Digital Engineering Measurement Framework

Version 0.95
January 25, 2022

(DRAFT – concept pending review and approval by participating organizations)

Developed and Published by Members of:

Practical Software & Systems Measurement

Systems Engineering Research Center

Aerospace Industries Association

National Defense Industrial Association

International Council on Systems Engineering

Department of Defense Research & Engineering

The Aerospace Corporation

PSM Product Number: PSM-2022-01-001

INCOSE Product Number: INCOSE-TPP-2021-128

Contents

List of Figures

LIST OF TABLES

Executive Summary

Many stakeholders and subject matter experts from across a broad cross-section of industry, government, and academia have come together here to work collaboratively on a consensus measurement framework to help enterprises transition from traditional document and artifact-based development to a digital model-based future and assess the measurable impacts and benefits they aspire to achieve.

A successful measurement program depends on establishing a clear context and operational definitions for the measures to be collected. The Digital Engineering (DE) measurement framework was developed using an approach based on Practical Software and Systems Measurement (PSM), detailing common information needs to derive an initial set of digital engineering measures. This is documented in an "Information Categories-Measurable Concepts-Measures" (ICM) Table, described in Section 7. The information needs address goals and the project (or product) and enterprise perspectives (What do we want to know with respect to the goals?) to provide insight and drive decision-making. The framework identifies an initial set of measures to address these information needs. For the highest priority measures, sample measurement specifications have been developed to describe these measures in detail along with guidance for their use.

This initial DE measurement framework proposed by our team of representative stakeholder experts is intended to help projects and enterprises establish an initial path toward a measurably effective transition and implementation of digital engineering methods. It is but the first steps along this path, it will be a long and challenging but rewarding journey, and our industry will learn, iterate, and evolve as we go. We hope enterprises across a variety of application domains will find this initial measurement guidance useful to assess the effectiveness of their respective digital engineering transformation initiatives.

1. Introduction

1.1 Background

Our industry is undergoing profound changes from traditional engineering requirements, design, development, integration, and verification methods based on documents and artifacts to a future based on digital models and cross-functional digital representations of system designs and end-to-end solutions. This document adopts a definition of digital engineering (DE) from the Defense Acquisition University (DAU):

> An integrated digital approach that uses authoritative sources of systems' data and models as a continuum across disciplines to support life cycle activities from concept through disposal.[2]

This document also adopts a generalization of the digital and model-based aspects of engineering process from the initial release of the ISO/IEC/IEEE DIS 24641:2021(E) standard: Systems and software engineering – Methods and tools for Model-based systems and software engineering:[3]

> formalized applications of modeling to support systems and software engineering

Many of the measurable benefits of DE are associated with the use of both data and digital models as a community "source of truth" for all life cycle activities. Model-based systems and software engineering (MBSSE) is an approach that uses models to drive all aspects of the product life cycle and that data is created once and reused by all downstream data consumers.[4] A practice is "model-based" to the extent that the artifacts it generates are sufficiently precise and complete that they improve life cycle efficiency and productivity.[5]

INCOSE is among many stakeholders that see digital MBSSE as foundational to the future of our industry:

> The future of Systems Engineering is Model Based, leveraging next generation modeling, simulation and visualization environments powered by the global Digital Transformation, to specify, analyze, design, and verify systems.
>
> *INCOSE Systems Engineering Vision 2035 (draft, March 2021)*

INCOSE defines Model-Based Systems Engineering (MBSE) as the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. MBSE has a particular value in DE as an approach to define the relationship between systems and lower level models as well as the life cycle process flow. MBSE supports system models that are useful for showing relationships among system functions, requirements, suppliers, acquirers, and users.

The draft MBSSE standard is integrating MBSE and the terminology and practices of Model-Driven Development (MDD) from the software community into a single MBSSE process framework. MBSSE is a Systems and Software Engineering approach centered on evolving models that serve as the "main / major source of knowledge" about the entity under consideration.

Thus, DE has three interrelated concerns: the transformation of engineering activities to fully digital infrastructure, artifacts, and processes; the use of data and models to improve the efficiency and productivity of engineering practice; and the use of MBSSE practice to fully integrate system data and models with engineering, program management, and other domains and disciplines.

Seldom have industry, government and academia been so unified in a commitment to change how we define, develop, acquire, implement, and maintain systems and products. As of this writing, our industry is still in the early stages of this digital transformation, and our processes, tools, methods, and measures must mature to fully achieve the apparent benefits of applying digital engineering methods and models across the product and system life cycle.

Organizations must be able to measure the effectiveness and business impact of their transformation efforts relative to traditional engineering methods. That is what has brought this broad team of stakeholder experts together to define a proposed consensus measurement framework to help enable and assess effective digital engineering transformations.

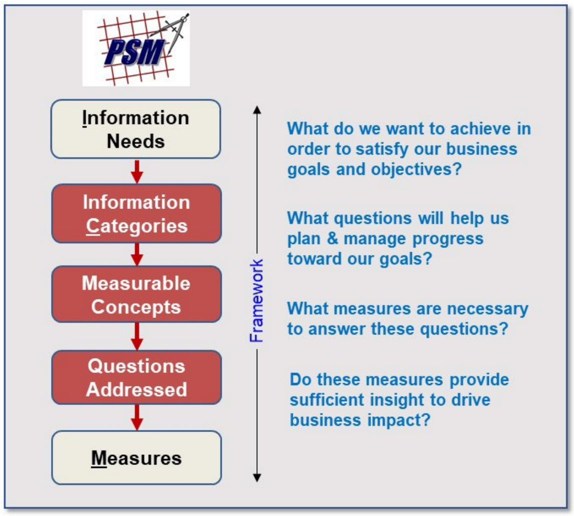


Figure 1-1: PSM Measurement Process

Measures of effectiveness start with objectives. Accordingly, the stakeholder author team has chosen to build this measurement framework aligned with information needs (What do we want to know?) to define measures for decision making, following a process based on *ISO/IEC/IEEE 15939-2017 Systems and software engineering—Measurement process*[6] and Practical Software and Systems Measurement (PSM).[7] The PSM process is summarized in Figure 1-1 and described in detail in Section 4.

Motivation for transformation towards a broad digital engineering initiative for model-based design, development and acquisition was sparked by the June 2018 release of the U.S. Department of Defense (DoD) Digital Engineering Strategy.[8] The strategy outlines five elements:

1. Formalize the development, integration and use of models to inform enterprise and program decision making.
2. Provide an enduring, authoritative source of truth.
3. Incorporate technological innovation to improve the engineering practice.
4. Establish a supporting infrastructure and environment to perform activities, collaborate, and communicate across stakeholders.
5. Transform the culture and workforce to adopt and support digital engineering across the life cycle.

Figure 1-2: DoD Digital Engineering Strategy

respective digital transformation initiatives. These included partnerships with industry associations (INCOSE, NDIA Systems Engineering Division, AIA Engineering Management Committee, and others) on several collaborative initiatives such as:

- DoD Digital Engineering Working Group (DEWG)
- Digital Engineering Information Exchange Working Group (DEIXWG)[9]
- INCOSE Model-Based Capability Matrix (MBCM)[10]
- PSM User's Group Workshop for adapting Systems Engineering Leading Indicators for Digital Engineering,[11] as input to a planned revision of the PSM/INCOSE/MIT/SEA Systems Engineering Leading Indicators Guide.[12]

These sources were a basis for identifying some of the business information needs that are now articulated in the PSM DE measurement framework.

1.3 Stakeholder Collaboration in DE Measurement Working Group

A broad set of stakeholders across government, industry, and academia shared business imperatives to implement their digital engineering transformations and realize measurable benefits in performance, effectiveness, and product quality relative to traditional engineering methods. Defining a set of measures for digital engineering was identified by the DoD Digital Engineering Working Group as one of the "pain points" for enabling digital transformation.

In 2020, the AIA Engineering Management Committee (EMC) defined a strategic project plan to define a set of measures for digital engineering. Motivated by similar concerns, other industry associations (NDIA Systems Engineering Division, INCOSE, and member companies) offered to collaborate with AIA on this project, using a PSM measurement process applied successfully on a collaborative PSM/NDIA/INCOSE project to define a measurement framework for Continuous Iterative Development (CID).[13] Other stakeholders with related objectives subsequently joined the DE measurement working group as listed in section 1.4, with the goal that the team could develop a digital engineering measurement framework with wide consensus for its use across the industry.

The team started by gathering a set of information needs and objectives for digital engineering outcomes, which proved to be strongly aligned with research underway at the Systems Engineering Research Center (SERC) on DE benefits and measures described in section 1.3. This formed the basis for definition of the DE measurement framework described in the remainder of this document.

1.4 State of the Practice

Several organizations have performed research studies on digital model-based engineering that have factored into this DE measurement framework. The SERC at Stevens Institute of Technology (supported by researchers at Virginia Tech) collaborated with INCOSE and the NDIA Systems Engineering Division on a survey to benchmark the maturity of Model-Based Systems Engineering (MBSE) practices across an enterprise. Survey questions were derived from the INCOSE Model-Based Enterprise Capability Matrix[14] and included questions on the maturity of DE/MBSE measurement. An additional study focused on deriving a DE Metrics framework from that survey and other literature provided an additional broad categorization of the DE/MBE measurement landscape. These studies created a framework for describing the benefits of DE but also discovered that actual measurement in the community is still at its early stages. Analysis results are published in the following SERC reports:

- SERC-2020-SR-001, Benchmarking the Benefits and Current Maturity of Model-Based Systems Engineering across the Enterprise: Results of the MBSE Maturity Survey.[15]
- SERC-2020-SR-003, Summary Report on Digital Engineering Metrics[16]

The SERC survey analysis substantiated an industry early in its digital transformation progress with low maturity in measures of digital engineering effectiveness, with much room for future improvement but optimistic on the benefits and value to be achieved on DE. The SERC additionally conducted a literature review of digital engineering benefits and measures, whether perceived, observed, or measured. As depicted in Figure 1-3, value assessments were summarized in 5 categories:

- Quality
- Knowledge Transfer
- Velocity/Agility
- User Experience
- Adoption

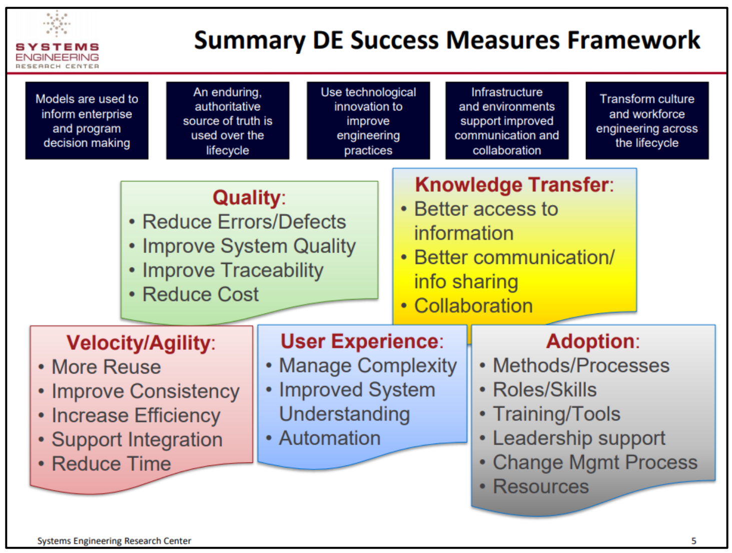


Figure 1-3: SERC Digital Engineering Success Measures

Early discussion between the subject matter experts in the DE Measurement Working Group and members of the SERC research team settled on eight primary benefits of DE. These primary benefits (things an enterprise should do with data and models) were linked to secondary benefit measures and organizational adoption measures in a causal analysis.[17] This causal analysis and the expertise of the working group created the initial set of measurement concepts and constructs in this framework, which were used to define the initial version of the ICM Table presented in section 7. The initial set of constructs are intended to isolate those measurements that are most closely linked to the primary benefits of DE. This is not intended to replace any other measurement constructs that are associated with other disciplinary engineering processes.

The primary benefits are linked causally to potential secondary benefit measures as shown in Table 1-1. This is provided as a historical example and not intended to reflect the final specifications in this document.

Table 1-1 - Primary Benefits and Secondary Benefit Measures from the Causal Analysis

Primary Benefits	Description	Secondary Benefit Measures
Higher level support for automation	Use of tools and methods that automate previously manual tasks and decisions	Greater use of tools, easier to make changes, reduce time, reduce effort, increase consistency, increase

Primary Benefits	Description	Secondary Benefit Measures
		productivity, increase efficiency, improve system quality, reduce cost
Early Verification and Validation (V&V)	Moving tasks into earlier developmental phases that would have required effort in later phases	Reduce defects, reduce rework, reduce effort, reduce time
Reusability	Reusing existing data, models, and knowledge in new development	Improve collaboration, increase productivity, improve system quality
Increased Traceability	Formally linking requirements, design, test, etc. via models	Better requirements generation, reduce rework, reduce effort, improve system understanding, better decision making
Strengthened Testing	Using data and models to increase test coverage in any phase	Reduce defects, reduce rework, reduce time, reduce effort, increase productivity, increase efficiency, improve system quality
Better Accessibility of Information (ASoT)	Leveraging an Authoritative Source of Truth (ASoT) to increase access to digital data and models to increase the involvement of stakeholders in program decisions	Easy to make changes, improved system understanding, reduce time, improve system quality
Higher Level of Support for Integration	Using data and models to support integration of information and to support system integration tasks	Better analysis capability, reduce rework, reduce time, increase efficiency, increase effectiveness, improve system quality, reduce cost, increase confidence
Multiple Model Viewpoints	Presentation of data and models in the language and context of those that need access	Greater use of tools, easier to make changes, improve collaboration, improve system understanding, increase stakeholder involvement, improve architecture

1.5 Contributors

Table 1-2: PSM DE Measurement Framework Editors

Editors	Organization
Joseph M. Bradley	Main Sail, LLC Leading Change, LLC
Cheryl Jones	US Army DEVCOM AC Practical Software and Systems Measurement (PSM)
Geoff Draper	L3Harris Technologies AIA Engineering Management Committee NDIA Systems Engineering Division
Tom McDermott	Systems Engineering Research Center (SERC) Stevens Institute of Technology
Paul Janusz	US Army DEVCOM AC Practical Software and Systems Measurement (PSM)

Table 1-3: Additional Authors and their Organization

Authors	Organization
Lennis Bearden	L3Harris Technologies
Giacomo Gentile	Collins Aerospace
Richard Halliger	Volkswagen Group
Drew Miller	The Boeing Company
Young Park	Collins Aerospace
Bob Scheuer	The Boeing Company NDIA Systems Engineering Division AIA Engineering Management Committee

