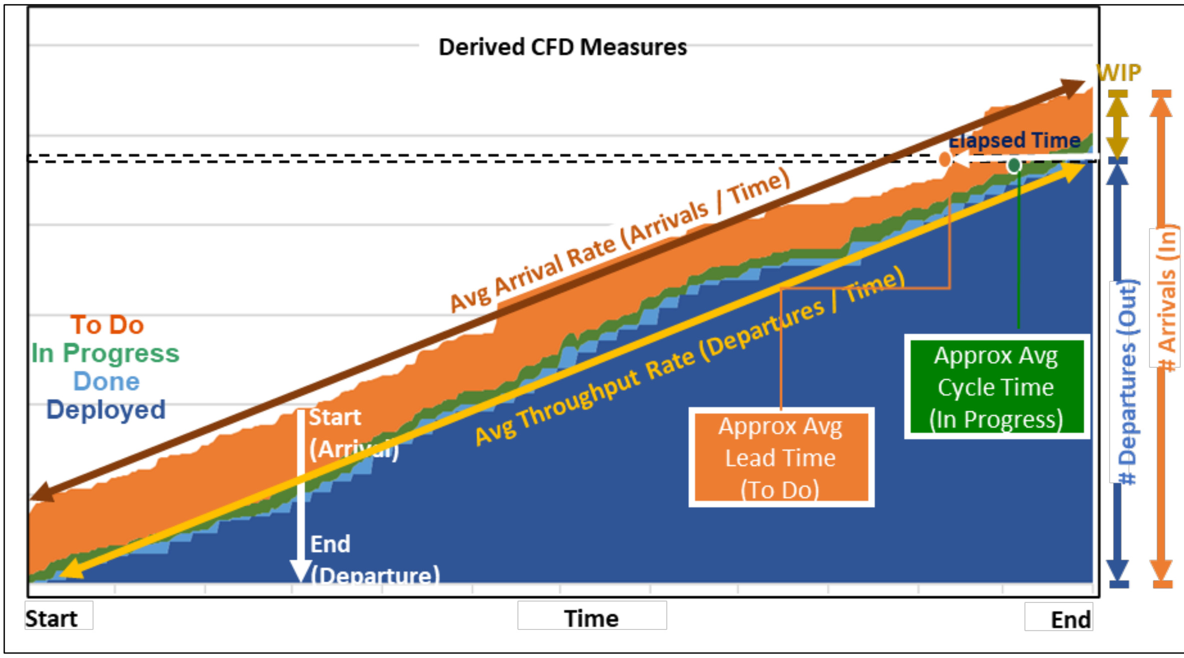
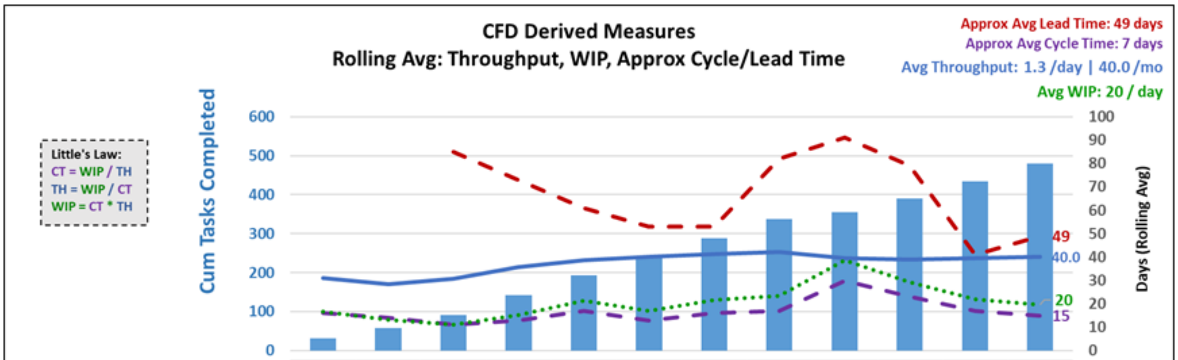


Figure 17: Notional CFD Diagram

Continuing from the above project CFD example, the project average WIP, average Throughput, and approximate average Cycle Time can be calculated and plotted over time, as in Figure 18.



	5/1/19	6/1/19	7/1/19	8/1/19	9/1/19	10/1/19	11/1/19	12/1/19	1/1/20	2/1/20	3/1/20	4/1/20
Departures (CFD Deployed)	31	57	92	143	193	240	289	338	355	390	434	480
Approx Avg Lead Time (Days)			85	73	61	53	53	82	91	79	41	49
Approx Avg Cycle Time (Days)	16	14	11	13	17	13	16	17	30	23	17	15
Avg Throughput (per mo)	31.0	28.5	30.7	35.8	38.6	40.0	41.3	42.3	39.4	39.0	39.5	40.0
Avg WIP CFD (per day)	17	13	11	15	21	17	22	24	39	29	22	20

Figure 18: Workflow by Period and Rolling Average

	This example provides further numeric substantiation of process effectiveness consistent with the CFD indicator analysis. Derived CFD measures for average WIP, average throughput, and average cycle time indicate fairly stable performance over time that could be useful in predictably planning future estimates. Approximate Lead Time (turnaround for implementing and deploying accepted customer requests) has reduced on average over the last year, even considering the two significant spikes in receipt of new requests and the short delays in throughput over the December holidays. Note that although CFD measures may indicate stable and consistent workflow process performance, this does not necessarily imply this level of performance fulfills the business need. Process improvements and performance efficiencies may yet be needed to meet the Voice of the Customer. Also note these measures may be specific to the team (e.g., methods for defining tasks, stories, story points) or application domain (e.g., embedded firmware, command and control, information systems, high reliability space applications), so organizations should be cautious about projecting performance across other projects. It may be most beneficial to monitor overall workflow trends and potential areas of concern rather than focusing on absolute measures.
Analysis Model	Is work arriving and being completed at consistent rates? Is there a steady proportionate ratio of WIP across workflow states, or are there queues, delays or inefficiencies indicated by widening CFD bands that should be addressed? The shapes of CFD bands indicate if the flow of work is being processed and completed at predictable steady rates (e.g., consistent slopes with relatively parallel bands). Other shapes (e.g., diverging bands, flat lines, S-curves) can indicate inefficiencies, mismatched arrivals and departures, or delays in completing the flow of work. Is cycle time and throughput compatible with achieving the project plan and product roadmap? Are these measures stable? Comparing derived average cycle time against actual calculations (see Cycle Time/Lead Time specification) can indicate potential process anomalies, such as giving preferential priority to certain tasks. What can be done to increase throughput or reduce WIP, if necessary, to meet performance objectives? Additional details of CFD derived measures and related topics such as technical debt are beyond the scope of this specification and are described further in referenced materials.
Decision Criteria	Significant variations (e.g., ± 10%) in the slope or width of CFD workflow band curves may indicate performance issues, queues or delays in bringing work to closure. Root causes should be analyzed, and corrective actions implemented as appropriate to bring workflow back within expected ranges needed to execute the plan.

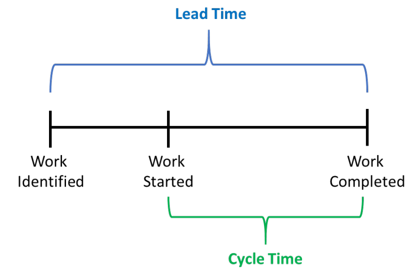
Additional Information	
Additional Analysis Guidance	Anomalous CFD band shapes indicating potential delays or negative trends in WIP, cycle time, or throughput may require analysis of root causes. Often reducing WIP or batch sizes can improve process throughput and stability.
Implementation Considerations	CFDs are often available as built-in reports from common agile workflow management tools, which provide additional filtering and reporting options according to the process workflow states in use. CFDs can also be constructed based on measures collected, analyzed and reported using spreadsheet tools. The sample intervals for collection or analysis of CFD data items (e.g., daily, weekly, monthly) may vary based on the program's defined processes or business environment.

Additional Specification Information	
Information Category	1. Schedule and Progress 2. Process Performance
Measurable Concept	1. Work Unit Progress 2. Process Effectiveness

Relevant Entities	Tasks, stories, features, capabilities.
Attributes	Arrivals / departures for workflow state transitions
Data Collection Procedure	Workflow state information (quantities by state over time) and Cumulative Flow Diagrams are typically obtainable directly from software task planning and management tools.
Data Analysis Procedure	Cumulative flow is analyzed by the team regularly (e.g., daily or weekly) to monitor work in progress and completion. Measures are analyzed periodically (e.g., monthly, quarterly, end of each iteration or release) to determine if process performance levels are in line with objectives and sufficient to meet work remaining in the project plan. Corrective actions and process improvements are identified to bring performance within expectations as needed.

8.5 CYCLE TIME/ LEAD TIME (TEAM OR PRODUCT MEASURE)

Measure Introduction	
Description	Cycle Time and Lead Time can be used to evaluate efficiency in developing work products and as predictors for estimating future work. Cycle Time and Lead Time are similar and related measures that determine the duration for completing new work or products. The differences are in when start times are measured, as depicted in the diagram to the right, and described further below. Refer also to Figure 2, Measurement Context Diagram.
Relevant Terminology	Cycle Time: The elapsed time from when work is started until the time work has been completed. (e.g., Capability, Feature, Story, Defect). Cycle Time is expressed in terms and context of the team capability. It is typically targeted at measuring repeatability and predictability of team performance for well-scoped work so that results are comparable across multiple similar efforts (stories, features, capabilities). It often excludes the up-front effort needed to define and prepare the work to be implemented, such as backlog, prioritization, planning, requirements analysis, design. Lead Time: Similar to cycle time but is expressed in terms and context of the user or stakeholder perspective. It is measured from the time work is identified and a request is provided to the time until the time it is satisfied. Lead Time includes these up-front necessary activities such as backlog, prioritization, planning, requirements analysis, and design. Lead Time, Cycle Time (and Release Frequency) are closely related measures calculated similarly. The primary difference is in the information need and objective (repeatable team performance vs. user/stakeholder need) which can drive when the start/end times are measured for various activities. Lead Time may also be used to measure a higher-level aggregate business need, as opposed to Cycle Time which may measure the base elements needed to ultimately satisfy that business need.