Table 1-4: Core Team Contributors and their Organization

Core Team	Organization
Kaitlin Henderson	Virginia Tech
Al Hoheb	Aerospace Corporation
Bill Luk	BAE Systems AIA Engineering Management Committee
Lisa Murphy	Siemens Digital Industries Software
Jeff Nartatez	Office of the Undersecretary of Defense for Research and Engineering (OUSD R&E) [SAIC]
Garry Roedler	INCOSE NDIA Systems Engineering Division
Alejandro Salado	The University of Arizona
Frank Salvatore	Office of the Undersecretary of Defense for Research and Engineering (OUSD R&E) [SAIC]
Paul Segura	The Boeing Company AIA Engineering Management Committee
Natasha Shevchenko	Software Engineering Institute, Carnegie Mellon University
Marilee Wheaton	Aerospace Corporation INCOSE

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

Table 1-5: Additional Contributors

Additional Contributors	Organization
David Allsop	The Boeing Company
Sanjay Angadi	Ansys AIA Engineering Management Committee
David Cooper	BAE Systems
Mimi Davidson	Office of the Undersecretary of Defense for Research and Engineering (OUSD R&E) [SAIC]
John Dilger	BAE Systems
Paul Embry	L3Harris Technologies AIA Engineering Management Committee
Ann Hodge	Sandia National Laboratories
Mike McLendon	Retired
Robert Minnichelli	Aerospace Corporation
Gery Mras	Aerospace Industries Association (AIA)
Rosco Newsome	Ernst & Young (EY)
Bill Nichols	Software Engineering Institute, Carnegie Mellon University
Ryan Noguchi	Aerospace Corporation
Macaulay Osaisai	L3Harris Technologies
Steven Quinn	BAE Systems
Donna Rhodes	Massachusetts Institute of Technology (MIT)
Chris Schreiber	Lockheed Martin Corporation NDIA Systems Engineering Division
Paul Solomon	Retired
Brian Tenney	Lockheed Martin Corporation
Sundar Thyagarajan	L3Harris Technologies
Steven Turek	United States Air Force
Timothy Walden	Lockheed Martin Corporation
John Wood	Naval Information Warfare Command

2. Major Concepts

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

DE is generally a set of methods, processes, and tools for the life cycle definition, development, and sustainment of complex engineered systems. DE creates not only the product itself, but also the digital data and models that define and then support the product over its life cycle. Because DE processes help to define the capabilities of the eventual system, DE measures can serve as useful leading indicators for other product related measures. DE can produce independent products in support of delivered data, hardware, and software products such as digital twins or other model- or simulation-based executable systems. For DE, stakeholder concerns include actual users of the system and software, as well as the development teams, support teams, acquirer, user, and enterprise managers. In an integrated DE environment, all workers, at all security and management levels, have secure and immediate access to the digital information they need to do their work. The measures need to provide value to all stakeholders and inform diverse data and information needs.

A challenge with measures is both ensuring that they provide information needed to support decision making and that they are actually collected and 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 and that the decisions made using those measures are producing the intended outcomes. If not, other measures may be required, or additional training may be required for decision makers on how the measures can be utilized. For DE, the information is related to the primary benefits listed in Table 1-1. DE measures should inform the team, product managers, and/or the enterprise that they are achieving these benefits.

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 DE measurement framework, in order to establish a common understanding, so that measures can be implemented and used consistently with community consensus.

2.1 DE Work Decomposition

Decomposition of the DE process flow is generally associated with models and underlying data, and the digital infrastructure supporting them. All are important concepts in the measurement approach and have related specifications. Figure 2-1 shows a basic decomposition of the work associated with DE.

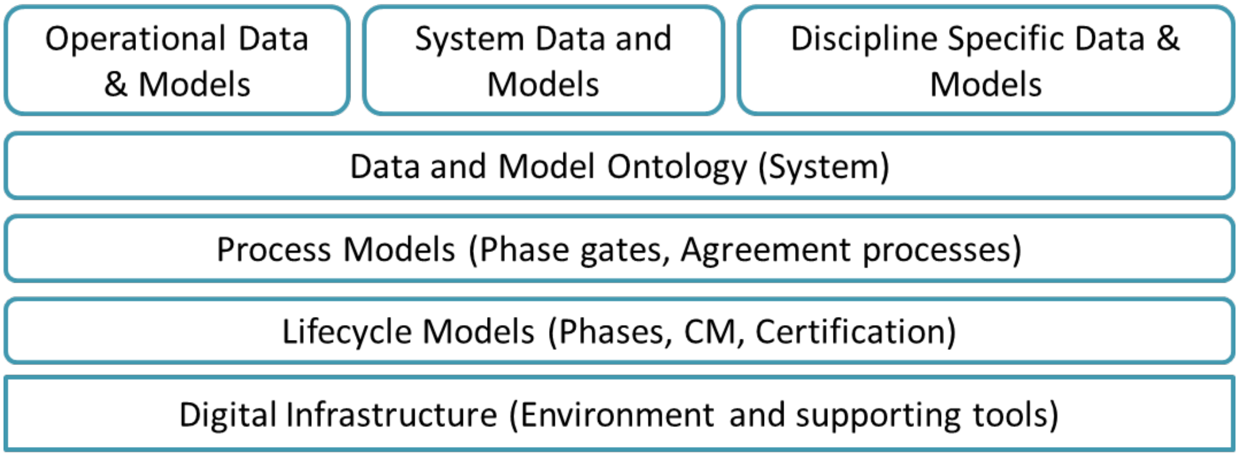


Figure 2-1: Decomposition of DE Processes

- Digital Infrastructure: The establishment of a set of computing assets and tools that support the other DE work efficiently and productively, as well as the training and organizational capabilities, to support this infrastructure. The digital infrastructure may be program and domain-specific and will integrate tools from multiple disciplinary practices. Work requires an established and up-to-date infrastructure, which may be developed incrementally while maintaining the integrity of the digital content and its timeliness. The digital infrastructure must support the information needs and related measurement data for the organization.
- Life cycle Models: DE supports multiple practices and life cycle approaches. DE measures are generally associated with life cycle phases, decision points, and information needs. The measurement model implementation must be tailored to the specifics of the system/program life cycle(s).
- Process Models: Work is planned and implemented through a set of defined processes that evolve and produce a set of life cycle artifacts in digital form designed to integrate across the products, people, and processes involved in the project. A DE process model defines stakeholder roles, digital artifacts, when they are required, how they are used, how they are managed, and how data is produced and consumed by the stakeholders.
- Data & Model Ontology: DE artifacts are maintained in a repository system, hereinafter simply call a repository, referred to as an Authoritative Source of Truth (ASoT). In a DE process stakeholders work from the same data and models. This repository consists of sets of application specific data models, which define how data is stored and accessed, and a set of domain ontologies, which define more generic concepts and relationships in the domain that support sharing of data and knowledge. In order for work to proceed efficiently, all users of the repository must be able to work from a common taxonomy and underlying set of ontological relationships maintained by the DE toolsets. An important benefit of DE is the potential ability to automate management of data and models, such that a change in one area of the ASoT is automatically reflected in all other areas. Work must include the development and maintenance of an appropriate data and model ontology.
- Operational, System, and Discipline Specific Models: DE is primarily concerned with the development and support of models and the data used by the models to support life cycle decisions. There is not a single model, but a set of models used to define the operational use of the system, analyze discipline specific concerns, and manage the relationships between individual models. In MBSE, the System Model is the result of a unique work activity that is

used as the central repository for design decisions that span multiple engineering and business concerns; design decisions are captured as model elements in that System Model.[18] Modeling concerns include abstraction, correctness, completeness, accuracy, authority, and validation. All of these affect the nature and amount of work necessary to develop and support models.

2.2 DE Concepts

2.2.1 Authoritative Source of Truth

The concept of an Authoritative Source of Truth (ASoT) is central to the use of DE. Use of the ASoT requires a set of digital artifacts that is structured such that every artifact (data element or model element) is owned by a single entity and managed in only one place. In use, linkages to these artifacts are by reference only. Because all other DE activities refer back to the primary "source of truth" location, updates to the artifact in the primary location propagate to the entire system without the possibility loss or duplication.[19] By definition an authoritative source of truth is an entity such as a person, governing body, or system that applies expert judgement and rules to proclaim a digital artifact is valid and originates from a legitimate source.[20]

The authoritative source of truth for a digital artifact serves as the primary means of ensuring the pedigree, credibility and coherence of the digital artifact that its creators share with a variety of stakeholders. It gives stakeholders from diverse organizations and distributed locations the authorization to access, analyze, and use valid digital artifacts from an authoritative source. The owners of digital environments or the community for digital engineering ecosystems provides stakeholders with an authoritative source of truth that assures confidence in the quality of the digital artifact across disciplines, domains, and life cycle phases.

In order to do so, a digital artifact's authoritative source of truth should meet four conditions. First, the digital artifact originates from a repository recognized by a governing entity as a System of Record (SoR). Second, the majority of experts accepts the credibility, accuracy, relevance, timeliness, and trustworthiness of a digital artifact because it meets their "criteria of truth". For example, in the MBE domain, the digital artifact may meet the criteria of truth when most stakeholders agree that the preponderance of evidence upholds the validity of the digital artifact because it represents a commonly accepted perspective of reality. Third, a digital artifact's source is an authoritative when most experts agree that the source is legitimate. Finally, the digital artifact originates from a technological system that maintains its integrity and reinforces the conditions. If the SoR satisfies the four conditions; then, it is the Authoritative Source of Truth for its digital artifact.[21]

2.2.2 Model Element

The ISO/IEC/IEEE draft MBSSE standard defines a model elements as atomic (elementary) items that represent individual components, actions, states, messages, properties, relationships, and other items that describe composition, characteristics, or behavior of a system.[22]

A model element is an abstraction drawn from the system being modeled, representing an elementary component of a model. The number and type of model elements in the ASoT will be determined by the development process. Delligatti states that if the system model is the central repository for design decisions, each design decision is captured as a model element or relationship between elements.[23] There is no predefined categorization of elements – they can be defined by the underlying ontology of the system model or of the tool used to create and manage the models.

DE processes do not explicitly determine and create model elements, they are created as a natural part of the modeling process. However, the DE measurement approach and associated measures should recognize a defined concept of a model element such that 1) the relative size of the DE effort can be measured and compared to other efforts or plans, and 2) the quality of the DE design decisions (correctness and completeness) can be measured.

As one example, Sparx Systems defines the following Model Element Objects associated with SysML:[24]

> Model - Creates a Package containing a SysML Model.
>
> Model Library - Creates a Package containing a SysML Model Library.
>
> View - Creates a stereotyped Class that defines a SysML View of a system, from the perspective of a SysML View Point.
>
> View Point - Creates a stereotyped Class that defines a SysML View Point, which specifies the rules and conventions for the construction and use of Views.
>
> Stakeholder - Creates a stereotyped Class that defines a SysML Stakeholder.
>
> Package - Groups model constructs in a single unit of containment.

As another example, IBM defines UML model elements into the following four categories:[25]

> Structural model elements - These elements model the static parts of a system. Some examples include classifiers such as actors, classes, components, information items, and nodes.
>
> Behavioral model elements - These elements model the dynamic parts of a system. Typically, you find behavioral model elements in state machine and interaction diagrams. Some examples include activities, decisions, messages, objects, and states.
>
> Organizational model elements - These elements group model elements into logical sets. A package is an example of an organizational model element.
>
> Annotational model elements - These elements provide comments and descriptions.

In order to extract measurement information from the ASoT, the project must determine the type of model elements it will measure. These will be constrained by the tools selected. Additional work is required to standardize on guidance for model elements that are most relevant to DE measurement.

2.2.3 Life cycle Phase

A life cycle is the evolution of a system, product, service, or other human-made entity from conception through retirement.[26] Every developed product has a life cycle, even if it is not formally specified. The purpose of specifying a life cycle is to establish a framework for meeting stakeholder needs in an orderly and efficient manner, increasing the likelihood for optimizing the use of resources against the schedule. A life cycle consists of phases, with each life cycle phase having a purpose and an outcome. Life cycle phases and decision gates for transition between phases can be used to mature the product design by establishing specific checkpoints to ensure that acquirer and user needs are properly understood and met before committing time and resources too early. These checkpoints provide the development team, support team, management (internal and external), and other key stakeholders an incremental view of the progress being made with respect to planned expectations for that point in the life cycle, as well as related risks and issues. The checkpoints also provide opportunities for follow-on course correction to help ensure the project's successful mission delivery.

Each life cycle phase represents a team's work on the product leading to a release, as well as the work required to support, update, and then retire the product after a release. Each life cycle phase is an agreement between stakeholders in the project to create a product baseline and a decision point (called phase gates) that formally defines how the project should move forward. Each phase can have one or more gates. Each life cycle phase produces a set of artifacts that are used by the following phases. The total set of these artifacts is termed the baseline. Often programs use a phase gate review process to determine artifact expectations or suitability for the next phase. Each gate has a target status; when the product has that status, the product can pass through the gate.[27]

In a DE-based project, all artifacts are managed in the ASoT. Configuration management of these artifacts from phase to phase and gate to gate must be assured to create consistency of artifacts across stakeholders. A primary benefit of DE is to improve the quality of the product as it moves from phase to phase. As many of these artifacts are not the actual product, it is important to maintain a formal process to assess their quality at each phase.

3. Terms and Definitions

Terms and definitions used in this document are derived from the following primary sources:

- ISO/IEC/IEEE 15288:2015 Systems and software engineering — System life cycle processes
- ISO/IEC/IEEE CD 24641:2020 (E) Systems and software engineering – Methods and tools for Model-based systems and software engineering
- ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description
- ISO/IEC/IEEE 24765, which is published periodically as a "snapshot" of the SEVOCAB (Systems and Software Engineering Vocabulary) database and is publicly accessible at computer.org/sevocab
- Defense Acquisition University (DAU) as collected on the U.S. Undersecretary of Defense for Research and Engineering Digital Engineering Website (https://ac.cto.mil/digital_engineering/)
- Digital Information Exchange Working Group (DEIX) Topical Encyclopedia (https://www.omgwiki.org/MBSE/doku.php?id=mbse:topical_encyclopedia_for_digital_engineering_information_exchange_deixpedia)
- Model Based Engineering Forum website (modelbasedengineering.com)

3.1 Digital Engineering

The following are general terms and definitions associated with DE that are used throughout this measurement framework:

Table 3-1: Digital Engineering Terms and Definitions

Term	Description
Digital Engineering Ecosystem	An interconnected infrastructure, environment, and model-based engineering (MBE) methodology that enables the exchange of digital artifacts from an authoritative source of truth. It uses rule-based transactions for its stakeholder-network during the entire system life cycle. (*DEIX Topical Encyclopedia*)
Digital Engineering	An integrated digital approach that uses authoritative sources of systems' data and models as a continuum across disciplines to support life cycle activities from concept through disposal. *(DAU Glossary)*
Digital Thread	An extensible, configurable, and component enterprise-level analytical framework that seamlessly expedites the controlled interplay of authoritative technical data, software, information, and knowledge in the enterprise data-information-knowledge systems, based on the Digital System Model template, to inform decision makers throughout a system's life cycle by providing the capability to access, integrate, and transform disparate data into actionable information. *(DAU Glossary)*
Digital Twin	An integrated multi-physics, multiscale, probabilistic simulation of an as-built system, enabled by Digital Thread, that uses the best available models, sensor information, and input data to mirror and predict activities/performance over the life of its corresponding physical twin. *(DAU Glossary)*
Digital Artifact	The artifacts produced within, or generated from, the digital engineering ecosystem. These artifacts provide data for alternative views to visualize, communicate, and deliver data, information, and knowledge to stakeholders. *(DAU Glossary)*

Term	Description
	A digital artifact is any combination of professional data, information, knowledge, and wisdom (DIKW) expressed in digital form and exchanged within a digital ecosystem. (*DEIX Topical Encyclopedia*)
Model-based	Represented using a formalism which has a formal syntax and semantics, usually with a theoretical basis, and expressible in a symbolic language Note 1 to entry: Presentation of such models is often graphical but the definition mandates that the graphical representation be translatable into a symbolic language, thereby constraining interpretation of the graphical representation. Note 2 to entry: In order to satisfy specific stakeholder concerns, "model-based" is often used as a qualifier to characterize a kind of design, or practice, e.g. model-based system engineering, model-based design, model-based specification. *(ISO Online browsing platform)* An umbrella term that describes a technology approach where rigorous visual modeling principles and techniques form the technical foundation for an engineering or development process in order to increase its efficiency and productivity. (*modelbasedengineering.com*)
Model-based development	Development that uses models to describe the behaviour or properties of an element to be developed. *(ISO Online browsing platform)*
Model-based engineering	A software and systems development paradigm that emphasizes the application of modeling principles and best practices throughout the life cycle. *(ISO Online browsing platform)*
Model library	A group of model elements that are intended to be reused in other models. (*modelbasedengineering.com*)
Model	A mathematical or physical representation (i.e., simulation) of system relationships for a process, device, or concept. *(IEEE Standards Dictionary, IEEE Std 1641)* Representation of a real world process, device, or concept. *(IEEE Standards Dictionary, IEEE Std 2413-2019)* A representation of an object or system of interest. A Model has a well-defined abstraction boundary, sometimes referred to as a System Boundary, which defines what is inside and outside the scope of the subject system. The complexity of large models is sometimes managed by projections on the model elements they contain, where the projections are called Views, which are defined from the perspectives (Viewpoints) of various system stakeholders. (*modelbasedengineering.com*)
System model	An interconnected set of model elements that represent key system aspects including its structure, behaviour, parametric, and requirements. *(earlier version of ISO/IEC/IEEE 24641:2000 (E) this is not included in the latest release)* A system model - is used to represent a system and its environment - may comprise multiple views of the system to support planning, requirements, architecture, design, analysis, verification, and validation - is a representation of a system with various degrees of formalism often expressed as a combination of descriptive and analytical models. The system model is an integrating framework for other models and development artefacts including text specifications, engineering analytical models, hardware and software design models, and verification models. In particular, the system model relates the text requirements to the design, provides the design information needed to support

Term	Description
	analysis, serves as a specification for the hardware and software design models, and provides the test cases and related information needed to support verification and validation. *(IEO/IEC/IEEE 24641:2021 – DIS)*
Descriptive model	Model that shows an interconnected set of model elements which represent key system aspects including its structure, behavior, parametric, and requirements *(ISO/IEC/IEEE 24641:2000 (E))*
Discipline specific model	Representation of a system, or system elements from the perspective of a discipline addressing domain specific concerns where the model elements come from a specific discipline. *(ISO/IEC/IEEE 24641:2000 (E))*
Digital System Model	A digital representation of a defense system, generated by all stakeholders, that integrates the authoritative technical data and associated artifacts, which defines all aspects of the system for the specific activities throughout the system life cycle. *(DAU Glossary)*
Model element	Atomic (elementary) items that represent individual components, actions, states, messages, properties, relationships, and other items that describe composition, characteristics, or behavior of a system *(ISO/IEC/IEEE 24641:2000 (E))*
Model configuration item	A logical part of the model that is maintained in a controlled fashion, i.e., have a trackable revision history. *(ISO/IEC/IEEE 24641:2000 (E))*

3.2 Other (Need to Decide on the Categorization)

The following are relevant terms and definitions extracted from the CID document:

Table 3-2: Other Terms and Definitions

Term	Description
Capability	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.
Product	A product is the output of an enterprise that can be produced. There are four generic product categories: hardware (e.g., engine mechanical part); software (e.g., computer program); services (e.g., transport); and processed materials (e.g., lubricant).
Requirement	The need or demand for personnel, equipment, facilities, other resources, or services, by specified quantities for specific periods of time or at a specified time.
Problem Report	Identified issue with a digital artifact, either in the product or any other artifacts used to support it. Once approved for implementation, a Change Request may be created, or the Problem Report may be used to track implementation.
Defect	A defect is a condition in a product, that does not meet its requirements or end-user expectation, causes it to malfunction or to produce incorrect/unexpected results, causes it to behave in unintended ways, or leads to quality, cost, schedule, or performance shortfalls. Any digital artifact used to directly define, produce, or support the product should be included in the set of defects and process to manage them. Defects may be documented in problem reports, or they may be added to the planned work for consideration in future life cycle phases. Escaped Defects are defects detected or resolved after release of the baseline artifact containing the defect. Defects are generally tracked separately for internal and external baselines.

Term	Description
	Contained Defects, also known as Saves, are defects detected and resolved within a phase before internal or external baseline deliveries of the artifact and version containing the defect. Imperfection or deficiency in a work product or characteristic that does not meet its requirements or specifications. *(IEEE Standards Dictionary, IEEE Std 2675-2021)*
Change	Revision that adds, removes, or modifies any aspect of a digital artifact as managed in the ASoT.
Change Request	Requested change to the digital artifact. Some organizations may use Problem Reports instead of separate Change Requests to track issues.
Stakeholder	Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations (*ISO/IEC/IEEE 15288:2015 Systems and software engineering--System life cycle processes*)

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 4-1. More detail on the discussions in this section can be found in Practical Software and Systems Measurement (John McGarry (Author), 2001)[1]

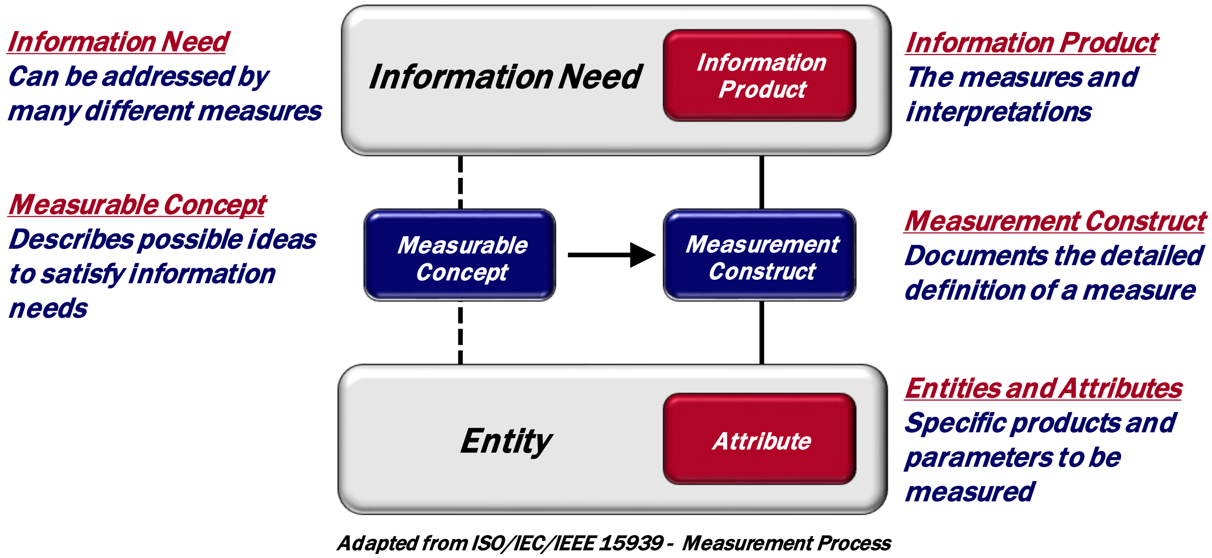


Figure 4-1: Information Model - High-Level View

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 4-2 illustrates the structure of a measurement construct in more detail.

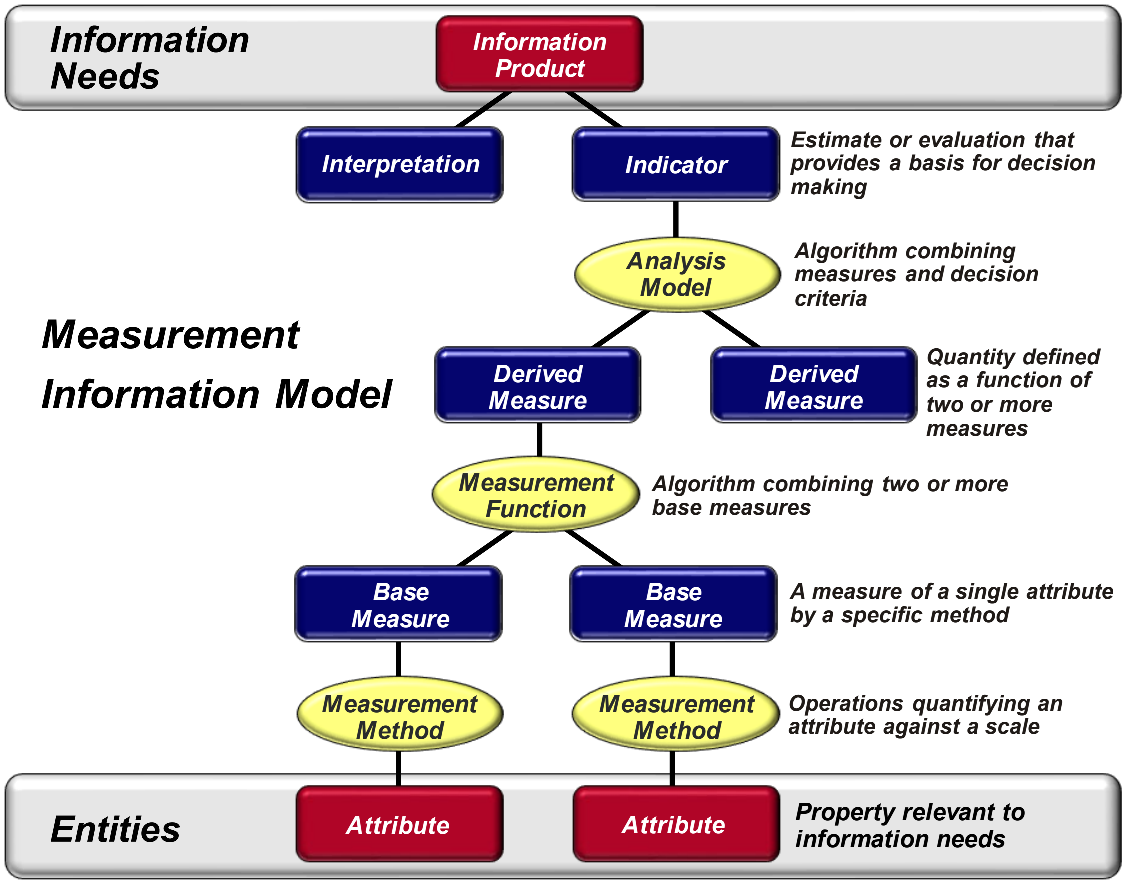


Figure 4-2: Measurement Information Model

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 4-3 contains a specific example of this, for the defect detection measure that is specified in Part 2 Section 8. The measurement specifications in Section 8 detail the information needs, base measures, derived measures, and analysis models for each proposed measure.

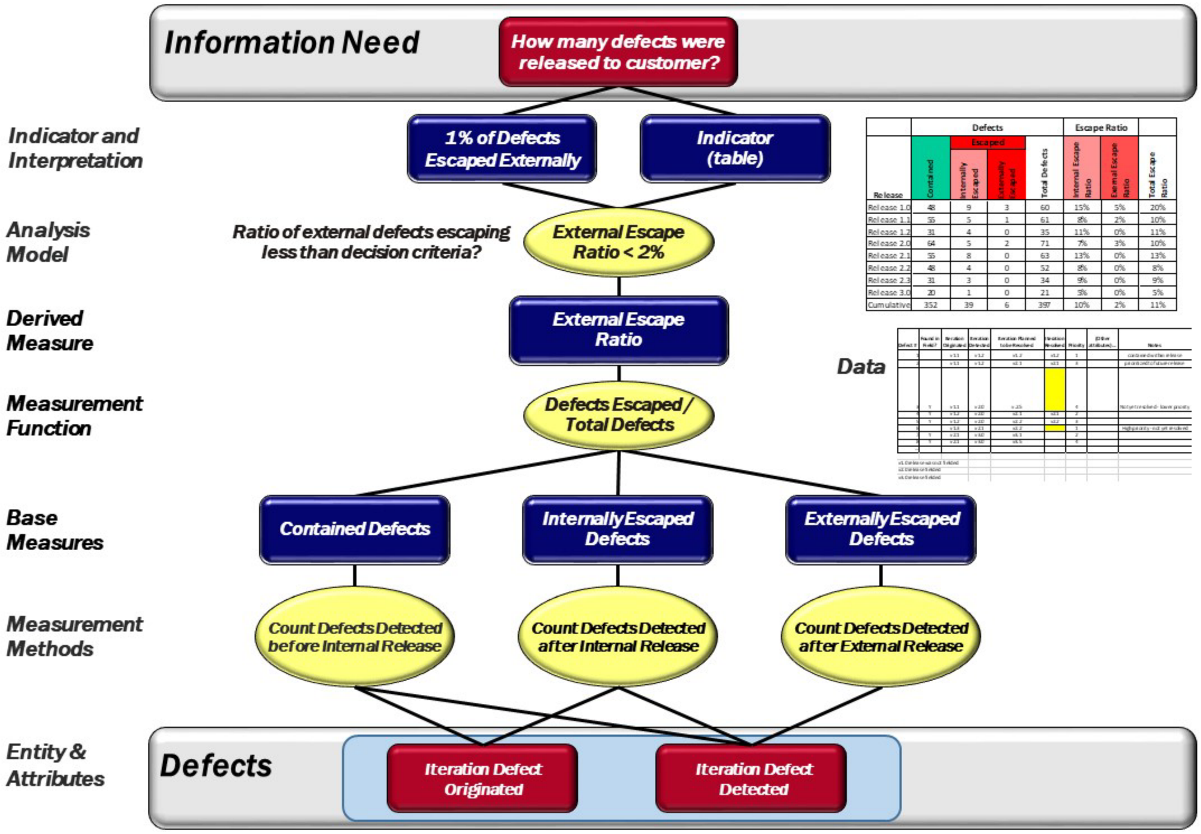


Figure 4-3: Mapping Data to Measures

5. Measurement Principles

The "Information Categories-Measurable Concepts-Measures" (ICM) Table provides the PSM DE measurement framework detailing common information needs and measures that are effective for DE approaches. The information needs address project 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. The ISO24641 foresees a specific task (chapter 4.1.2) in the definition of the goals in adopting MBSSE and of the methodology to define the achievement of the same goals. To do this, there are two parts that standard suggests: establishing a methodological part and a tools part. It is therefore stated that the adoption of MBSSE implies having the ability to address both aspects. The emphasis on tools is needed since the adoption of MBSSE is inextricably linked to the tool that needs to be designed to allow for the collection of measures easily.