Information Need and Measure Description	
Information Need (Cycle Time)	How long does it take to release a viable product *(team, product, enterprise)*
Information Need (Lead Time)	How long does it take to deploy an identified feature/capability, once a request is submitted? *(product)*
Base Measure 1	Start time for a process activity *(date and time)*
Base Measure 2	End time for a process activity *(date and time)*
Derived Measure 1	Elapsed Time = (End Time – Start Time) + 1 *(Units may vary based on team context, capability, cadence; e.g., hours, days, weeks, months. May also vary based on calendar time vs. work days. Results with fractional values are rounded up to the next unit.)* Examples: 1: Cycle Time = 08/21/2019 – 08/20/2019 = 2 days 2: Cycle Time = Fri 09/13/19 – Mon 09/02/19 = 12 calendar days = 10 workdays = 2 work weeks 3. Cycle Time = 09/01/19 12:52 – 09/01/19 08:05 = 5 hours 4. Lead Time = 08/31/19 – 6/15/19 = 78 calendar days

Indicator Specification

Indicator Description and Sample

Figure 19: JIRA Control Chart focusing on an area of interest

Figure 19 is from a [JIRA Control Chart](). Report filters are available to focus on areas of interest (e.g., time ranges, event types, product or team characteristics). Other tools and charts are also common in industry, but typically include information such as:

- Plots of cycle time or lead time measures for software deliveries over a defined time range.
- Statistical analysis of process performance measures (e.g., mean, median, rolling average, standard deviation)

In the example shown, the team has delivered 278 releases over the past 6 months with a median cycle time of nearly 4 days. There has been a fairly broad range of performance during this period overall (indicated by a consistently wide standard deviation in the blue shaded range) with some outliers that resulted in an overall mean cycle time of 5-6 days. The team usually meets the organizational objective for implementing releases of new capability within one week of starting work, but still has challenges with consistent performance and predictability, as indicated by the rolling average and standard deviation for the 55 most recent releases. Analysis of the outliers showed poorly defined requirements which caused extended periods of work in progress for those stories. The team (supplier and acquirer) implemented actions to ensure that requirements analysis and design were properly vetted before work started.

The frequency of product releases appears relatively consistent judging by the spread of plotted clusters of product deliveries over this time period. However, the team did not show much improvement in performance over time. The team is aiming to reduce cycle times and variation by improving its processes and training for story point estimation, and consistently planning and implementing smaller batch sizes for future releases.

Analysis Model	Analysis of Cycle Time or Lead Time measures can indicate process performance trends or potential indicators of issues for root cause analysis and performance improvement. Example analyses may include: • Process efficiency and stability (increase/decreasing delivery times or throughput) • Predictability for future performance (narrowing or widening standard deviation in delivery outcomes) The analyst may consider questions such as: • Is the cycle time consistent across iterations? • Is cycle time increasing or decreasing? • Do the cycle time and lead time performance (Voice of the Process) meet the business need (Voice of Customer)? • How predictable is the release cycle? Can we reliably estimate future performance? • What are the root causes for process outliers? • Are process improvements effective? • Are any corrective actions needed to bring performance in line with expectations? Shorter cycle times can indicate effective delivery flow and quicker time to market. Longer cycle times are often correlated to the number of items for Work in Progress (WIP). Consider moderating attributes of the assigned work and resources in order to achieve predictable performance. Tuning small batch sizes for WIP is a common approach used to achieve a consistent delivery cadence. Teams should implement improvements to bring capability and performance in alignment with the business need. Lead times and release frequency can be optimized by managing backlog depth to reduce latency of critical capabilities or applying additional resources to work concurrently.
Decision Criteria	Investigate outliers for cause of variations. Review each outlier that is more than 10% from the average cycle time.

Additional Information	
Additional Analysis Guidance	Under consistent conditions, cycle time and lead time can be used as measures of team capability and throughput that can be used in lieu of traditional size-based productivity measures (such as lines of code / hour). Reductions in cycle time and lead time measures can indicate faster delivery to the customer, which yields additional potential business benefits such as: • Increased productivity • Identification of innovation opportunities • Higher customer satisfaction and employee satisfaction
Implementation Considerations	Cycle time and lead time measures can be automatically collected and analyzed by many common tool suites. Refer to Data Collection Procedure for details.

Additional Specification Information	
Information Category	Process Performance – Process Effectiveness
Measurable Concept	Process Efficiency - Speed
Relevant Entities	Features, Stories; Defects
Attributes	Time stamps for process state transitions (start, end)

Data Collection Procedure	Cycle Time and/or Lead Time indicators are often generated directly from software project management tools, such as: • VersionOne = Select reports -> Work item Cycle (ensure start cycle is In Progress) • Jira (Control Charts for selected measures) = Reports -> Control Chart -> Refine Report -> Choose Cycle time status Data for these indicators can also be collected manually: • Excel = Subtract Start Date from End Date and average across all Features or Stories
Data Analysis Procedure	Data is analyzed at the end of each iteration by the team during the iteration review and considered during the planning session for the follow-on iteration. Performance trends of team or organizational capability may be analyzed at periodic intervals (e.g., quarterly) by the program to assess systemic issues and identify improvement actions to align performance with business objectives.

8.6 Defect Detection (Team, Product, or Enterprise Measure)

Measure Introduction	
Description	Programs strive to deliver products of acceptable quality for use by internal or external customers, and to manage the extent of defects and rework that could inhibit the effective use of these products in operations. Acceptable quality can often be a tradeoff against other attributes, such as speed, cost, and time to market. Quality objectives may vary by application domain and the business goals of the enterprise, but the objective is generally to minimize the quantity of defects detected after release (escaped) or conversely, to maximize the defects detected during development prior to product release (contained). This may be accomplished through defect detection processes such as effective peer reviews, automated testing throughout development, and other verification and testing approaches.
Relevant Terminology	Defect terminology is defined in Section 3: Ontology and Definitions and in Figure 3: Defect Terminology.

Information Need and Measure Description	
Information Need	How many defects were contained (discovered) prior to internal release? How many defects were released (escaped) to an internal customer (e.g., Integration and Test, Formal Test) or released (escaped) to an external customer (e.g., end users)? For each major release, how many defects were detected in internal development (contained, saves)? What is the ratio of escaped defects (internal and external) to all defects? Does committed work (stories, features, capabilities) work as expected?
Base Measure 1	Contained Defects (integer scale)
Base Measure 2	Internally Escaped Defects (integer scale)
Base Measure 3	Externally Escaped Defects (integer scale)
Derived Measure 1	Total Defects = Contained Defects + Internally Escaped Defects + Externally Escaped Defects
Derived Measure 2	Internal Defect Escape Ratio = Internally Escaped Defects / Total Defects
Derived Measure 3	External Defect Escape Ratio = Externally Escaped Defects / Total Defects
Derived Measure 2	Total Defect Escape Ratio = (Internally Escaped Defects + Externally Escaped Defects) / Total Defects

Indicator Description and Sample	The concept of categorizing defects as either contained or escaped is key to this measure and others (e.g., Defect Containment). As shown in Figure 3 on page 7, and repeated below in Figure 20, all defects detected before the release (during development, noted in the blue box) are Contained Defects. All defects detected after release in internal or external operations (noted in the beige and orange boxes) are Escaped Defects.

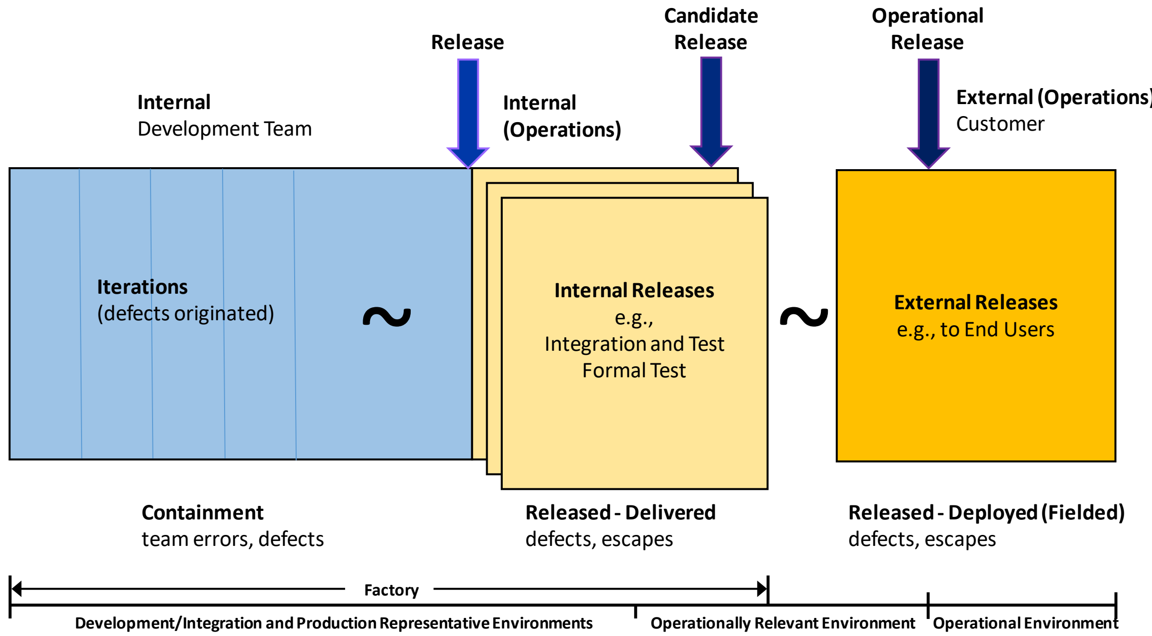


Figure 20: Defect Terminology

The Defect Escapes table (Table 6) is used to show Contained and Escaped Defects for each release along with the Defect Escape ratio. This measures the quality of the completed product based on the number of defects detected before release (Contained Defects) and after release (Escaped Defects). It also monitors the effectiveness of defect detection processes and verification activities performed during development to detected defects prior to release. Note: while only major releases (e.g., 1.0, 2.0, 3.0) are external releases, it is possible to detect external escapes attributed to minor releases after investigation and assignment of iteration introduced.