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.
- As organizations stand up or start digital engineering efforts, it will be valuable to create an initial set of measures at that point. Otherwise, the organization is trying to create measures once the operation is already up. This is likely to delay the development of measures.
- The selected measures should have an identified stakeholder, inform decisions or answer key programmatic questions, and drive actions. As measures are developed, the concept of a leading or responsible stakeholder can bring value. "Who owns this measure?" is a good question to ask and answer. This helps keep measures in use that are valuable to the organization rather than a disembodied set of measures that no one feels attachment to or takes responsibility for. There may stakeholders that are interested in the measure reported upon, which may be different that the "owning stakeholder".
- Project and enterprise measures are included: not all can be aggregated. While some measures provide direct information, it may also be related to another quantity or measure that is important, yet not be directly aggregated from the reported measure.
- 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 set of measures are process agnostic, but they were specifically developed for digital engineering. Other PSM materials represent a broader set of materials and processes.
- The collection of measures should be automated by utilizing the functionality of existing tools or by creating custom tools to the extent practical. These tools should be integrated with business workflows, used development processes, and with other adopted DE practices.
- For the highest priority measures, sample measurement specifications have been developed that detail the identified measures. Measurement specifications have been developed for the following Information Categories:
  - Schedule and Progress

- Size and Stability
- Product Quality
- Process Performance
- Technology Effectiveness

See Part 2, Section 8 for these specifications. The ICM table and the sample measurement specifications can also be found at http://www.psmsc.com/DEMeasurement.asp.

6. Next Steps

This version of the PSM DE measurement framework is an initial set of measures, where subject matter experts in the nascent field of digital engineering have proposed these measures. Several of these have proven to be useful in practice and several that are more exploratory, but that we expect to be of use based on the expertise of the participants. Additional measures will be considered and added in future releases.
Known future additions include:

- Measure people *adoption*, and enterprise process adoption (adoption)
- Analyze breadth of *usability*, and issues with usability (user experience)
- Measure *productivity* indicators (velocity/agility)
- Generate *new value* to the enterprise (quality and knowledge transfer)
- Measure Supportability and Maintainability (impact assessment agility)
- Identify typical digital artifacts

7. Information Categories, Measurable Concepts, Measures (ICM) Table

Table 7-1: Information Categories, Measurable Concepts, and Measures

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
Schedule and Progress	Architectural Completeness	How complete is the functional architecture? Does the architecture provide coverage of all required functions? Is the functional architecture sufficiently complete to proceed with design at acceptable risk? What is the extent of traceability across digital model elements?	What is the amount of schedule and design risk for each project? What is the architecture progress across projects? What is the extent of model traceability across projects?	Functional Architecture Completeness and Volatility Model Traceability		Functional Architecture Completeness and Volatility Model Traceability
Schedule and Progress	Model Coverage	What is our progress in completing the digital model? What is the extent of traceability across digital model elements?	What is the modeling coverage and progress of the digital engineering capability across projects? What is the current upper limit of the digital engineering capability?	Total Elements Modeled Elements	Measurement is against only the content that is modeled or “digital", including requirements, functional elements, logical elements, interfaces, etc. Model elements are created to fulfill the functional requirements and functional interfaces allocated during the architectural design phase.	Model Coverage Model Traceability

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
Size and Stability	Functional Size and Stability	What is the size and scope for the digital engineering project or product? How much work must be done? How many functions and interfaces have been identified in the system functional architecture? How much is that changing? How does DE product size relate to estimates and measures of cost, schedule, productivity, or performance?	Is the current project similar in size and scope to historical projects? Is the work scope changing? Is the schedule and effort sufficient to address changes? How does DE product size relate to estimates and measures of cost, schedule, productivity, or performance?	Digital Engineering Product Size (Model Elements) Functions Identified Functional Change Requests	In development, product size can be determined by a count of model elements. Function Volatility includes the aspects of continuing to identify new functions and/or having the functional allocation continue to change. In maintenance, change requests are often used as a measure of work scope.	Product Size Functional Architecture Completeness and Volatility
Product Quality	Functional Correctness	How many defects were detected (contained) prior to internal release? For each major release, how many defects were detected by the external user (escapes)? What is the ratio of escaped defects (internal and external) to all defects? Can the use of digital engineering detect defects earlier (e.g., prior to implementation)?	How many defects were released (escaped) to an external user? How much has the use of digital engineering contributed to the earlier detection and containment of defects? Has the defect detection curve shifted to the left?	Defect Detection		Defect Detection

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
Product Quality	Functional Correctness	Are we finding and removing defects early in the lifecycle? Are we finding and removing defects prior to operations? How many contained defects in the requirements, architecture, or design phases would have affected the operational product?	Is product quality improved using digital engineering methods?	Defect Detection (Contained, Escaped) Defect Resolution	For digital engineering focus on the defects for modeling and simulation (including drawings).	Defect Resolution
Product Quality	Functional Correctness	Is rework identified and managed? How much rework effort is spent maintaining planned or unplanned changes to digital engineering work products across the life cycle?	How much is rework reduced through use of digital engineering? Can changes to engineering work products be implemented more easily and with less effort in a digital engineering environment relative to traditional methods?	Acceptance of Completed Work (Model Elements, Artifacts) Rework or Rework Defects		Adaptability and Rework
Product Quality	Functional Correctness	What traceability gaps or defects exist in the digital model? Does model traceabilility support change impact assessments	Is architectural traceability improved using digital engineering methods relative to traditional approaches?	Model Traceability Gaps		Model Traceability

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
		(requirements, design, compliance)?				
Process Performance	Process Effectiveness	How many released, validated system definitions/analyzed elements were functionally correct, but returned for rework?	Is the organization learning how to reduce the number of errors released to operations?	Modeling Errors		Model Coverage
Process Performance	Process Effectiveness	Are we containing defects in early phases using models and shared information?	Are we finding and removing defects earlier using digital engineering methods relative to traditional methods?	Defect Detection Defect Resolution Defect Containment (Escaped) Rework Effort Reworked Model Elements	For digital engineering focus on the defects for modeling and simulation (including drawings). The focus is whether the process is improved using digital engineering, versus the raw numbers.	Defect Resolution
Process Performance	Process Efficiency - Automation	What percentage of artifacts are automatically generated from digital models? To what extent are artifacts facilitating program reviews?	What is the extent of automation across projects? How much is automation contributing to meeting our performance and quality objectives? What is the return on investment for automation? How much can cycle	Digital Engineering Product Automation Cycle Time		Product Automation

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
			time be reduced through automation of digital engineering tasks?			
Process Performance	Process Efficiency - Speed	How long does it take to deploy an identified feature/capability? How long does it take to deploy a viable product for operational use after a request is received? Where is the deployment bottleneck; in planning/backlog, implementation, or deployment of the implemented capability?	How long does it take to develop a digital engineering model or product? Does the process performance meet business objectives?	Deployment Lead Time Cycle Time	Proper analysis also requires an enterprise approach for quantifying size or complexity of work products.	Deployment Lead Time
Process Performance	Process Efficiency	Is productivity improving over time (normalized model element/artifact delivered by effort)? How many model elements/artifacts are being produced per release? How many can be expected to be produced for the next release?	Is productivity improving over time (normalized model element/artifact delivered by effort)? Is our productivity sufficient to meet our customer's needs? How much is productivity increased through the use of digital engineering?	Productivity Model Elements/Release Artifacts/Release		
Technology Effectiveness	Technology Performance	What is the runtime performance of the capability or system?	How much does runtime effect interoperability of the	Runtime Performance		Runtime

Information Categories	Measurable Concepts	Project Information Needs	Enterprise Information Needs	Potential Measures	Notes (Guiding Objectives)	Specification
		What is the likelihood that runtime performance will meet operational requirements (for each alternative solution)? Where are the runtime performance bottlenecks, and how can operational performance be optimized?	system? Where is redesign needed to solve compatibility issues?	Elapsed Time		

8. Measurement Specifications

8.1 Functional Architecture Completeness and Volatility

Measure Introduction	
Description	This measure is used to evaluate progress toward completion of a functional architecture in a system or product development. A functional architecture is foundational for aligning the problem space with solution space. Completeness and stability (i.e., absence of volatility) in the functions comprising the functional architecture provide a direct view into the maturity of a system development with Digital Engineering. At the team level, architecture progress is measured based on declared functions and associated interfaces at each designated boundary and associated level(s) of design in the system. At the program or enterprise level, this measure can be used to monitor overall progress toward definition of a complete functional architecture that emerges from a system's functional requirements and containing all levels of a system's functional design. It may also provide an indication as to the fidelity of a system functional definition as each level is iteratively decomposed into member functions and interfaces across architectural levels and boundary partitions. It may further be used to augment measurement of product quality by indicating product readiness with respect to expected capability/performance, allocated functionality, or verified functional traceability to source requirements.
Relevant Terminology	Function: A task, action, or activity that must be accomplished to achieve a desired outcome - originating from source functional requirements, use cases, and functional decomposition Source Functional Requirement: Statement that identifies what results a product or process shall produce; a requirement that specifies a function that a system or system component shall perform Source Functions: Functions identified from source requirements Derived Functions: Functions that are not explicitly stated in source requirements, but are inferred from contextual requirements or decomposition during analysis, design, or architecture Allocated Functions: Function that levies all or part of the performance and functionality of a higher-level requirement on a lower level architectural element or design component Function Volatility: Rate of change over time in function identifications or allocations.

Information Need and Measure Description	
Information Need	How complete is the functional architecture? Does the architecture provide coverage of all required functions? Is the functional architecture sufficiently complete to proceed with design at acceptable risk?
Base Measure 1	Source Functions - Number of source functions within defined boundary partition [integer]
Base Measure 2	Derived Functions - Number of derived functions within defined boundary partition [integer]
Base Measure 3	Allocated Functions - Number of source functional requirements identified, decomposed, and allocated to design components with complete traceability within defined boundary partition [integer]
Base Measure 4	Milestone Date [date] $T0 = Date_{Start}$; $T1 = Date_{SRR}$; $T2 = Date_{SFR}$; $T3 = Date_{PDR}$; $T4 = Date_{CDR}$

Derived Measure 1	Total Functions Identified (Committed) = Number of source functions identified + Number of derived functions identified [integer]
Derived Measure 2	Functions Allocated (Completed) = Number of source functions allocated + Number of derived functions allocated [integer]
Derived Measure 3	% of Functions Allocated = Functions Allocated (Completed) / Total Functions Identified (Planned) [real]
Derived Measure 4	Function Volatility (Identification) = (Change in Number of Identified Functions) per Increment of Time [integer] Ideally, Function (Identification) Volatility = 0 after System Functional Review (SFR) with 100% of functions identified within defined boundary partition
Derived Measure 5	Function Volatility (Allocation) Volatility = (Change in Number of Allocated Functions) per Increment of Time [integer] Ideally, Function (Allocation) Volatility = 0 after Preliminary Design Review (PDR) with 100% traceability to source functions within defined boundary partition

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

In Figure 8-1, the graph shows the identification and completion of functions over time in a system development program. On the X-axis, key program milestones are identified that have a direct correlation to understanding the functional maturation over time. The Y-axis identifies functional counts based on what is presented at start of program by stakeholders (i.e., source functions) and identified by the design team (i.e., derived functions) over the course of the system design.

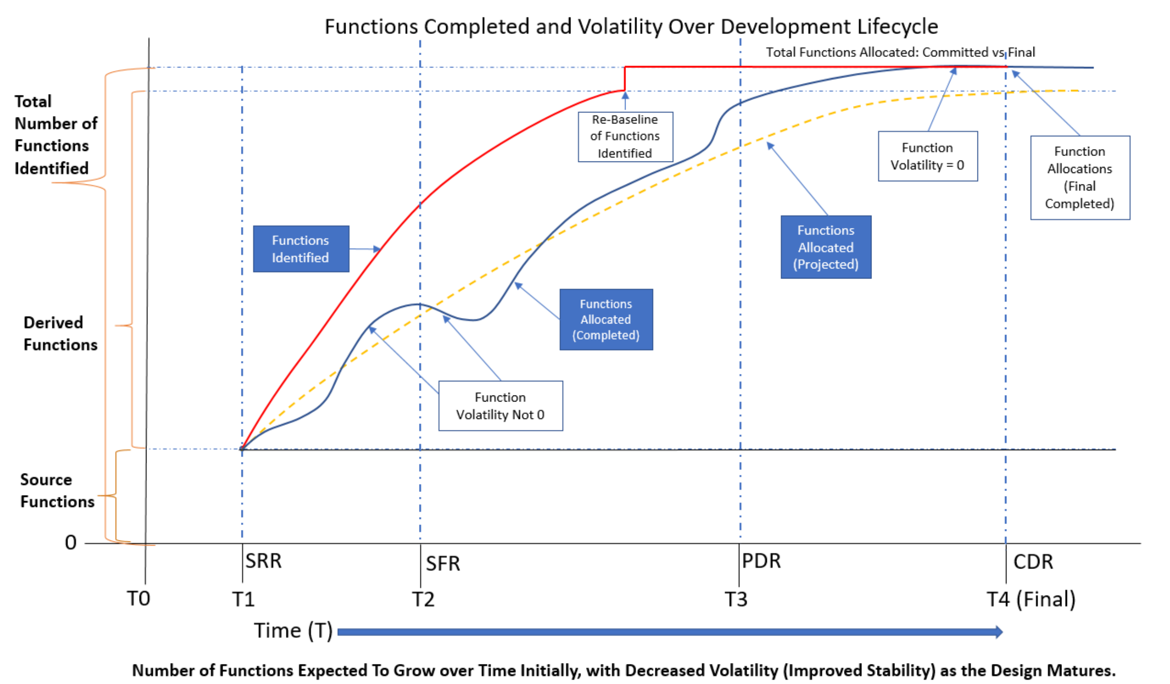


Figure 8-1: Functions Completed versus Plan and Volatility Over Time

The solid blue line represents the number of identified functions for a system over time of the system's development. The dashed orange line shows the projected allocated functions (identified or predicted) prior to the start of the effort. The solid orange curve shows number of functional allocations which have been achieved over time following establishment of the system requirements baseline for the program at SRR. At

</td></tr>
</table>

SRR, the functions identified reflect those established as source functions per the acquirer-supplier agreement.

Slope of the solid orange line at any point represents the function volatility being experienced on the program. An increase in slope of the line represents greater volatility while a decrease in slope indicates less volatility. As functions are identified and then allocated to design components, changes are observed in allocation counts for total identified functions. The fluctuations in the orange solid line show that functions may be identified or eliminated based on refinement of the design over time. Negative slope indicates volatility associated with a net reduction in total identified or allocated functions. Complete function stability is indicated when the slope of the functions allocated line is zero (functional identification and allocation over time), thus signaling that all identified functions have been allocated to components in the system. The offset of the solid blue line from the solid orange line represents a normal time lag associated with allocating identified functions via the design process.

Analysis Model

$$Function\ Completeness = \frac{Functions\ Completed\ (In\ Allocation)}{Total\ Identified\ Functions}$$

$$Function\ Volatility\ = |\#Added\ Functions| + |\#Changed\ Functions| + |\#Deleted\ Functions|$$

($Applicable\ to\ both$ Δ $Functions\ Identified\ or$ Δ $Function\ Allocations$)

Data Point	Time 0	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6*	Time 7	Time 8	Time 9
Source Functions	25	25	25	25	25	25	25	25	25	25
Derived Functions	TBD	75	75	75	75	75	70	80	80	80
Total Identified Functions	TBD	100	100	100	100	100	95	105	105	105
Change in Functions Total from Prior Time	N/A	+100	0	0	0	0	-5	+10	0	0
Functions Completed (in Allocation)	0	40	55	52	52	65	75	90	105	105
Function Allocations Remaining	N/A	60	45	48	48	35	20	15	0	0
Δ Function Allocations (from Prior Time)	N/A	+40	+15	-3	0	+13	+10	+15	+15	0
Function Volatility	N/A	40	15	3	0	13	10	15	15	0
Function Completeness	N/A	0.40	0.55	0.52	0.52	0.65	0.79	0.86	1.00	1.00

Figure 8-2: Functional Completeness & Volatility Analysis (Example Use Case)

* Function Completeness with Re-Baselined Functions Included

Analysis Model Considerations:

- Functional Completeness is Achieved When Functions Completed (In Allocation) Equals the Total Identified Functions (i.e., = 1)
- Function Stability is Achieved (i.e., Function Volatility = 0) when the total number of added functions, changed functions, and deleted functions for a given unit of time is zero when compared to the previous unit of time of a measurement
- The Functional Architecture Completeness and Volatility measure is complete when both Functional Completeness = 1.0 and Functional Volatility = 0 for the same unit of time.

General Considerations:

- This model gives an understanding of key characteristics centered on functional requirements and functional allocations to potential design components involved in the system under design.
- Understanding of functional analysis and architecting in the system are needed to firmly employ this analysis model and achieve reliable indication of program health or signal risks.
- Completeness refers to the full establishment of functions allocated to design elements of a system's architecture.

	• Volatility refers to the extent of change in total count(s) of involved elements. In this measure, those elements are functions, with volatility expressed as having a magnitude of positive, negative, or zero value. • Stakeholder vague desires need to be converted to explicit and unambiguous requirements statements prior to addressing system functional volatility. • The requirements baseline needs to be solidified at SRR per stakeholder agreement. Delay in achieving the requirements baseline will potentially introduce additional functional volatility and allocation delays.
Decision Criteria	• Have all functional requirements, use cases/scenarios, and other functional sources been identified? • What is the number of functions in each defined boundary partition (e.g., system design level, subsystem design level, specific boundary area, etc.)? Measures of Functional Architecture Completeness and Volatility can be key indicators in determining when the architecture is sufficiently mature to justify proceeding with system design at acceptable risk. If the architecture is incomplete or continuing to undergo significant changes, this may indicate a risk of future rework.

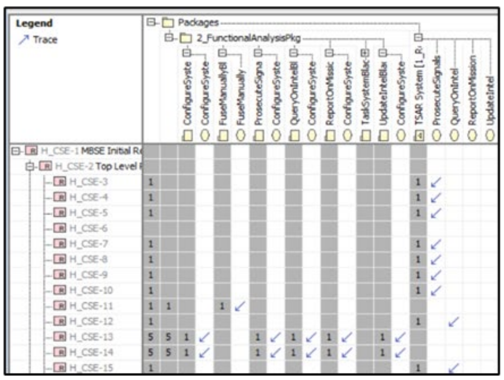
Additional Specification Information	
Information Category	Schedule and Progress
Measurable Concept	Work Unit Progress
Relevant Entities	Requirements, Use Cases, Design Level or Defined Boundary Partition, System Under Design
Attributes	Functions Identified, Functions Allocated, Function Allocations Completed for each entity
Data Collection Procedure	At the team level, data is collected at the end of each derivation increment of time by the team. Functions must be tested and satisfy "Done" criteria, with no orphan functions or functions with unterminated interfaces to be counted as completed. If a function does not satisfy "Done" criteria, then it is not considered "Complete" and it is not included in the Total Functions Allocated. For product 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 derivation increment of time by the team during the derivation review and considered during the planning session for subsequent lower-level functional definitions. The data is also aggregated and analyzed at summary levels across derivation increment or releases to ensure that the program is completing its committed functional assignments. Functional Completeness is Achieved When Functions Completed (In Allocation) Equals the Total Identified Functions (i.e., = 1) Function Stability is Achieved (i.e., Function Volatility = 0) when the total number of added functions, changed functions, and deleted functions for a given unit of time is zero when compared to the previous unit of time of a measurement The measure is complete when both Functional Completeness = 1.0 and Functional Volatility = 0 for the same unit of time.

8.2 Model Traceability

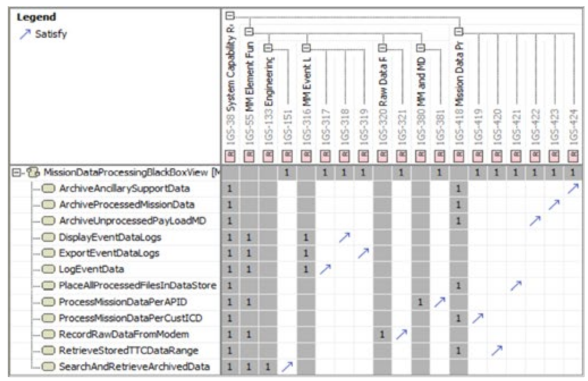
Measure Introduction	
Description	The usefulness and quality of a digital model depends on the completeness and integrity of the relationships among model elements. Traceability between elements, such as requirements allocation and flow down to architectural, design, and implementation components, provides assurance that the system solution is complete and consistent. Gaps in bi-directional traceability within a digital model can indicate where further analysis or refinement are needed. Traceability also supports impact assessments as a result of engineering changes. Traceability reports and analyses are greatly facilitated by modern digital modeling tools. The traceability concepts and indicators in this specification are representative examples of more general traceability mappings and reports across the development life cycle, such as: • Traceability between stakeholder needs, system requirements, and allocated or derived requirements at each level of the system hierarchy • Traceability and flow down of requirements to the logical or physical solution domain (e.g., design, implementation, integration, verification, validation) • Allocation and traceability of performance measures or parameters, such as Measures of Effectiveness (MOEs) or Key Performance Parameters (KPPs) • Traceability of system interfaces
Relevant Terminology	Model Element — Modeling constructs used to capture the structure, behavior, and relationships among system model components.

Information Need and Measure Description	
Information Need	What is the extent of achieved coverage across digital model elements? What is the extent of traceability from requirements down to the logical or physical solution domain? What traceability gaps or defects exist in the digital model?
Base Measure 1	Model Element 1 (source or parent element for traceability) *The base model element set from which traceability is derived or allocated,* *e.g., a stakeholder need or requirement.*
Base Measure 2	Model Element 2 (destination or child element for traceability) *The base model element set derived, mapped, or traceable to Model Element 1, linked using a model element relationship, such as a «deriveReqt» or «refine» relationship.* *e.g., a system requirement derived from and traceable to a stakeholder need.*
Derived Measure 1	Model element traceability [integer] • Total Model Elements = Model Elements Traced + Model Elements Not Traced]
Derived Measure 2	Model traceability (coverage) [real: percentage] • Percent Traced = ((Model Elements Traced) / (Total Model Elements)) * 100 • Percent Not Traced = ((Model Elements Not Traced) / (Total Model Elements)) * 100

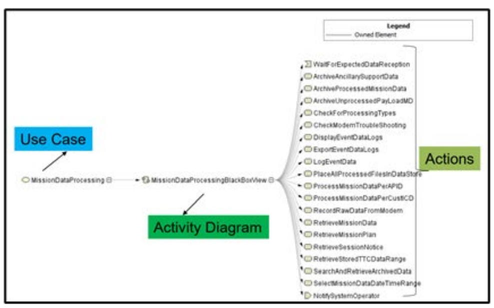
<table>
<tr><th colspan="2">Indicator Specification</th></tr>
<tr><td>Indicator Description and Sample</td><td>
Model Traceability can be depicted using visual or tabular summaries of the relationships among model elements. The specific indicators may depend on the model elements for which traceability is being measured, and the built-in reports and analyses provided by the digital modeling tool. Traceability among model elements might be implemented by showing requirements derivation and coverage of stakeholder needs into system and component requirements.
Representative example indicators used to assess traceability dependencies among selectable model elements (e.g., requirements, use cases, activities, logical architecture and design, physical design, interfaces, parameters, measures of performance) are depicted in Figure 8-3. Here, mostly 2-dimensional matrices containing model specific model elements of interest are utilized. Alternatively, the relationship between model elements might be depicted as flow down. With respect to Figure 8-3 (bottom left), a specific use case is linked to related actions via an activity diagram.



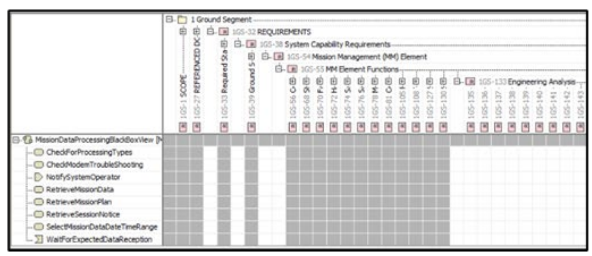
Traceability Between Model Elements (Dependency Matrix)



Relationships to Problem or Solution Domain («satisfy» or «refine» Matrix)



Relation Map Diagrams (Model Traceability, Ownership)



Identifying Model Traceability Gaps (Orphans)

<u>Excerpts from</u>: ‘MBSE and Requirements Analysis, Key to Successful System Engineering’, M. Osaisai and F. Markham, 2019 MBSE Cyber Experience Symposium. Used with permission from Macaulay Osaisai. All other rights reserved.

Figure 8-3: Example Traceability and Dependency Diagrams
Traceability and coverage (or lack thereof) can be quickly visualized, and gaps or defects addressed, e.g., systems that do not satisfy requirements or any unsatisfied requirements might indicate incomplete work or systems without required functions.
For further visual, tabular, and reporting capabilities and information, refer to the advanced topic discussion section below.
</td></tr>
</table>

Analysis Model	Review and analyze traceability dependencies among model elements to assess the completeness, adequacy, quality, and integrity of the digital model. The analysis may vary according to the types of specific model elements selected, but general guidelines may include: • Each source (parent) model element (Model Element 1) should be traceable to one or more allocated or derived destination (child) model elements (Model Element 2). • Each destination (child) model element (Model Element 2) should be derived from, or refine, a parent requirement or model element (Model Element 1). • Determine if the set of linked dependencies are, in aggregate, sufficient to adequately implement the parent requirement or model element.
Decision Criteria	In case an agreed-upon, specific coverage of stakeholder needs into system and component requirements has not been met, the team shall specifically address these gaps. To validate whether the system meets stakeholder needs, at minimum, the system requirements should be traceable to these stakeholder needs. Model elements that do not satisfy requirements, might be obsolete and shall be evaluated.

Additional Information	
Additional Analysis Guidance	Traceability can be useful indicators of model quality and modeling progress. Revisions to the model elements or relationships may be needed to close gaps. Derived measures of traceability for the selected model elements, such as Percent Traced and Percent Not Traced to assess the completeness and integrity of the digital model. Track progress in completing the traceability measures as the modeling effort matures.
Implementation Considerations	Traceability reports and analyses are typically available directly as built-in features of modeling tools. Traceability and analyses depend on the quality of relationships and dependencies established between modeling elements. Modeling conventions and guidelines should be established to assure the consistent use of model elements and relationships across the project. Failure to establish and enforce consistent modeling practices can impact model quality, and the integrity and usefulness of traceability measures. When stakeholders over-emphasize specific system requirements or physical implementations, then they should provide a rationale why these efforts are valid based on the needs-to-requirements flow down.

Additional Specification Information	
Information Category	Product Quality
Measurable Concept	Functional Correctness (Completeness)
Relevant Entities	Model components
Attributes	Level (e.g. system, requirements, design, component)
Data Collection Procedure	Counts of model elements and type are typically provided by modeling tools. Queries, scripts, or APIs may be available to automate the collection of model element count measures.
Data Analysis Procedure	Traceability reports (bi-directional linkages between selected parent and child modeling elements) are often generated directly from modeling tools. Review mappings between model elements for sufficient coverage. Generally, each parent must have one or more children, and every child must have at least one parent. Look for incorrect or disconnected traceability, such as orphans and barren requirements. Ensure adequate representations of associated modeling views and diagrams, e.g., use cases, sequence diagrams, activity diagrams.

Advanced Topic - Traceability for Complex Systems and Missions

Discussion

Beyond traceability of elements within a digital model as described above, traceability concepts can be scaled and expanded to consider higher level challenges, such as complex systems, compliance, and mission engineering. Addressing these challenges is enabled by digital transformation and advancements in sophisticated toolsets.

Complex systems can generally be decomposed into a hierarchical set of layers and a parallel set of legal frameworks to handle components and manage complexity. Distinct components and links between components within each layer are assigned Universally Unique Identifiers and attributes to enable 1 .. n relation traceability, inheritance, and assurance of data integrity across objects in the complex system. While the term Traceability Layer refers to general concept of the decomposing a system into vertical layers, an exemplary hierarchical set of Traceability Layers might include:

- Mission Layer (mission needs, mission threads, mission effect chain)
- Compliance and Strategy Traceability Layer (strategies, and legal or compliance constraints)
- Requirements Traceability Layer (decomposition of requirements and link relationships)
- Functional Allocation Traceability Layer (requirements allocation to domain functions)
- Component Traceability Layers (hierarchical physical and cyberphysical component hierarchy linked to functions or requirements)
- Sub-Component Traceability Layers (hierarchical physical and cyberphysical sub-component hierarchy linked to larger components)
- Elementary Components Traceability Layers (allow further decomposition, as needed)

Measures of traceability provide indicators of percentage coverage across layers (vertical, horizontal, requirements, mission needs, etc.).