Table 6 Defect Detection by Release

	Defects				Escape Ratio		
		Escaped					
Release	Contained	Internally Escaped	Externally Escaped	Total Defects	Internal Escape Ratio	Exernal Escape Ratio	Total Escape Ratio
Release 1.0	48	9	3	60	15%	5%	20%
Release 1.1	55	5	1	61	8%	2%	10%
Release 1.2	31	4	0	35	11%	0%	11%
Release 2.0	64	5	2	71	7%	3%	10%
Release 2.1	55	8	0	63	13%	0%	13%
Release 2.2	48	4	0	52	8%	0%	8%
Release 2.3	31	3	0	34	9%	0%	9%
Release 3.0	20	1	0	21	5%	0%	5%
Cumulative	352	39	6	397	10%	2%	11%

	In the example above, Release 1.0 had a ratio of 20% of total escaped defects, with 5% of recorded defects detected after release to the customer. This gradually improved over time to a ratio of 5% on Release 3.0. This was due to a more stable set of requirements, improved test coverage and a more mature product. The Defect Escape Ratio was higher for Release 1.0 because the team decided to implement the more difficult functionality in the first release. Sixty-four defects were discovered in Release 2.0 due to a significant product update. Only 2% of defects were detected externally by the customer. An alternative way to apply the concept of contained and escaped is to implement the Defect Containment measure. Instead of identifying defects as contained or escaped in relation to the release to an internal or external customer, they would be identified in relationship to iterations. Defects detected in the iteration in which they were inserted (originated) are contained and those detected in later iterations are escaped. Defect counts could be shown in a table as in Table 7 below, identifying which iteration the defects were originated and which iteration the defects were discovered. If this information is unknown, those defects could be tracked separately as Unknown. If legacy defects are detected that were inherited (not originated) by the development team, those could be tracked as Legacy. In a manner similar to the Defect Escape Ratio, various ratios could be determined (e.g., ratio of defects discovered one iteration after they were inserted). See the PSM core framework for more information on Defect Containment.

Table 7: Defect Resolution Lag Time

Defect Resolution Lag Time
As of 19 Dec 19

Defects		(Iteration) **1**	**2**	**3**	**4**	**5**	**6**		
Defect Discovered (Iteration) **Unknown**	0								
Legacy	0								
1	82	29	2	19	17	4	11		
2	123		27	71	6	7	12		
3	282			122	60	29	71	Blank	0%
4	112				16	2	94	>1 Iteration	41%
5	7					5	2	1 Iteration	21%
6	54						54	Same Iteration	38%
	Total	29	29	212	99	47	244		
	660								

	For this data, 38% of the defects were resolved in the same iteration they were detected. This is less than the organizational goal of 80%. Another 21% were detected in the next release. 41% of defects took at least two iterations to detect, which indicates that the assessment of the iterations needs to be improved, possibly with increased automated test. Some of these escaped defects were not found until after internal release, once an end-to-end test was performed.
Analysis Model	The Defect Escape Ratio is analyzed to determine the quality of a given release and whether the team is improving over time. The Defect Escape Ratio should be getting smaller over time. The defect containment indicator can be used to evaluate the adequacy and completeness of the testing process and the sufficiency of the automated test. The enterprise may analyze defect escape ratio across multiple programs, especially external escapes, to evaluate those programs that are successfully handling defects.

Decision Criteria	Is the Defect Escape ratio acceptable? Is the ratio getting better over time? Are at least 80% of defects detected in the iteration where they were originated? Are at least 98% of defects detected before external release?

Additional Information	
Additional Analysis Guidance	These tables could be separated by priority (e.g., priorities 1-3 and priority 1) or other attributes. This measure may be used in conjunction with other quality measures including the Defect Density, Defect Resolution, and Rework measures. By looking at both internal and external escapes, the team can determine where improvement actions are needed. A project may intentionally decide to defer defects and add them to the backlog for consideration for resolution in a later iteration or release. These deferred defects may be tagged and tracked separately.
Implementation Considerations	Defects in the problem reporting tool must be discernable whether they were detected before (contained) or after (escaped) the release to an internal or external customer. A parameter or a review of the dates could be used to determine if defects are contained or escaped.

Additional Specification Information	
Information Category	Product Quality
Measurable Concept	Functional Correctness
Relevant Entities	Defects
Attributes	Project activity or iteration where defects are detected (e.g., development, internal release, external release).
Data Collection Procedure	Defect data is recorded in the problem reporting tool as defects are detected. Each defect must be categorized as contained or escaped by assigning a parameter in the tool or by the iteration or date detected.
Data Analysis Procedure	Defect counts and ratios are analyzed at the end of each major release to determine status and progress over time.

8.7 Defect Resolution (Team or Product Measure)

Measure Introduction	
Description	Defect Resolution refers to the process of correcting defects that are detected in the system. It is used in conjunction with the Defect Detection measures to ensure that critical defects are resolved in an efficient manner and do not result in inherent quality problems.
Relevant Terminology	The terms defects (team errors), iterations, containment, escapes, and releases is defined in Section 3: Ontology and Definitions and in Figure 3: Defect Terminology. These terms are also used in the measurement specification for Defect Detection.

Information Need and Measure Description	
Information Need	• When are detected defects resolved? Are high priority defects resolved prior to release? • How many iterations does it take to resolve defects? (aging) • Which defect types have the greatest impact? • Are certain defects taking longer to resolve than others? • How effective was the defect resolution process?
Base Measure 1	Defects detected, per iteration (integer scale)
Base Measure 2	Defects resolved, per iteration (integer scale)
Base Measure 3	Iterations to Resolve (# of iterations between detection and resolution) (integer scale)
Derived Measure 0…n	Resolved 0...n Iteration = the number of defects that are resolved 0..n iterations after being detected Note: Defects resolved in iteration 0, are contained defects.

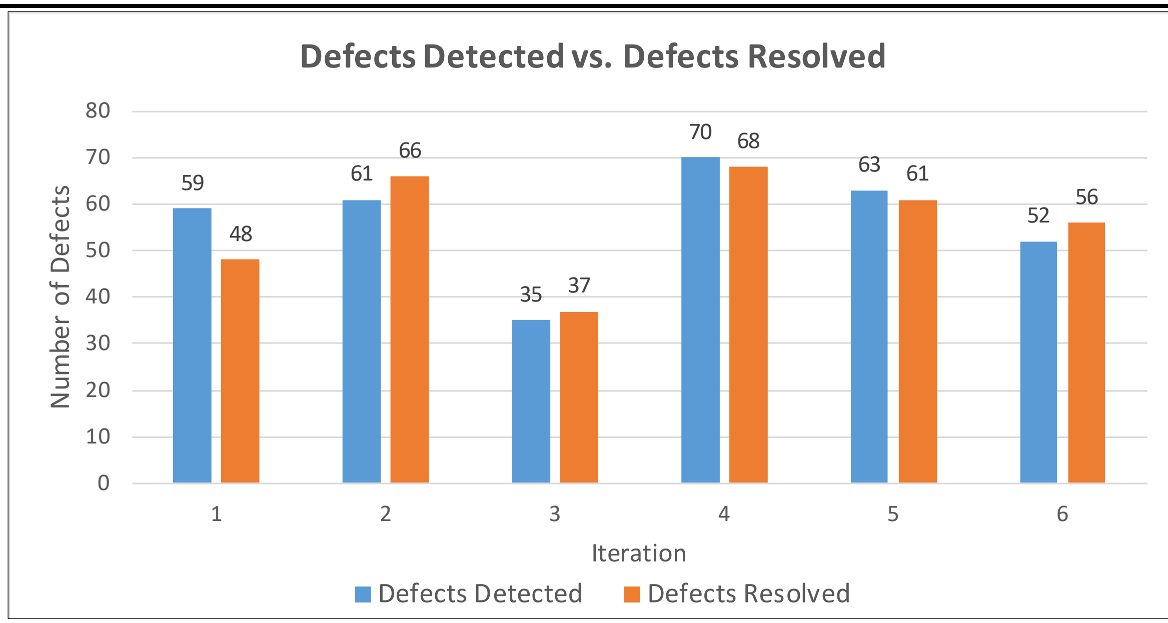
<table>
<tr><th colspan="2">Indicator Specification</th></tr>
<tr><td>Indicator Description and Sample</td><td>


Figure 21: Defects Detected versus Resolved

Figure 21 shows that for Iteration 1, not all the defects discovered in Iteration 1 were resolved in Iteration 1. These defects were then deferred, put on the product backlog, prioritized, and planned to be resolved in upcoming iterations. For Iterations 2 and 3, more defects were resolved than detected, meaning that defects discovered from previous iterations were resolved, thus reducing the product backlog.
</td></tr>
</table>

Figure 22 shows the cumulative number of defects detected and resolved. In Figure 21 and Figure 22, Iteration 6 was planned to address defects vs. adding new features and capabilities.