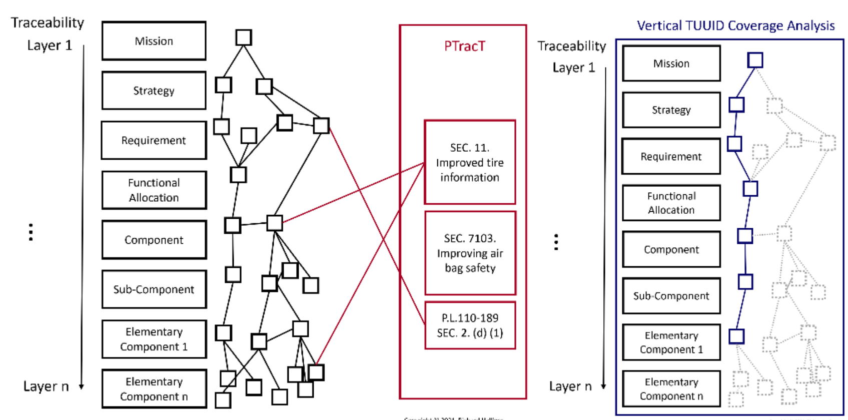


Figure 8-4: Traceability for Complex Systems and Missions[1]

Figure 8-4 illustrates of the concepts of decomposing complex systems into layers (left), the associated legal framework components run in parallel to these layers (middle), and traceability that runs through all layers to the top layer, e.g., Mission Layer (right). The red linkages illustrate the concept of horizontal traceability coverage. The second diagram illustrates traceability measures for vertical coverage.

1 Copyright 2021 by Richard Halliger. Reprinted with permission.

Description

The discussion above introduces high level concepts only. A more detailed description with measures and mathematical analyses is beyond the scope of this digital engineering measurement framework document. Further description and details are published in a white paper on the PSM website, https://www.psmsc.com/Prod_TechPapers.asp

8.3 PRODUCT SIZE

Measure Introduction	
Description	Many process measures for estimating and managing engineering product development depend upon a meaningful characterization of the scope or quantity of work to be performed. Often product size is used as a proxy for determining measures such as effort (hours), schedule (months), productivity (size/hour), or capability performance (product delivered / months). Proxies for product size in this context are commonly used in many engineering disciplines, such as unit or component counts, Lines of Code (LOC), number of drawings, or capabilities. No such established proxies, conventions, or models for size or productivity are commonly used yet for digital engineering in practice. However some measure of product size is needed, at both the project and enterprise levels, to normalize historical performance data, characterize prior work, relate it to estimating future work, and to quantify business improvement trends. A measure of product size is also needed to support the definition or evaluation of digital engineering measures defined elsewhere in this document. This draft Product Size measure is offered to initiate this discussion across the industry and to advance a needed conversation toward industry consensus. It is more theoretical than practical, since limited experiential data exists. It is fully expected this definition will evolve over time, with the advantage that encapsulating a size measure here in this specification enables reference and reuse by other measurement specs with reduced impact on rework as digital engineering practices mature across the industry. The current proposed definition of digital engineering product size is based on the concept of **model elements** generated as an outcome of the modeling process, as described in section 2.2.2. Organizations may establish conventions for the model elements to be counted and analyzed for sizing, based on their applications, methodology, tools, or domain. Examples include structural (static) model elements, behavioral (dynamic) model elements, organizational model elements (packages, libraries), or annotational (descriptive) model elements. Refer to section 2.2.2 for additional details.
Relevant Terminology	Model Element — See definition in section 3.

Information Need and Measure Description	
Information Need	What is the size and scope for the digital engineering project or product? How much work must be done? How does product size relate to estimates and measures of cost, schedule, productivity, or performance?
Base Measure 1	Product Size (Model Elements): model elements (planned and actual) [integer] *Organization or project-specific units and scope for what model elements are counted.*
Base Measure 2	Effort (Labor Hours): hours (planned and actual) [integer] *Estimated or actual effort in labor hours for the work to be designed or implemented.*
Base Measure 3	Duration (Calendar Months): months (planned and actual) [integer] *Length of the design or development effort (planned or actual) for the scope of work related to Product Size.*
Derived Measures	Product Size is the primary measure delivered through this initial draft specification. But several candidate measures could be derived from the base measures, such as these below. • Productivity = (Product Size) / (Effort) *Number of model elements generated per unit effort (e.g., model elements / labor hours)* • Progress = (Product Size$_{\text{actual to date}}$) / (Product Size$_{\text{planned}}$) *Percent of planned model elements completed to date for characterizing progress and work remaining. Can also be used to characterize growth and stability (actual size vs. planned).* • Throughput = (Product Size) / (Duration) *Number of model elements completed per calendar period, e.g., model elements / calendar month. Can also be used to characterize a size vs. schedule relationship.*

	Detailed description of these derived measures is beyond the scope of this Product Size measurement specification. It is expected these derived measures will be further defined by other specifications as necessary to satisfy related information needs.

<table>
<tr><th colspan="2">Indicator Specification</th></tr>
<tr><td>Indicator Description and Sample</td><td>
Indicators for Product Size will generally plot the size (number of model elements) over time and the relationship with effort and schedule, such as the example indicator concepts below.

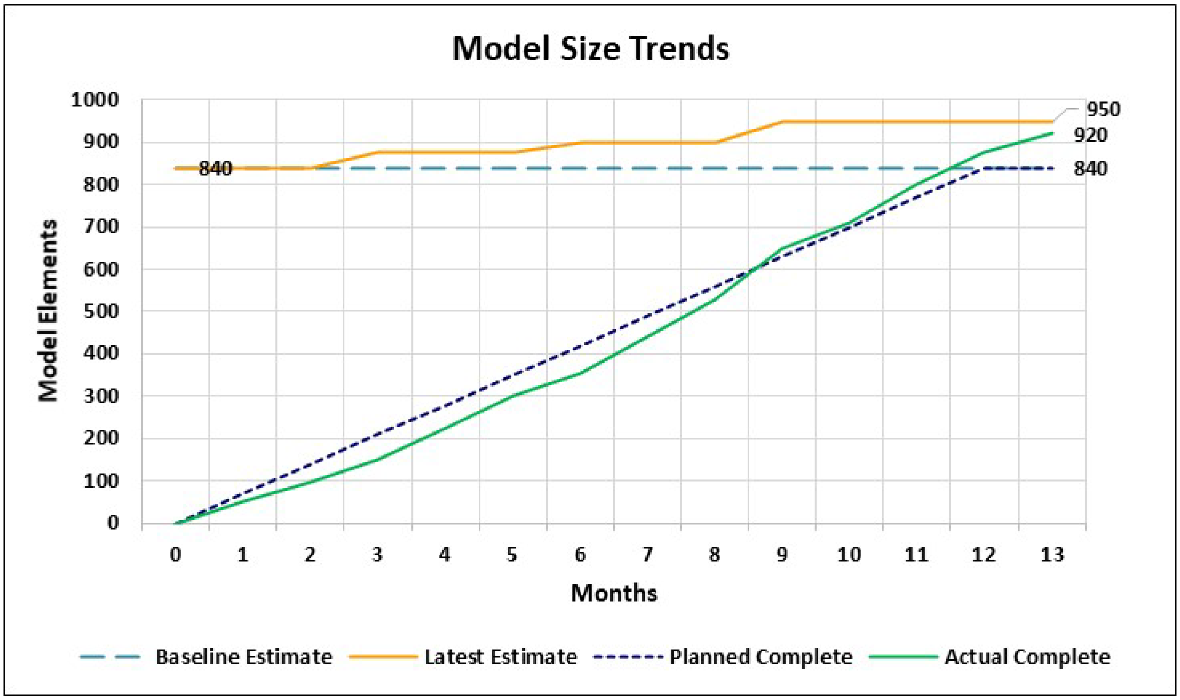
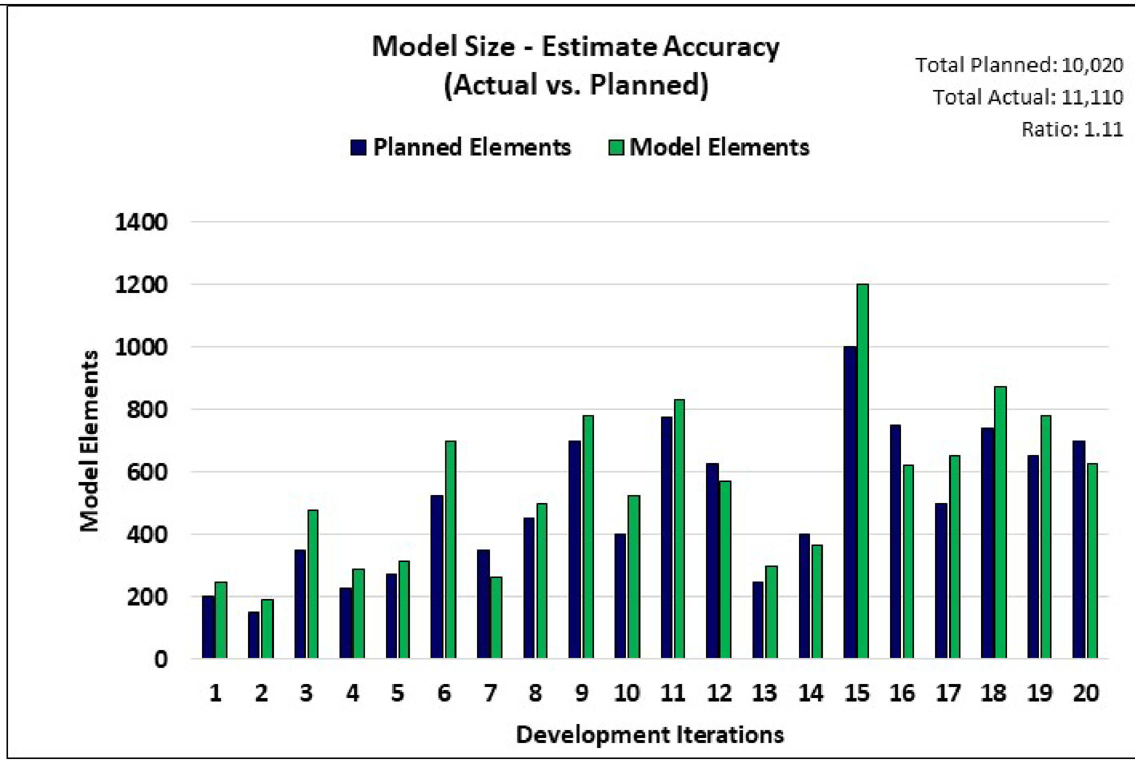
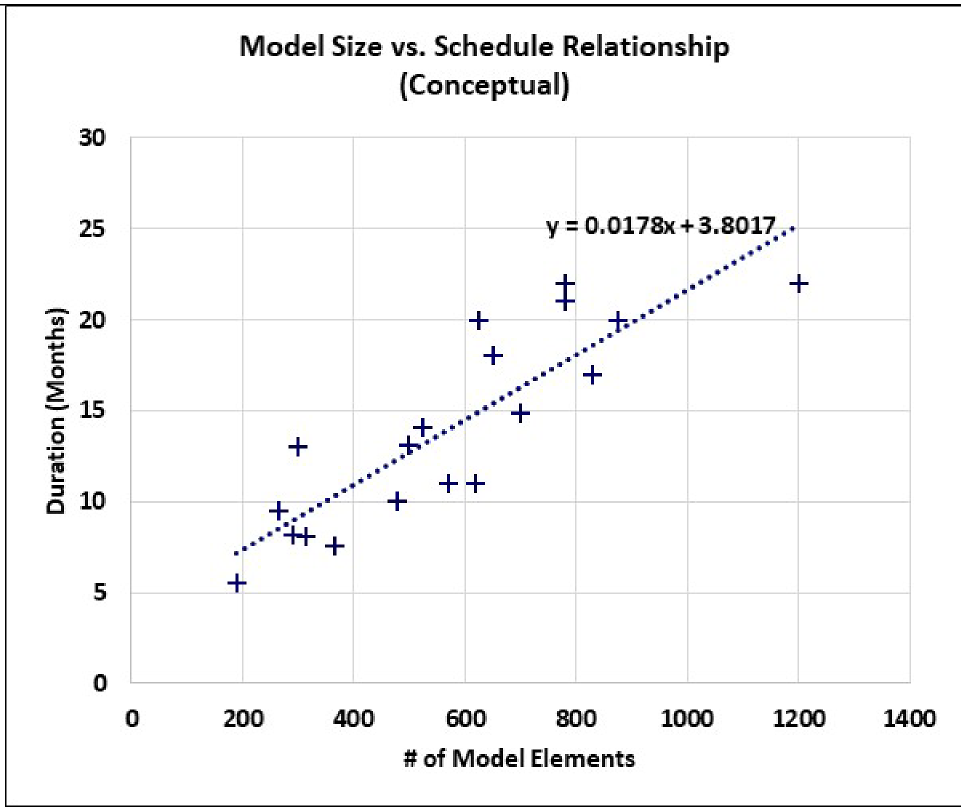


Figure 8-5: Model Size Trends

This indicator plots planned vs. actual product size (number of model elements) over time. The original baseline estimate (light blue dashed line) was 840 model elements over a 12 month schedule (70 elements/month). Initial estimates early in the project were not yet well understood as requirements and scope were being defined, and the total estimate was increased (orange solid line) to 875 elements in month 3, and 900 elements in month 6. Change requests were received from the acquirer following a design review, and the scope was increased to an estimated 950 elements in month 9. Planned and actual modeling development progress by month (cumulative model elements completed) are plotted in the blue dashed line and green solid line, respectively. These indicate that modeling progress was behind plan for the first 8 months, but recovered in month 9 as the team identified reuse opportunities across separate modeling efforts and became more productive in leveraging capabilities of the modeling tools. However, the total count of modeling elements overall was under-estimated so the original completion date of month 12 was delayed a month to complete the additional modeling effort. Upon completion, the actual count of final modeling elements (920) was 80 more than the original plan (840), but 30 less than the last estimate (950).
</td></tr>
</table>

Indicator Description and Sample (continued)	 **Figure 8-6: Model Size - Estimate Accuracy** This indicator depicts the accuracy of model size estimates (planned vs. actual number of model elements) for a sequence of iterations. The tendency has been to under-estimate the quantity of model elements needed (by 11% overall), which can lead to challenges meeting budget or schedule.

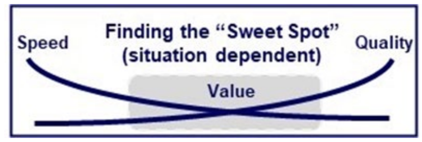
	 **Figure 8-7: Model Size versus Schedule Relationship** This indicator plots the size (number of model elements) of a set of products vs. their schedule duration. This example depicts a fairly consistent correlation between model size and schedule. A similar indicator might be plotted vs. effort (hours). This relationship might be used to validate schedule estimates for future development components based on their model size.

Analysis Model	Throughout the product development, compare the actual size of completed digital engineering products (count of modeling elements) versus plans and estimates, and consider the causes for deviations and if adjustments to plans are necessary. • Are model element counts for completed components consistent with engineering plans? • Are model components and elements being completed at a rate needed to meet progress, cost, and schedule? • What are the reasons for deviations in model element counts vs. the plan? Was work mis-estimated or misunderstood? Were there changes in the scope of work? Was work more complex than expected? • If actuals deviate significantly from estimates, do plans need to be adjusted for current or future work? Over time, a historical database of digital engineering modeling attributes (size, cost, effort, schedule) can be established across projects and used to inform estimates for future projects. For example: • Is the new project similar in size and scope as historical projects? • Can the actuals from those completed projects be used to develop or validate estimates on the current project? • Are projects becoming more productive? Are effort, cost, schedule, and efficiency improving?
Decision Criteria	Variances in model element counts exceeding defined thresholds (e.g., ± 10%) should dictate reconsidering the feasibility of current plans, estimates, or resources.

Additional Information	
Additional Analysis Guidance	As modeling efforts complete, track historical performance in size estimates vs. actuals for potential use in predicting future performance. Using a count of model elements as an indicator of model size and effort is needed - similar to using Lines of Code (LOC) as a proxy for software effort - with many of the same benefits and liabilities that LOC incurs. Refer to well-established analysis guidance for other traditional size-based measures (such as LOC and other measures described in the PSM guide *Practical Software and Systems Measurement: A Foundation for Objective Project Management.*) Note that counts of modeling elements are likely to be specific to a given product or project and not directly comparable across projects due to varying modeling conventions, counting rules, estimating standards, or modeling tools. It is also likely that other attributes, such as domain or complexity, will factor into these relationships. Use of model size measures at the enterprise level is therefore likely problematic until modeling conventions, counting rules, and standards are applied consistently across projects. Size is just a means to an end. Accurate size counts are not the primary objective of this measure. Size is a proxy for the amount of work to be performed, and to enable achieving the more important attributes of accurate estimates and feasible plans (e.g., effort, schedule, cost). Product size is a basis for many other indicators, such as productivity, rework, cost and schedule estimating relationships, and other derived measures.
Implementation Considerations	A count of model elements can typically be obtained from project modeling tools.

Additional Specification Information	
Information Category	Size and Stability
Measurable Concept	Functional Size and Stability
Relevant Entities	Model components
Attributes	Model element type (e.g., requirements, design, architecture)
Data Collection Procedure	Counts of model elements and type are typically provided by modeling tools. Queries, scripts, or APIs may be available to automate the collection of model element count measures.
Data Analysis Procedure	Regularly analyze stability of model size and growth trends against plans and decision criteria (weekly, monthly), taking corrective action as needed to bring plans back into alignment.

8.4 Defect Detection

Measure Introduction	
Description	Figure 8-8: Speed - Quality Tradeoffs  Programs strive to deliver products of acceptable quality for use by internal or external users, 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, value, and time to market. Defects are typically collected, analyzed, and monitored across lifecycle process activities or boundaries, such as stages, phases, iterations, or releases. In this specification the term iteration will most often be used to describe concepts, but could be interpreted and applied in other contexts tied to the project lifecycle model. 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, modeling, simulation, automated testing throughout development, and other verification and testing approaches.
Relevant Terminology	Defect — See definition in section 3, Terms and Definitions Contained Defects (Saves) — Defects detected and resolved before internal or external release of the iteration containing the defect. Escaped Defects — Defects detected or resolved after internal or external release of the iteration containing the defect. Defects are generally tracked separately for internal and external releases.

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 user (e.g., Integration and Test, Formal Test) or released (escaped) to an external user? 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?
Base Measure 1	Contained Defects [integer] *Defects detected and resolved before internal or external release of the iteration containing the defect.*
Base Measure 2	Internally Escaped Defects [integer] *Defects that escape across development iterations but are detected and resolved prior to internal product baseline releases*
Base Measure 3	Externally Escaped Defects [integer] *Defects that escape from development iterations and are not resolved until deployed to external operations*
Derived Measure 1	Total Defects = Contained Defects + (Internally Escaped Defects + Externally Escaped Defects) [integer] *Total defects detected after initial development, whether contained or escaped across iterations and releases*
Derived Measure 2	Internal Defect Escape Percentage = (Internally Escaped Defects / Total Defects) * 100 [percentage] *The percentage of total defects that escape across internal iterations but are resolved prior to release of products to operations*
Derived Measure 3	External Defect Escape Percentage = (Externally Escaped Defects / Total Defects) * 100 [percentage] *The percentage of total defects that escape from internal iterations and are not resolved until deployed to operations*

Derived Measure 4	Total Defect Escape Percentage = (Internally Escaped Defects + Externally Escaped Defects) / Total Defects * 100 [percentage] *The total percentage of defects detected that escape from development iterations.*

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 8-9 all defects detected before the release (during development iterations , 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 8-9 depicts the Contained and Escaped Defects for each iteration and internal/external release along with the corresponding 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 depicts the effectiveness of defect detection processes and verification activities performed during development prior to release.

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

Figure 8-9: Defect Detection by Iteration/Release

In the example above, Iteration 1.0 had a ratio of 20% of total escaped defects, with 5% of recorded defects detected after release to the external user. This gradually improved over time to a ratio of 5% on Release 3.0. This was due to a more stable set of requirements, better models, 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 user.

Defect containment measures are the complementary and equivalent inverse of defect escape measures, and are preferred by some organizations to characterize the effectiveness of their internal quality and defect removal processes. In the example above a Total Escape Ratio of 11% is equivalent to a defect containment effectiveness ratio of 89%.

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 user, they would be identified in relationship to iterations. Defects detected in the iteration in which they were injected (originated) are contained, and those detected in later iterations are escaped. Defect counts could be shown in a table as in Table 8-1 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 or phase after they were inserted). See the PSM core framework for more information on Defect Containment.

Table 8-1: Defect Containment

Defect Containment

Defects			Defect Resolved (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 iteration. 41% of defects took at least two iterations to detect, which indicates that the assessment of the work in that phase needs to be improved, possibly with better review processes, modeling, simulation, or 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 iteration and whether the team is improving over time. The Defect Escape Ratio should typically 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 review processes, modeling, simulation, or 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	Defects could be separated by severity, priority, or other attributes. This measure may be used in conjunction with other quality measures including the Defect Resolution, and Adaptability 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 user. In addition, in a model-based process, defects should be assessed for the internal development team iterations. 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 regular intervals (weekly, monthly), at major milestones, and at the end of each major release to determine status and progress over time.

8.5 DEFECT RESOLUTION

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. At the system level this measure is particularly concerned with the concept of defect containment - that defects are discovered and resolved within a iteration. Refer to Figure 1 of Defect Detection for relevant defect terminology. DE is generally concerned with the internal activities of a development team using data and models to resolve errors and defects before effort is committed to internal or external releases.
Relevant Terminology	Defect: See definition in section 3, Terms and Definitions Contained Defects (Saves): Defects detected and resolved before internal or external release of the iteration containing the defect. Escaped Defects: Defects detected or resolved after internal or external release of the iteration containing the defect. Defects are generally tracked separately for internal and external releases. Projects are organized into life-cycle activities characterized as development iterations, product iterations, and releases. Often programs use a gate review process to determine expectations or suitability for the next release.

Information Need and Measure Description	
Information Need	Are we finding and removing defects early in the lifecycle? Are we finding and removing defects prior to operations? How many contained defects in the requirements, architecture or design phases would have affected the operational product? Are we containing defects in early phases using models and shared information?
Base Measure 1	Defects Detected - defects found per iteration [integer]
Base Measure 2	Defects Resolved - defects resolved per iteration [integer]
Base Measure 3	Iterations to Resolve - number of iterations between detection and resolution [integer]
Derived Measure 1	Defect Backlog = (Defects Detected - Defects Resolved) - number of defects not completed (by iteration, or cumulative) and still pending in the product backlog for resolution [integer]

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

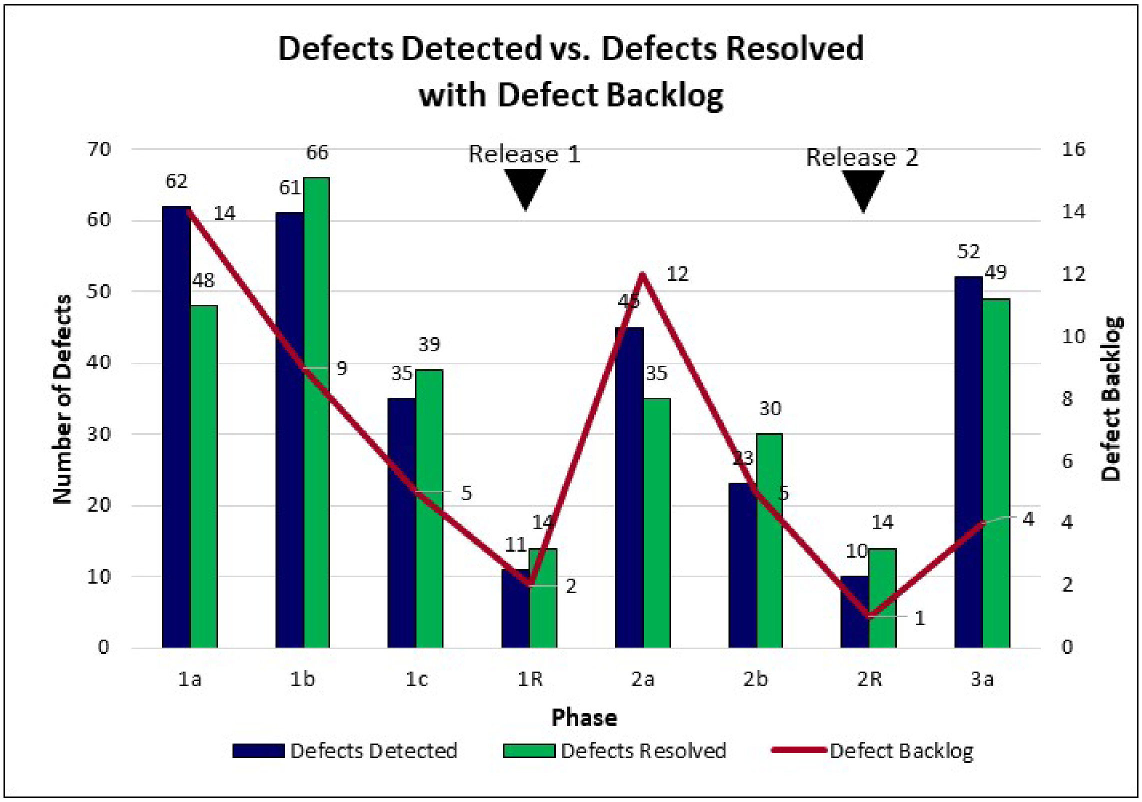


Figure 8-10: Defects Detected versus Resolved - by Iteration with Backlog

Figure 8-10 shows an example plot of the defects detected and defects resolved base measures calculated over a set of life-cycle iterations, which could be development iterations (1a, 1b, 1c) or releases (1R). Note that for iteration 1a, not all the defects discovered in iteration 1a were resolved in that iteration. The 14 unresolved defects were then deferred, and rolled over into the product backlog (green dashed line plotted on the right axis). Backlog defects are prioritized and planned to be resolved in upcoming iterations. For iterations 1b and 1c, more defects were resolved than detected, meaning that defects discovered from previous iterations were resolved, thus reducing the defect backlog. Figure 1 also shows the concept of development iterations leading to internal or external releases. Defects should be resolved prior to release if possible. The dashed backlog line depicts the reduction in backlog depth at the release milestones.

Usually, a project features a ramp up period, e.g., due to capability integration, and a resolve period including defection resolution efforts. In the best case, the backlog of defects is monotonically decreasing (defect burndown) in addition to a steep decrease of new defects detected. These two quality factors are typical indicators of release readiness. In general, DE activities should shift the defect detection and resolution activities to earlier in the overall project lifecycle, moving the peak of the curve to the left relative to traditional methods.
</td></tr>
<tr><td></td><td></td></tr>
</table>

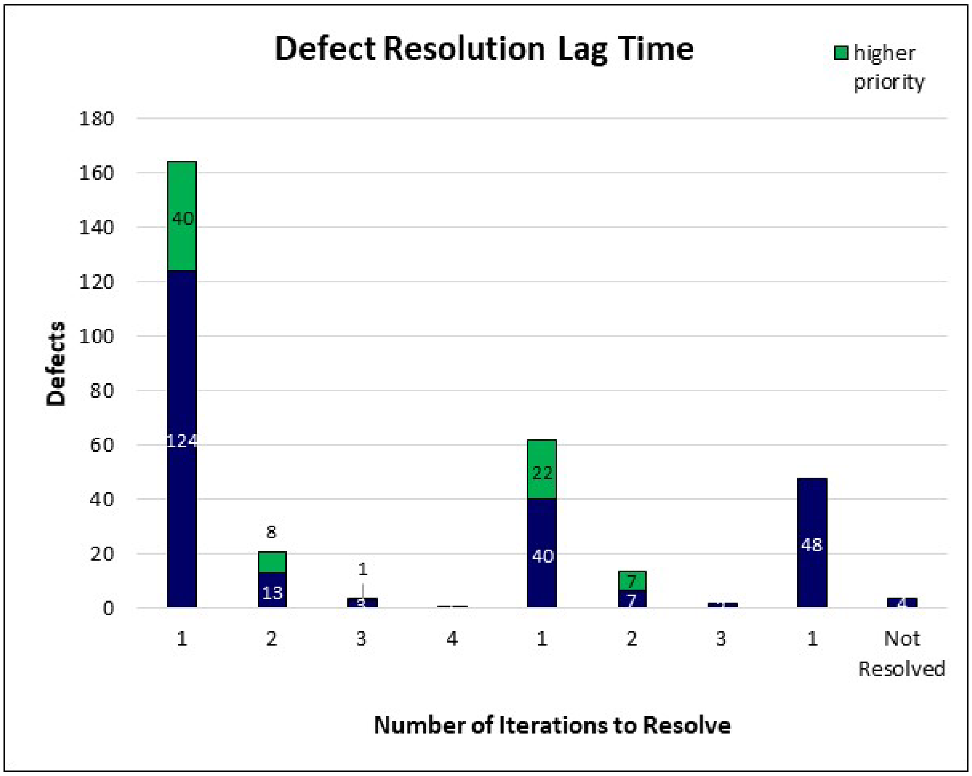


Figure 8-11: Defect Resolution Lag Time

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 8-11 as Defect Resolution Lag Time. In this example, the defects that took 3-4 phases to fix were lower priority defects dealing with minor changes to screen displays and software documentation.

Defect Containment

			Defect Resolved (Iteration)								
Defect Detected (by iteration/release)			1a	1b	1c	1R	2a	2b	2R	3a	Not Resolved
	1a	62	48	11	2	1					
	1b	61		55	6						
	1c	35			31	3	1				
	1R	11				10	1				
	2a	45					33	10	2		
	2b	23						20	3		
	2R	10							9	1	
	3a	52								48	4
	Total	**299**	48	66	39	14	35	30	14	49	4

Escapes

Saves

>1 Iteration	2%
1 Iteration	12%
Same Iteration	75%

Notes

Defects are planned to be resolved in the same phase they are detected.

The goal is to detect the defect within the same iteration as they orginate in or the next iteration.

The Threshold is detecting defects more than 1 phase after they originate.