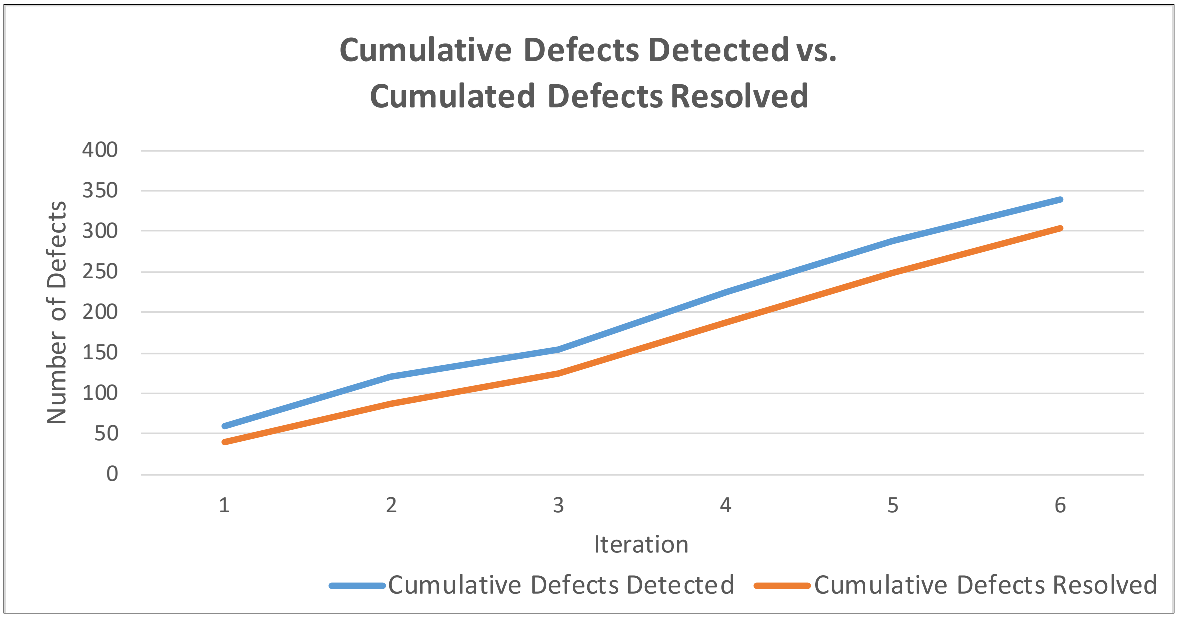


Figure 22: Cumulative Defects Detected vs. Cumulative Defects Resolved

An issue that is often evaluated is how long it takes to resolve discovered defects. In a simplistic case, one can look at how many iterations it takes to resolve the defect. This is shown as a simple bar chart in Figure 23 as Defect Resolution Lag Time. In this example, the defects that took 4 and 5 iterations to fix were lower priority defects dealing with minor changes to screen displays and software documentation.

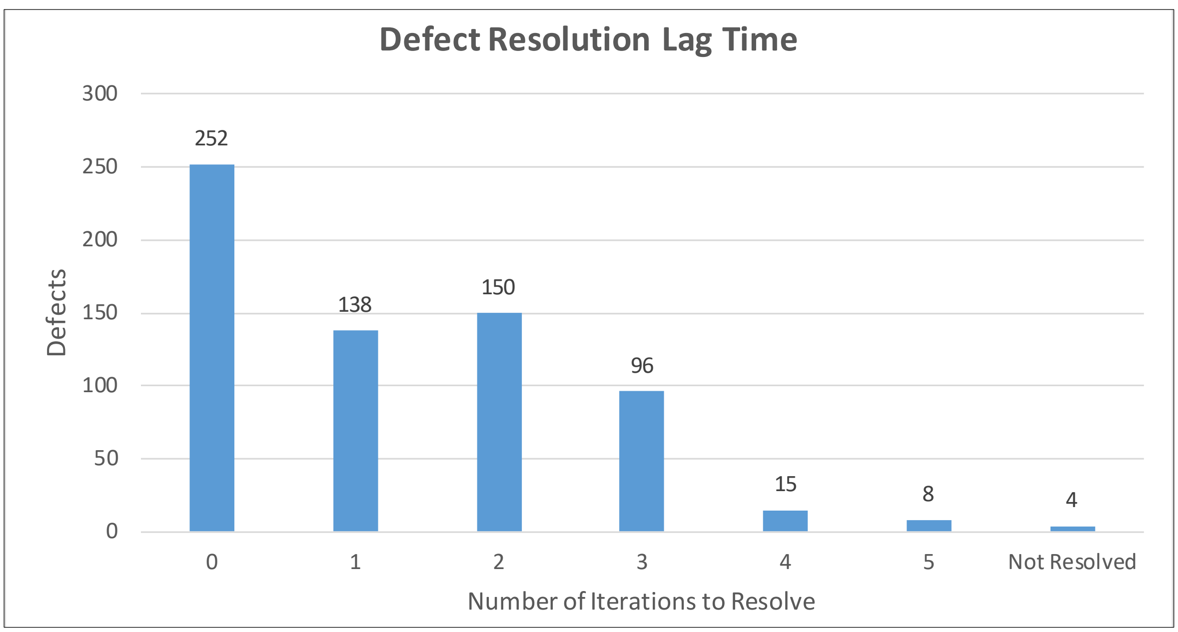


Figure 23: Defect Resolution Lag Time

Preferably, a defect would be resolved in the same iteration as it was discovered (the green series of diagonal cells in Table 8 below). All cells to the right of this diagonal represent escaped defects across iterations. Filtering can be applied for the most critical or highest priority defects. Defects that are not resolved after multiple iterations may represent a risk to the inherent quality of the product, may represent an issue with the defect resolution process, or may indicate lower priority defects that have not been prioritized for implementation. Analysis of the Defect Resolution Lag Time measure should focus on the high priority defects and ensure they are being resolved in a timely matter.

Table 8: Defect Resolution Lag Time

Defect Resolution Lag Time

As of 19 Dec 19

Defect Detected (Iteration)		Defect Resolved (Iteration) 1	2	3	4	5	6	Not Resolved
1	59	48	11					
2	61		55	6				
3	35			31	4			
4	70				64	6		
5	63					55	8	
6	52						48	4
Total	**340**	48	66	37	68	61	56	4

Blank	0%
>1 Iteration	0%
1 Iteration	10%
Same Iteration	89%

Analysis Model

Figure 21, Defects Detected vs. Defects Resolved, shows the difference/delta between defects discovered and defects resolved, by iteration.

The Cumulative Defects Detected vs. Resolved indicator can be used in conjunction with the Feature or Capability Backlog measure. When checked cumulatively, if the number of defects discovered is greater than the number of defects resolved, the backlog is growing. If the number of defects discovered is less than the number of defects resolved, the backlog is getting smaller.

Decision Criteria

In Figure 21, for each defect that does not get resolved in the same iteration as it is discovered, the defect and its priority shall be considered during the planning session for the follow-on iteration.

In Figure 22, when the difference/gap between cumulative defects discovered and cumulative defects resolved exceeds 20% of the cumulative defects discovered, the team shall consider having an iteration specifically designed to resolve the outstanding defects.

In Figure 23 and Table 8, defects with Priority 1 and 2should have a defect resolution lag time not greater than 1 iteration. If not, the defect shall be considered for resolution in the next iteration, with customer approval of this action. Priority 3 through 5 defects may be deferred until later iterations, based on customer priorities.

In Figure 23 and Table 8, most Priority 1 and 2 defects should be resolved prior to release (e.g., a condition of release). Some may be deferred to a later release, with customer agreement. Priority 3 through 5 defects not resolved may be released with customer approval and have a customer approved work around.

Additional Information	
Additional Analysis Guidance	Considering the nature of agile development, a defect lower in severity and priority in the product backlog may not be resolved immediately but, be deferred to be resolved in a later iteration. To account for this planned delay, the Defect Resolution Lag Time could be derived from the Iteration the defect was resolved to the Iteration the defect was planned to be resolved (instead of Iteration the defect was detected). The derived measure for Defect Resolution Lag Time listed above is measured for defects that were resolved. The lag time for open, unresolved defects would be calculated by the Current Iteration less the Iteration the defect was detected. More advanced analysis may evaluate (new) defect insertions during defect resolution, or defects resolutions that failed. Recidivism rates may be an important customer concern.

Implementation Considerations	Counting methods need to be defined to determine:
	• What constitutes/does not constitute a defect
	    • E.g., peer review findings may be considered errors and not considered internal defects
	    • E.g., an internal error that is sent back to the originating team and results in rework, may be considered a defect
	• When defects will/will not be counted (e.g., upon hand-off to another team/3rd party)
	• Internal defects vs. external defects (e.g., defects discovered by the developer, by the customer in an operationally representative environment, or by the customer in operations)
	Determining a value for the Iteration the defect was detected and the Iteration the defect was resolved may be tool dependent.
	As an alternative view, these measures and indicators may be constructed using only Priority 1-3 defects that affect functional performance.
	Some iterations may consist of only defects resolutions. Keep this contextual information in mind when it comes to analyzing the data.

Additional Specification Information	
Information Category	Product Quality
Measurable Concept	Functional Correctness
Relevant Entities	Defects
Attributes	Iteration Defect was Detected Iteration Defect was Resolved Defect Priority
Data Collection Procedure	Data is collected at the end of each iteration by the team lead from the team tracking tool.
Data Analysis Procedure	Iteration the defect was detected and Iteration the defect was resolved are discussed during the defect tracking and defect resolution meetings. Data is analyzed at the end of each iteration by the team during the iteration retrospective meeting and considered during the planning session for the follow-on iteration.

8.8 Mean Time to Restore (MTTR)/ Mean Time to Detect (MTTD) (Product or Enterprise Measure)

Measure Introduction	
Description	In an operational environment, continuity of deployed services is fundamental to the delivery of user value. MTTR is essential for systems in which operational availability is critical. This includes both critical embedded systems as well as those systems focused on the delivery of software services. Operations can be impacted by planned or unplanned outages. Operational service incidents are typically recorded in a trouble ticket which is used to track the incident to closure and restoration of service. Each trouble ticket has an associated restoration time. Sometimes there may be an alternative or workaround that enables the service to continue in the field, such as redundant paths or resources, even if in a degraded mode. Some repairs must be returned to the factory for correction and redeployment. The enterprise may collect the average time to detect a service-impacting issue (Mean Time to Detect) and the average restoration time (Mean Time to Restore). This provides measures of operational effectiveness for maintaining service continuity, across all tickets, or classes of tickets. A summary of these concepts is depicted visually in Figure 2, Measurement Context Diagram. MTTR, MTTD and other operational measures of service continuity can be applied in each of many potential stakeholder environments including the development/integration environment(s), production representative environment, or operationally relevant environment, or the operational environment. The enterprise generally focuses on actual measures from the operational environment. The product team may also focus on ensuring MTTD/MTTR objectives will be met as the system is developed and sustained.
Relevant Terminology	Mean Time to Detect (MTTD): Time required to identify an interruption to service delivery. MTTD measures how long it takes the operations team to detect that an incident has occurred which affects delivery of operational services. Mean Time to Restore (MTTR): Time required to restore service after an outage occurs. MTTR measures how long it takes the operations team to restore the system to an operational state, either through a rollback, restart, fix in operations, return to the factory for repair, or another action. Sometimes synonymous with Mean Time to Recover, but with a focus on restoration of operations.

Information Need and Measure Description	
Information Need	What is the reliability and availability of operational capabilities? How long does it generally take to restore service when a service incident occurs? How quickly can we recover from failures that impact the system in operations (e.g., impacts service reliability or availability), or the software in development or test? *(time to restore the build or the service to a previous, known good state.)*
Base Measure 1	Failure Occurrence Time *(timestamp)*
Base Measure 2	Failure Detection Time *(timestamp)*
Base Measure 3	Service Restoration Time *(timestamp)*
Derived Measure 1	Time to Detect = (Failure Detection Time) – (Failure Occurrence Time) *(units for elapsed time may vary; seconds, minutes, hours, days)*
Derived Measure 2	MTTD = ∑ (Time to Detect) / N *(rolling average Time to Detect, based on N previous failures)*
Derived Measure 3	Time to Restore = (Service Restoration Time) – (Failure Occurrence Time) *(units for elapsed time may vary; seconds, minutes, hours, days)*
Derived Measure 4	MTTR = ∑(Time to Restore) / N *(rolling average Time to Restore, based on N previous failures)*

Indicator Specification

Indicator Description and Sample

When practicing CID, a key concern is speed: to deliver software rapidly and frequently. However, quality should be maintained. In particular, when practicing Continuous Deployment into operations it is important to be able to quickly recover when a new release/deployment introduces a failure in this live environment.

MTTD and MTTR indicators can be represented in multiple ways (e.g., graphical, tabular). In Figure 24, three measures are plotted for each operational outage: Time to Detect, Time to Repair, and Time to Restore (sum of detection + repair). A comparison of data across outages indicates general trends, severity, and operational impacts. A summary of statistical measures (mean, median, standard deviation) for each of detection time, repair time, and total restoration time is provided in the table below the chart. A rolling average of Mean Time to Restore (MTTR) is also plotted for the 10 most recent outages.

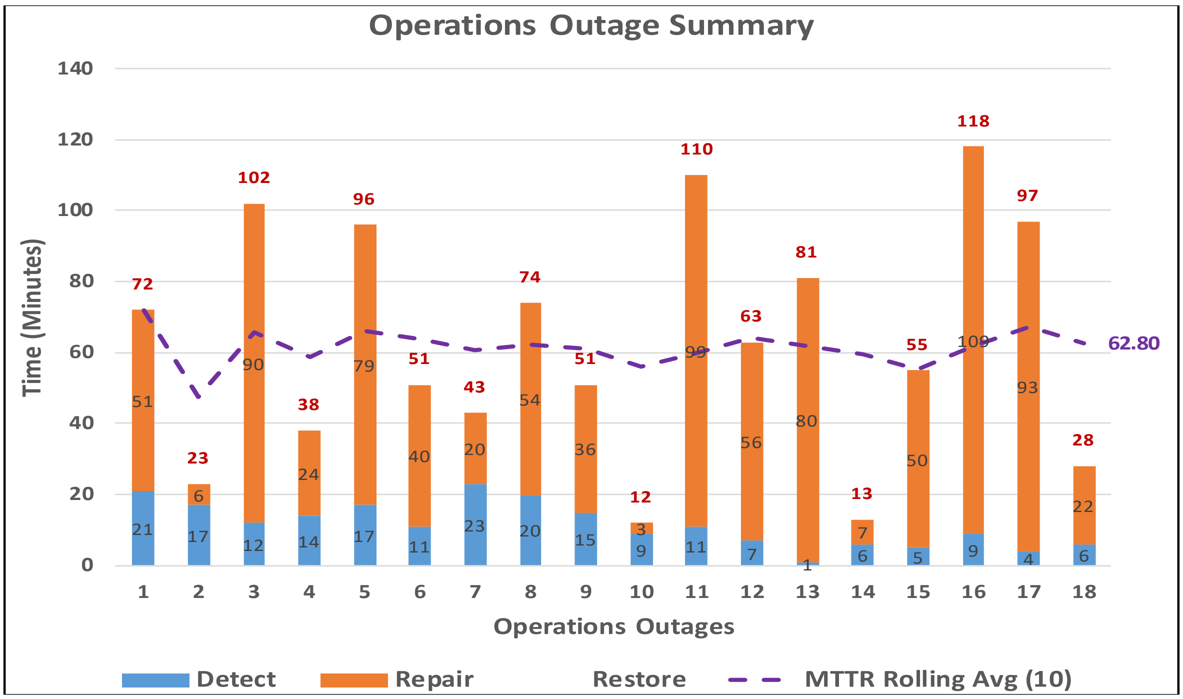


Figure 24: Operations Outage Summary

In this example, although there are significant variations in individual outage samples (some anomalies are more complicated to fix than others), in aggregate the MTTR rolling average is holding fairly steady (around 1 hour to restore service). Similarly, the mean and median times for Time to Detect, Time to Repair, and Time to Restore are consistent despite a large standard deviation. (Table 9)

In the sample indicator, the four short MTTRs are cases where the system was rolled back to a previous version. The longest cases are indicative of complex issues that required additional repair time. The lengthy MTTR in Outage 16 involved an update to a critical component. The fix/corrective action was not implemented correctly, which resulted in Outage 17. An alternative solution was implemented, and the software was shown to work in the next iteration.

Table 9: MTTR Statistics

	Detect	Repair	Restore
Mean	11.56	51.06	62.61
Median	11	51	59
Std Dev	6.31	34.09	33.28

In this example, feedback from the user community indicates outages of greater than 30 minutes can have a significant impact on Operations, due to reports that are due twice hourly. Missing two consecutive reports impacts decision making. This example program is considering ways to shorten restore times, such as implementing automated roll-back capabilities where any new deployment/release that introduces a failure can be rolled back and the previous release rapidly restored. Program personnel are also conducting a Pareto analysis of outage times by defect type to determine which outage types are most costly, so that resources can be prioritized on targeted improvement actions.

<table>
<tr>
<td>Analysis Model</td>
<td>
Data is gathered from service incident tickets and classified or filtered into affinity groupings of interest (e.g., priority, type, component, severity, impact, duration, detection method). Trends and root causes are evaluated. Improvement plans may be defined and implemented with corrective/preventive actions to mitigate the frequency or impact of future occurrences, as appropriate, relative to business objectives. The effectiveness of improvement actions should also be measured.
Both MTTD and MTTR need to be evaluated as to whether they meet the business/mission needs in terms of reliability and availability. Projections and actuals are evaluated against objectives, and trends are analyzed to project whether required objectives will be met.
A good pipeline should include significant automated testing such that any failure-inducing defects or issues are detected before deploying into the operational environment.
MTTD and MTTR are measures of failure trends for a set of issues across a range of time, and they characterize the capability to maintain and rapidly restore operations and operational service. Analysis and improvement actions can vary based on the situation and trends of performance measures and whether these are reliable predictors of future performance so improvement actions can be effective. Examples of potential areas for investigation are summarized in the table below:
<table>
<tr><td>Trend</td><td>MTTD</td><td>MTTR</td></tr>
<tr><td>Increasing</td><td>• Ineffective monitoring, detection processes, tools, training
• Incomplete knowledge of failure modes</td><td>• Increasing complexity of system, software, or architecture
• Lack of rollback capability or strategy
• Lack of effective redundancy
• Developer changes / inexperience</td></tr>
<tr><td>Steady</td><td>• Established MTTD met and satisfied - no further improvement needed
• Predictable capability; does it meet the business need (Voice of the Customer)?
• Lack of continuous improvement</td><td>• Established MTTR met and satisfied - no further improvement needed
• Predictable capability; does it meet the business need (Voice of the Customer)?
• Lack of continuous improvement</td></tr>
<tr><td>Declining</td><td>• Improved monitoring effectiveness
• Defect prevention initiatives</td><td>• Improvements through automation, tools
• Added capability or capacity (redundancy, etc.)</td></tr>
<tr><td>Erratic</td><td>• Inconsistent monitoring or reporting processes</td><td>• Unstable processes
• Immature system
• Ineffective process improvement</td></tr>
</table>
</td>
</tr>
<tr>
<td>Decision Criteria</td>
<td>
After deployment, when MTTR or MTTD is above mission or business objectives, a decision as to whether the system should be rolled back to a previous version may be considered. If the decision is not to roll-back, the user may create a high priority change request to resolve the issue causing the high MTTR. Increasing trends in MTTR or MTTD measures, may also lead to the creation of new defects or stories to improve performance, or the need to evaluate and improve the development/test processes. This is especially important when a safety critical or mission critical failure occurs.
When additional defects are introduced after improvements are made, special attention should be applied to the resolution process.
During development and test, for any MTTD or MTTR that is more than 10% above the objective or mean, investigate the root cause(s) and decide if additional improvements or testing is required. Trends over time should be improving (getting smaller) as additional functionality is added and as the system nears deployment. Regular occurrences above the objective may mean that the system is not mature enough for operations, and deployment may need to be delayed. For trends that are increasing above the objective or mean, additional focus or process improvements may be required.
</td>
</tr>
</table>

Additional Information	
Additional Analysis Guidance	MTTR is an essential measure for systems in which operational availability (Ao) is critical, with a focus on safety-critical and mission-critical failures. MTTR is also paramount when practicing full continuous deployment into Operations: in this case Operations is an operational environment supporting live operations/missions and thus the system must maintain high reliability and availability. However, even in testing environment, a failure means that integration or test activities are impacted (and possibly deployment which may lead to cost/schedule overruns). Additional analyses of MTTD/MTTR measures can be utilized to determine appropriate actions to improve availability and rapid recovery from operational issues. Examples include statistical analysis methods, profiles of defect distribution or characteristics, Pareto charts, root cause analysis, or other quality management tools.
Implementation Considerations	Measuring individual failures and restorations should be automated as much as possible, based on timestamps in logs or other automated data collection mechanism.

Additional Specification Information	
Information Category	Process Performance
Measurable Concept	Process Efficiency – Speed Supportability – Maintainability – Dependability – Reliability
Relevant Entities	Service incidents
Attributes	Time of outage, detection, and restoration; defect priority and reason code; affected elements
Data Collection Procedure	Date/time is collected at the start of each failure or service outage, and at the time of operations or service restoration. The delta between these is the individual outage TTR. These are collected to calculate a historical mean MTTR.
Data Analysis Procedure	Data is analyzed periodically during development and test, and trends are evaluated. During operations, data is analyzed when safety or mission critical failures occur, as well as periodically.

8.9 Release (or Deployment) Frequency (Product of Enterprise Measure)

Measure Introduction

Description

As described in Overarching Principles, products are typically planned and developed iteratively (e.g., capabilities, features, stories, tasks) into a set of internal releases, candidate releases, and deployed product releases. This is represented conceptually in Figure 25.

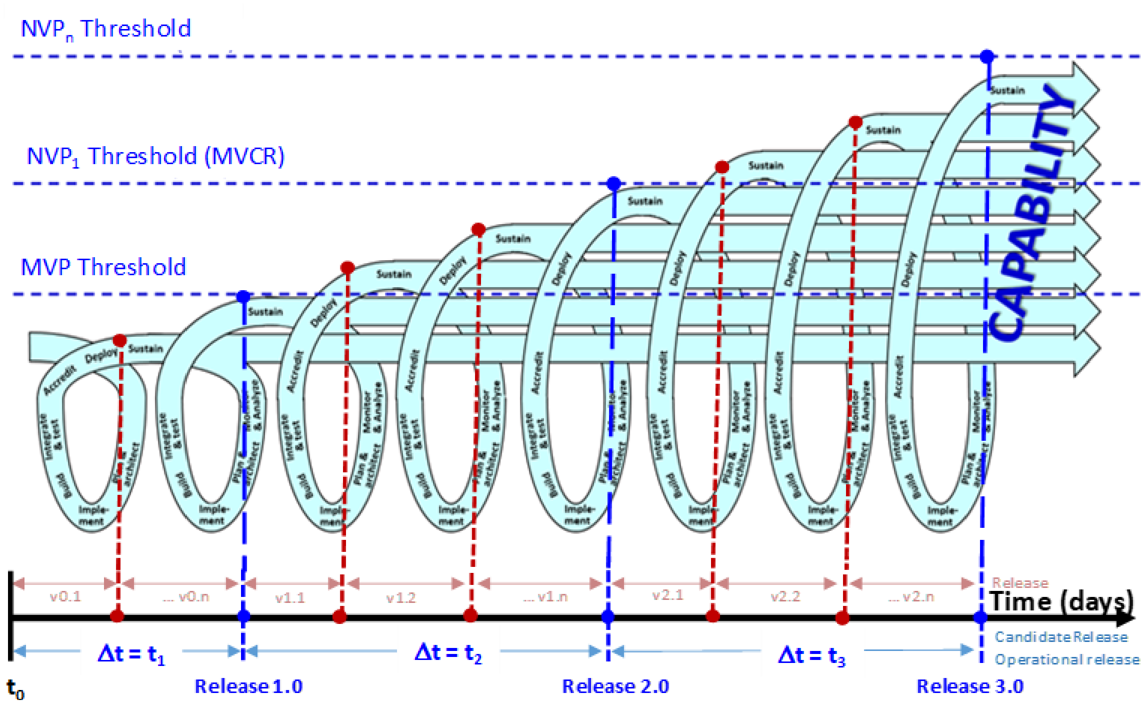


Figure 25: Iterative Development

The speed and frequency at which products are released are crucial in providing useful capability to users as rapidly as possible. The scheduling, duration, and frequency of releases can vary widely (e.g., months, weeks, days, or on demand) based on domain or business need. Products may be iteratively released on a predictable fixed cadence, or on demand as needed. The time and effort to develop candidate product releases and transition them to deployed external product releases are primary measures of efficiency in making features/capabilities available to users, as depicted in Figure 26.

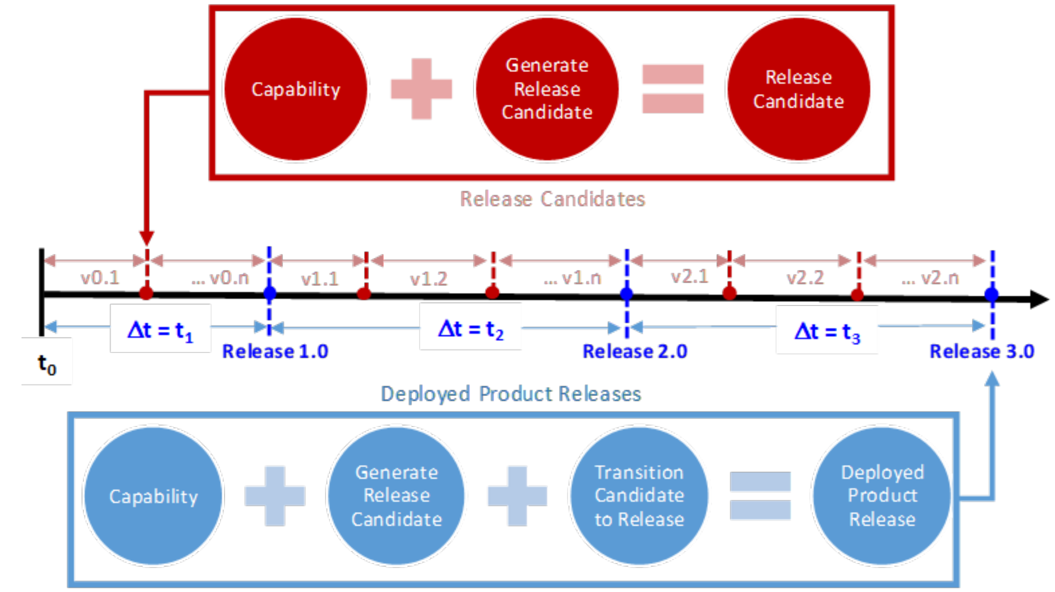


Figure 26: Product Iterative Releases (Conceptual)

Relevant Terminology	MVP	Minimum Viable Product
	MVCR	Minimum Viable Capability Release
	NVP	Next Viable Product
	Release	Internal Release; Candidate Release (External Relase); Operational Release (Deployment Release)
	Refer to glossary for definitions.	

Information Need and Measure Description	
Information Need	How long does it take to deploy an identified feature/capability? *[Product]* What is the cadence or frequency for product release or deployment? *[Product, Enterprise]* How long does it take to release a minimum viable product? *[Product, Enterprise]* How much effort/cost/time is needed to develop new products and transition them to release? *[Product, Enterprise]*
Base Measure 1	Release Start Date (datestamp) *(release, candidate release, or operational release)*
Base Measure 2	Release End Date (datestamp) *(release, candidate release, or operational release)*
Base Measure 3	Effort Hours to generate a release (integer) *(internal release candidate or external deployed release)*
Base Measure 4	# of Releases (for a specified data range)
Derived Measure 1	Release Duration = (Release End Date) – (Release Start Date) Note: release durations may be tracked for features/capabilities at various stages of maturity • Time to Minimal Viable Product (MVP) • Time to Minimal Viable Capability Release (MVCR) • Time to Next Viable Product (NVP_n)
Derived Measure 2	Release Frequency = (# of Releases) / date range (e.g., days, weeks, months, quarters, years)

Derived Measure 3	Average Release Duration = ∑ (Release Duration) / (# of Releases) *Note: weighting can be used to emphasize the most recent releases.*
Derived Measure 4	Average Release Transition Time = ∑ (Release Transition Time) / (# of Releases)

Indicator Specification

Indicator Description and Sample

In this example, (Table 11) a commercial software company deployed a new product (Tango) to the market in October 2018 (MVP release), with a business objective to release iterations twice monthly to support quarterly product capabilities releases. Ten product releases were completed between October 2018 and March 2019. The table below summarizes, for each release, the start and end dates for each release (from which duration is calculated), the type of release, and the total labor spent in hours.

Following the higher effort for the initial MVP R2018.01 release, durations of iterations have averaged 18 days. The initial MVP did not meet market needs, however, a Minimum Marketable Product (MMP) was available two months later in December 2018. After the MMP, the NVP release occurred 90 days later, in line with the business objective of quarterly releases.

A longer duration for the R2019.01 iteration (25 days) at the end of 2018 is attributed to staffing reductions due to holiday vacations. Overall averages for release time and labor across releases is shown in the Table 10, by calendar year.

Table 10: Product Release Averages

Days to Release	2018	2019
# of Releases	5	5
Days to Release (Avg)	25.8	19.4
Labor to Release (Avg)	2402	1671

Table 11: Release Frequency and Labor Hours

Release	Started	Ended	Release Type	Duration	Year	Labor Hours
R2018.01	8/20/2018	10/23/2018	Candidate Release - MVP	64	2018	6182
R2018.02	10/22/2018	11/5/2018	Release - Internal	14	2018	1313
R2018.03	11/5/2018	11/26/2018	Release - Internal	21	2018	1663
R2018.04	11/19/2018	12/5/2018	Candidate Release - MMP	16	2018	1477
R2018.05	12/3/2018	12/17/2018	Release - Internal	14	2018	1372
R2019.01	12/17/2018	1/11/2019	Release - Internal	25	2019	1553
R2019.02	1/11/2019	1/26/2019	Release - Internal	15	2019	1658
R2019.03	1/26/2019	2/11/2019	Release - Internal	16	2019	1389
R2019.04	2/11/2019	3/5/2019	Candidate Release - NVP	22	2019	2002
R2019.05	2/25/2019	3/16/2019	Release - Internal	19	2019	1756

The product team plots each release in the Figure 27 below for a visual comparison of durations (vertical bars aligned with the left axis) and labor hours invested (red line aligned with the right axis). A rolling average of the durations for the most recent 3 product releases is calculated and displayed in the dashed line.

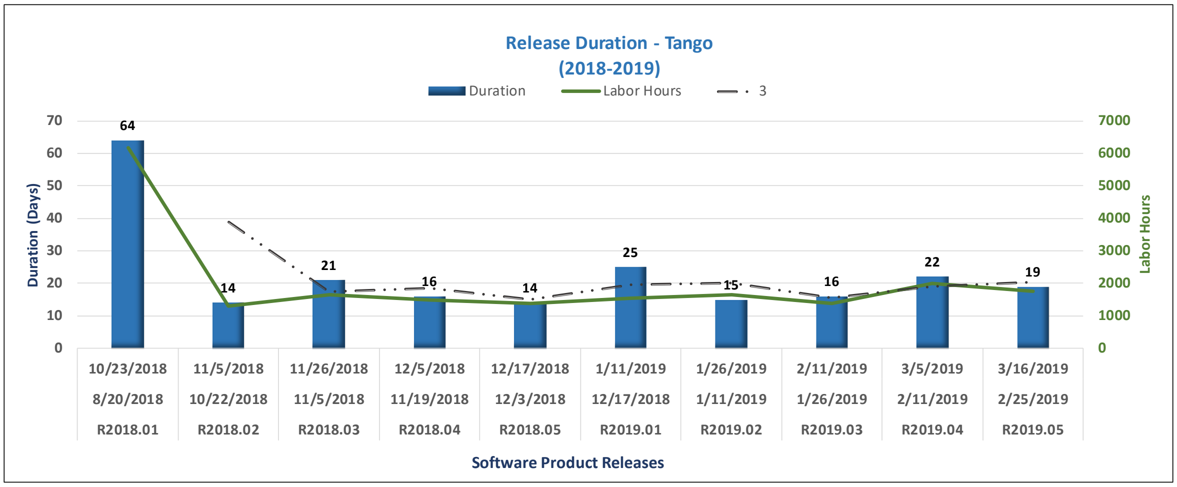


Figure 27: Release Duration for Product Tango

In Figure 28 the marketing department tracks the release frequency for all three of the company's products at the enterprise level against the business plan for twice monthly iterative releases.

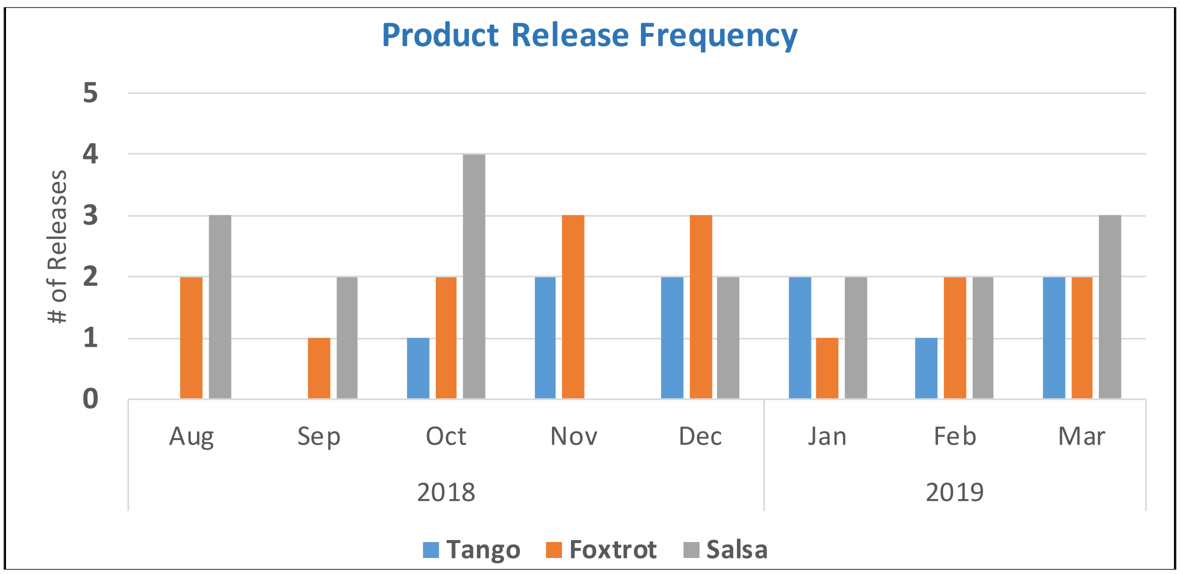


Figure 28: Product Release Frequency

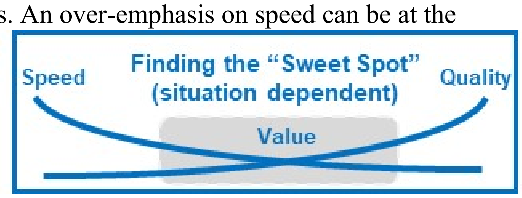
Analysis Model

Can we consistently release product baselines at a rate needed to meet demand?

Is the process performance (time and labor) for generating and deploying product releases improving? Does it take more/less/same amount of time to transition release products to candidate release products or operational release products?

Not all development organizations are in control of when their internal baselines may be deployed to live operations. For instance, deployments for military platforms must be certified and coordinated with the user community. As shown in Figure 1, additional effort may be needed to prepare and transition candidate releases to operationally representative environments. This may require a separate set of release measures to manage and optimize the rapid delivery of capabilities to end users. This preparation and transition effort may increase significantly as the baseline grows. Not only must the new capability be verified, so must prior functionality be verified through regression testing. If done manually, the additional effort for testing and release can scale at a rate incompatible with maintaining product release timelines.

	Automation can help improve build, testing, and release efficiency to maintain a consistent release transition cadence. The time to build and create product releases is directly related to the quantity and size of design and implementation features. Smaller batch sizes enable releasing products more quickly. Efficiency of the deployment and release process further accelerates the speed at which products can be released to the customer. Releases are typically built by automated build/test automation frequently on the baseline. Releases are typically built every day or upon every merge or end of a sprint. The frequency of failures for releases impacts confidence in the software baseline. Ideally over time, releases can be produced more efficiently by replacing manual steps with automation.
Decision Criteria	• If the effort to transition release products to candidate releases or operational releases is increasing steadily beyond performance goals, consider approaches such as automation or reducing batch size to increase release frequency and speed the delivery of capability to users. • Once stabilized, action may be needed if the quality of deployed products declines or if the team is unable to sustain release timelines. • Does adding features/capabilities result in increased cost to create a candidate release or operational release?

Additional Information	
Additional Analysis Guidance	Release frequency (how often?) have close dependencies with Lead Time and Cycle Time (how fast?) measures. All these measures rely on the batch size of the capability or stories being released, and the efficiency of the pipeline in generating and provisioning products. Automation of the build/test elements has a profound impact on all these measures. Consistency of staffing and team composition can also impact the team's ability to release their capabilities as needed. Generally faster release cycles on a predictable cadence are desirable to quickly deliver value to users and obtain feedback. There can be a tension between speed and quality tradeoffs. An over-emphasis on speed can be at the expense of product quality. There is often a 'sweet spot' tradeoff between speed and quality that delivers a best value solution based on project objectives. Quality needs to be monitored, in addition to speed, to ensure that these measures are appropriately balanced.  Additional statistical measures can be generated (e.g., mean, median, standard deviation, quartiles) to determine the aggregate performance, repeatability, and consistency of product release timelines.
Implementation Considerations	Applying Build/Test Automation to generate releases as early in the program as possible is recommended. Successfully generating releases as early in the release cycle should be a team priority. Integrity of the product baseline can be ensured by enforcing quality criteria for baseline merges to proceed successfully through the build/test automation pipeline.

Additional Specification Information	
Information Category	Process Performance
Measurable Concept	Process Efficiency – Speed
Relevant Entities	Releases, Effort
Attributes	Quantity, Labor Hours

Data Collection Procedure	Date/time is collected at the start and end of each iteration or release (iteration or deliverable, internal or external), typically obtained directly from automated tools. Each release must meet entry and exit criteria to be considered complete. Cycle time is calculated as the difference between release start time and release end time. Release frequency is calculated as the number of releases completed per unit time (e.g., day, week, month, year).
Data Analysis Procedure	Measures of the release process are analyzed at end of each release for performance within acceptable bounds, with corrective actions or improvements taken as necessary.

8.10 Team Velocity (Team Measure)

Measure Introduction		
Description	Velocity is a measure of team performance and the amount of work that is completed in an iteration, typically a count of completed story points or equivalent. Velocity calculations can be used to estimate the amount of work that can be accomplished by the team in future iterations and when planned deliveries will be completed.	
Relevant Terminology	Velocity	The average amount of work a team completes in an iteration or release. Used for planning and measuring team performance.
	Acceleration	Change in velocity across iterations.

Information Need and Measure Description	
Information Need	Is the team performing as expected? Does the team consistently meet the anticipated velocity? How much work can be accomplished by the team in a future iteration?
Base Measure 1	Story Points Completed (integer scale)
Base Measure 2	Iterations Completed (integer scale)
Derived Measure 1	Average Velocity = Story Points Completed / Iterations Completed
Derived Measure 2	Team Acceleration = (Current iteration Velocity – Reference Comparison iteration Velocity) / Reference Comparison iteration Velocity Note: the Reference Comparison iteration Velocity may be calculated as the Average Velocity across all teams, or by setting a goal for all teams to meet.
Derived Measure 3	Average Acceleration = Sum (Team Acceleration 1 …Team Acceleration N) / N

Indicator Specification

Indicator Description and Sample

In Figure 29, Story Points Completed is graphed for each iteration [blue bars]. Average Velocity is then graphed as of each iteration [red line] based on last 4 iterations (4-iteration rolling average).

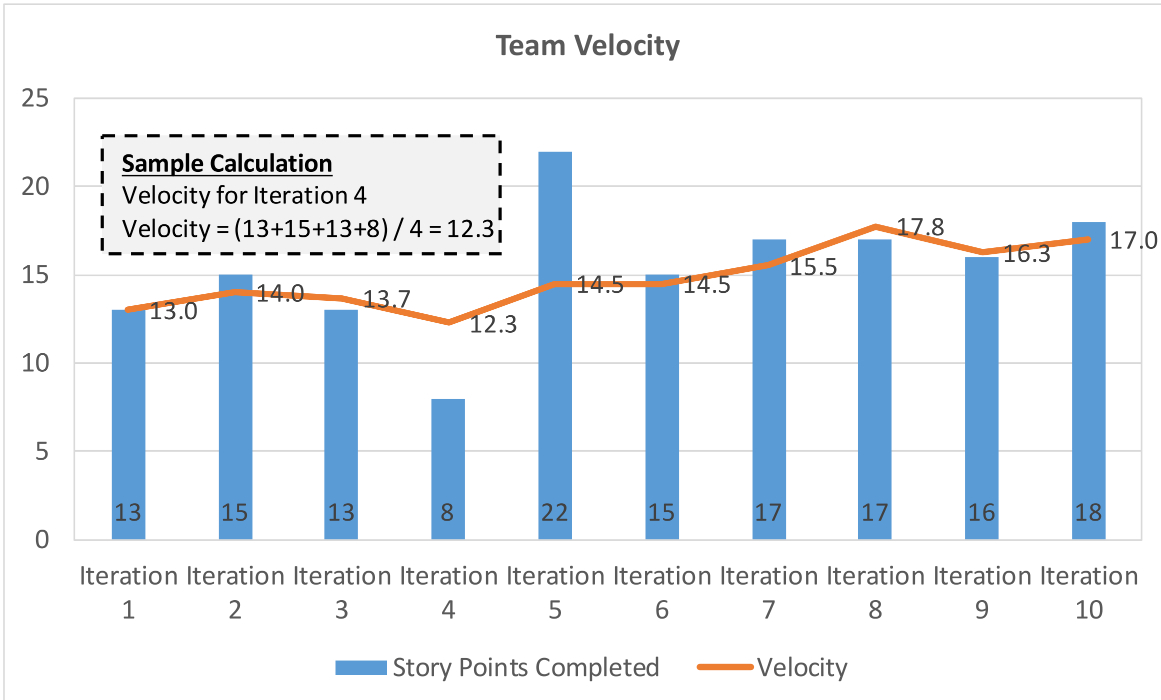


Figure 29: Team Velocity

Iteration 4 had a significant drop in velocity, followed by a large increase in iteration 5. This was due to several stories in iteration 4 that had defects. These defects were resolved in iteration 5, along with the completion of the iteration 5 assigned stories. Velocity improved and became more consistent after iteration 5, as the team became more experienced. This team established a consistent velocity after iteration 6.

Changes in velocity across iterations can be analyzed in more detail using acceleration measures. For instance, in Table 12, Teams 1, 2 and 5 show significant positive acceleration, which is typical for early iterations. Team 3 shows a dramatic drop, which should be analyzed to determine if there is a problem. Team 4 shows no variation, which may suggest a reporting anomaly.

Table 12: Sample Acceleration

	Iteration 1 Points	Iteration 2 Points	Acceleration
Team 1	10	12	20%
Team 2	8	9	13%
Team 3	14	8	-43%
Team 4	12	12	0%
Team 5	8	11	38%
Overall			5.6%

<u>Sample Calculation</u>
Team 1 Acceleration = (12-10) / 10 = 0.2
(20% positive acceleration)

Analysis Model

Do we have a consistent velocity? Why is the velocity changing over time? Based on past performance, is the average team velocity adequate to complete defined features allocated to this team? Variations may occur due to vacations, sick leave, changes to team size/composition, or implementation difficulties.

Decision Criteria

Velocity of +/- 10% should result in analysis at iteration review.

Additional Information	
Additional Analysis Guidance	Use this with the Committed Backlog and story point-to-feature ratio to ensure project will release identified features as scheduled (e.g., will velocity for remaining iterations be sufficient to complete committed features)? Will current average velocity be adequate to complete committed features by end of project? This assumes an ability to estimate average number of story points per feature (and then capability), based on performance. This measure can be used with Reference Comparison iteration Velocity for normalization. Acceleration can be tracked over time to develop predictive trends in performance. For example, performance tends to increase slowly in the first few iterations, then increase sharply, then plateau. Knowledge of long-term acceleration trends can enhance planning accuracy. Comparing individual team acceleration trends can highlight teams that have problems or that should serve as exemplars. Tracking program level acceleration trends is useful for bidding future work.
Implementation Considerations	In general, velocity is specific to a team and cannot be aggregated across teams to the project level. If velocity is normalized it can be used at the product or enterprise level. Usually, velocity should become more accurate and reliable over time as the team becomes more experienced, processes are established, data is regularly produced and reviewed, and the team gets better at estimating. Since story points may vary across teams, variations in velocity can be compared in percentage terms (positive or negative acceleration relative to prior reference iterations). This gives the program a way of determining which teams are struggling without having to normalize velocities.

Additional Specification Information	
Information Category	Process Performance
Measurable Concept	Process Efficiency - Speed
Relevant Entities	Features
Attributes	Stories, Story Points (estimated size)
Data Collection Procedure	Data is collected at the end of each iteration by the team lead from the team tracking tool. Story Points must be tested and satisfy the completion criteria, with no open defects to be counted as completed.
Data Analysis Procedure	Data is analyzed at the end of each iteration by the team during the iteration review and considered during the planning session for the follow-on iteration.

Bibliography

Defense Innovation Board (DIB), *Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage*, 2019, Software Acquisition and Practices (SWAP)

Defense Science Board (DSB), *Design and Acquisition of Software for Defense Systems*, Defense Science Board (DSB) Task Force on Design and Acquisition of Software for Defense Systems, 2018

Design and Acquisition of Software for Defense Systems, Defense Science Board (DSB) Task Force on Design and Acquisition of Software for Defense Systems. (2018, February). Retrieved from Defense Science Board: https://dsb.cto.mil/reports/2010s/DSB_SWA_Report_FINALdelivered2-21-2018.pdf

John McGarry (Author), D. C. (2001). *Practical Software Measurement: Objective Information for Decision Makers.* Addison-Wesley Professional.

Software Acquisition Pathway Interim Policy and Procedures. (2020, January 3). Retrieved from Defense Acquisition University: https://aaf.dau.edu/

Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage. (2019, May 3). Retrieved from https://media.defense.gov/2019/Apr/30/2002124828/-1/-1/0/SOFTWAREISNEVERDONE_REFACTORINGTHEACQUISITIONCODEFORCOMPETITIVEADVANTAGE_FINAL.SWAP.REPORT.PDF

Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability: An Introduction.* Daniel S. Vacanti, Inc.