Figure 8-12: Defect Containment

Ideally, a defect would be resolved in the same iteration as it was discovered (saves - the green series of diagonal cells in the figure above). All cells to the right of this diagonal represent escaped defects across iterations. In this example, the project is timely in resolving detected defects; 85% of identified defects are resolved in the same iteration, and 97% within one iteration.

Filtering can be applied for the most critical or highest priority defects. Defects that are not resolved or planned to be 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. With respect to Figure 8-12, large-scale projects might feature high-priority detected defects that are not resolved over multiple iterations due to system complexity.

Analysis Model

Defects Detected vs. Defects Resolved, with Net Defect Backlog:

- For each iteration, compare the quantity of defects resolved vs. defects found (ref. Defect Detection specification). Timely closure of high priority defects is a key measure, critical defects should typically be resolved in the same iteraction they are detected. What are the reasons defects were not resolved in the same iteration? Are resolutions intentionally deferred to future iterations? If so, why? Resolution of lower priority defects may be deferred in order to prioritize resources toward other higher priority work.
- Are defects in the product backlog being worked off at an acceptable rate? Is the defect backlog being managed? Defects in the backlog should be trending downward toward upcoming releases. If the defect backlog continues to grow, it may be necessary to add more resources to defect resolution, or focus a future iteration on closure of defects.

Defect Resolution Lag Time:

- What percentage of defects detected are being removed in the same iteration (Saves)? This correlates to the green diagonal. The resolution rate should align with process and quality performance objectives. For defects resolved in future iterations, are the escapes typically limited to only one iteration?

Decision Criteria	When the difference/gap between cumulative defects detected 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. Defects with Priority 1 and 2 should 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 acquirer approval of this action. Priority 3 through 5 defects may be deferred until later iterations, based on acquirer priorities. 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 acquirer agreement. Priority 3 through 5 defects not resolved may be released with a work-around approved by the acquirer.

Additional Information	
Additional Analysis Guidance	Relative to traditional development, DE/MBE activities should result in a shift of defect detection and resolution to earlier iterations of development and particularly should reduce the number of unresolved defects in a release iteration. At this early point in DE measurement, programs have seen around 20% reduction in release defects using DE processes from earlier experience. Programs should evaluate their use of data, models, and DE/MBE processes with respect to improvements over time in the quality (# defects) of releases. DE/MBE should result in early verification and product specification completeness in earlier life-cycle iterations accomplished via models and digital system views. A particular emphasis in this measure is determining if defects are both detected and resolved in earlier iterations than in traditional development, and that defects are resolved as early as possible. Programs may want to pay particular attention to the number of defects discovered and resolved early when implementing model-based versus document-based reviews and approval processes. Additional measures related to defects detected specifically in review processes may be of interest in the movement from document to model-based reviews. Defect resolution measures may be most crucial for high priority defects (e.g., severity levels 1-2), or other defect attributes such as specific defect categories or model element types. Additional selection filters may be applied to reduce the defect data set to the areas of particular interest. Root cause analysis should be performed in the event of performance or quality anomalies. As defects are detected and disposed, the team will note a defect that is significant. This will provoke an effort to determine when it was created and why it survived for any length of time. Similarly, if a particular category is having a number of defects to attract attention, a similar effort to determine the process gaps that are permitting the defects to be created.

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)
	• Classification of internal defects vs. external defects (e.g., defects discovered by the supplier, by the acquirer in an operationally representative environment, or by the acquirer in operations)
	
	The tools used for modeling and defect recording should support a means to collect the iterations or releases where defects were detected and resolved.
	
	Some iterations and releases may be planned to target only defect 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 or release where the defects were detected and resolved Other defect attributes as applicable (e.g., priority, categorization)
Data Collection Procedure	Data is collected in modeling and defect tracking tools during each iteration or phase. The defect records should be tagged in a timely fashion with the corresponding iteration or release and other relevant information.
Data Analysis Procedure	Iterations in which defects are detected and resolved are discussed during the defect tracking and defect resolution meetings. Data is analyzed periodically (e.g., weekly, monthly) and at the end of each iteration. Defect detection and resolution data is presented and used as a criterion for iteration completeness at iteration gates, release readiness, and associated reviews.

8.6 ADAPTABILITY AND REWORK

Measure Introduction

Description

Relative to traditional methods, model-driven approaches can enable greater resilience and adaptability to changes or maintenance of engineering work products (e.g., requirements, architecture, design, integration, testing) when they occur. Following an initial up-front investment, models and traceability to work products can be leveraged to reduce time and effort for implementation, maintenance, defect correction, and other modifications or rework. This is represented conceptually in the life cycle cost diagram shown at right[1].

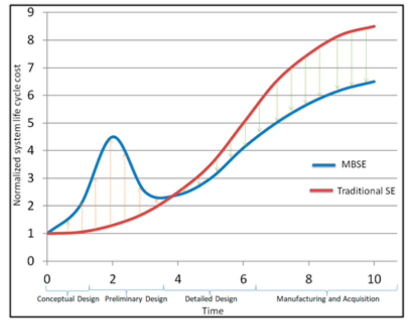


Figure 8-13: Economic Analysis of Model-Based Systems Engineering

Mahni and Purhhi, 2019

Change types can include:

- **Corrective actions**: repair or mitigation of system anomalies or defects that risk or prevent the work product from meeting its intended planned purpose
- **Perfective actions**: planned and scheduled enhancements to system products or services to implement improvements as a result of changes to requirements or user mission needs
- **Adaptive actions**: adapting system configurations to support other applications, systems, environments, or iterative refinements.

Traditionally, rework measures are focused on the effort to implement corrective actions for repair of defects. Here we envision the broader use of rework measures enabled through digital engineering to include change management, adaptability, and impact assessment contexts beyond simply the correction of defects.

In a digital engineering environment products are model-driven, providing additional opportunities to cost-effectively incorporate changes to digital models that are directly traceable to the implemented and tested work products, some of which can be automatically generated. Digitally engineered work products can therefore be more resilient to changes of all types described above with reduced rework effort for work products and model elements, whether planned (intentional, perfective or adaptive changes) or unplanned (correction of defects). Rework is typically measured in terms of the effort or schedule needed to implement the change action. These concepts for efficient model-based adaptation and rework are illustrated in Figure 8-14below.

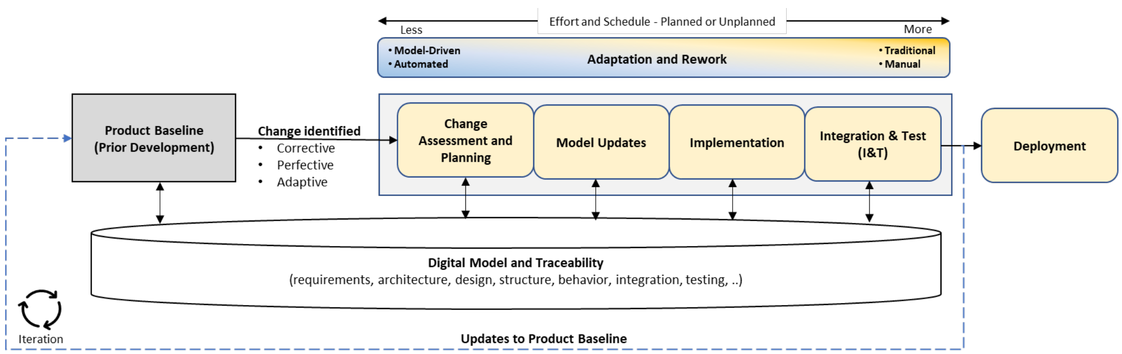


Figure 8-14: Digital Engineering Rework

These concepts align with the SERC causal analysis described in Table 1-1, including measurable benefits such as ease to make changes, customize designs, and reduce rework effort. Although limited quantitative data

[1] Azad M. Madni and Shatad Purohit, University of Southern California. 20 February 2019. MDPI Systems 2019, © by the authors. Licensee MDPI, Basel, Switzerland. http://www.incosewiki.info/Model_Based_Systems_Engineering/Files/d/d8/Madni_Purohit_2019_Economic_Analysis_of_Model-Based_Systems_Engineering.pdf. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/)

	of rework measures for model-based development currently exists, SERC research of industry literature cites reduction in defects, rework effort, rework cycles, percent rework, and technical debt as expected benefits. [2]
Relevant Terminology	<table><tr><th>Term</th><th>Synonyms</th><th>Description</th></tr><tr><td>Defect</td><td>Anomalies, Errors, Faults, Issues</td><td>A defect is a condition in a product (e.g. software, system, hardware, documentation) that does not meet its requirements or end-user expectation, causes it to malfunction or to produce incorrect/unexpected results, causes it to behave in unintended ways, or leads to quality, cost, schedule, or performance shortfalls. Defects may be documented in problem reports (or trouble tickets), or they may be added to the backlog for consideration in future iterations. [PSM CID Measurement Framework]</td></tr><tr><td>Rework</td><td></td><td>Action taken to bring a defective or nonconforming component into compliance with requirements or specifications. [IEEE SEVOCAB, PMBoK] The effort or schedule needed to implement changes to digitally engineered work products, including corrective, perfective, and adaptive change actions. [PSM Digital Engineering Measurement Framework]</td></tr></table>

Information Need and Measure Description	
Information Need	How much rework effort is spent maintaining planned or unplanned changes to digital engineering work products across the life cycle? [Project] Can changes to engineering work products be implemented more easily and with less effort in a digital engineering environment relative to traditional methods? [Enterprise]
Base Measure 1	Changes: Number of change requests to baselined work products, by change type [integer] *(see Attributes for other change request or defect characteristics useful for analysis or filtering)*
Base Measure 2	Model Elements Changed: quantity of model elements affected by the change request [integer] *(refer to Product Size measurement specification)*
Base Measure 3	Rework Effort: labor effort expended to implement a change request [integer]. Units: hours or equivalent.
Base Measure 4	Rework Cost: cost of rework expended to implement a change request (labor, material) [currency, e.g., dollars]
Derived Measure 1	Cumulative Changes: = Σ (Changes) [integer] *(total number of changes for the selected data set, filtered by change type and attribute)*
Derived Measure 2	Cumulative Rework Effort = Σ (Rework Effort) [integer] *Sum of rework effort for the selected change record data set.*
Derived Measure 3	Statistical analyses of rework correlated with selected change record attributes. *Examples: mean, median, variance, standard deviation, quartiles, correlations, outliers.*

Indicator Specification	
Indicator Description and Sample	Rework measures from traditional approaches (e.g., rework by stage or activity, percent of rework, Cost of Poor Quality) can be adapted and applied to digital engineering contexts and compared to legacy measures to assess measurable model-driven benefits. Rework analyses are often conducted across a set of many change requests, perhaps in affinity groupings or filters selected by product component, change request type, priority, or other parameters (see Attributes for additional examples). Indicators such as histograms, scatter diagrams,

[2] Systems Engineering Research Center. Summary Report, Task Order WRT-1001: Digital Engineering Metrics. SERC-2020-SR-003, June 2020. https://sercuarc.org/wp-content/uploads/2020/06/SERC-SR-2020-003-DE-Metrics-Summary-Report-6-2020.pdf

control charts, box charts or other indicator types can be used to collect and analyze a set of changes by attributes such as effort (e.g., hours), resources (e.g., full-time equivalent (FTE) staff allocated), cost ($), or schedule impact (hours, days, weeks). Such data for MBSE or DE rework is not yet consistently available in practice, so the indicators below are conceptual examples with artificial data for illustration only.

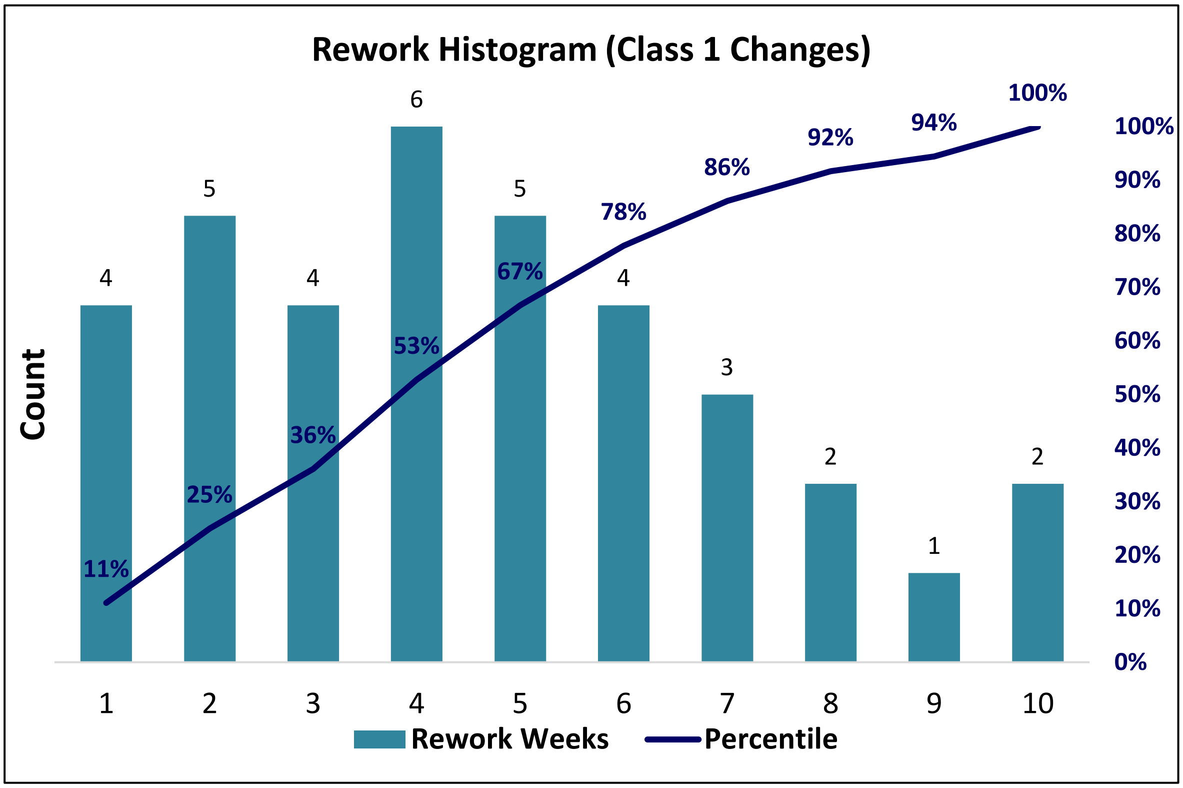


Figure 8-15: Rework

In this example, rework for Class 1 changes (planned modifications, or unplanned defects) is analyzed in a histogram by weeks of schedule duration to implement and test the change, including updates to associated models, work products, documentation, reviews, verification, and regression testing of changes. Twenty-five percent of Class 1 changes were completed within 2 weeks; over half within 6 weeks. This is a top level summary for a rework analysis; based on the distribution of rework and comparison against project plans/objectives, additional deep dive analyses may be needed to identify root causes and areas for further investigation or corrective action.

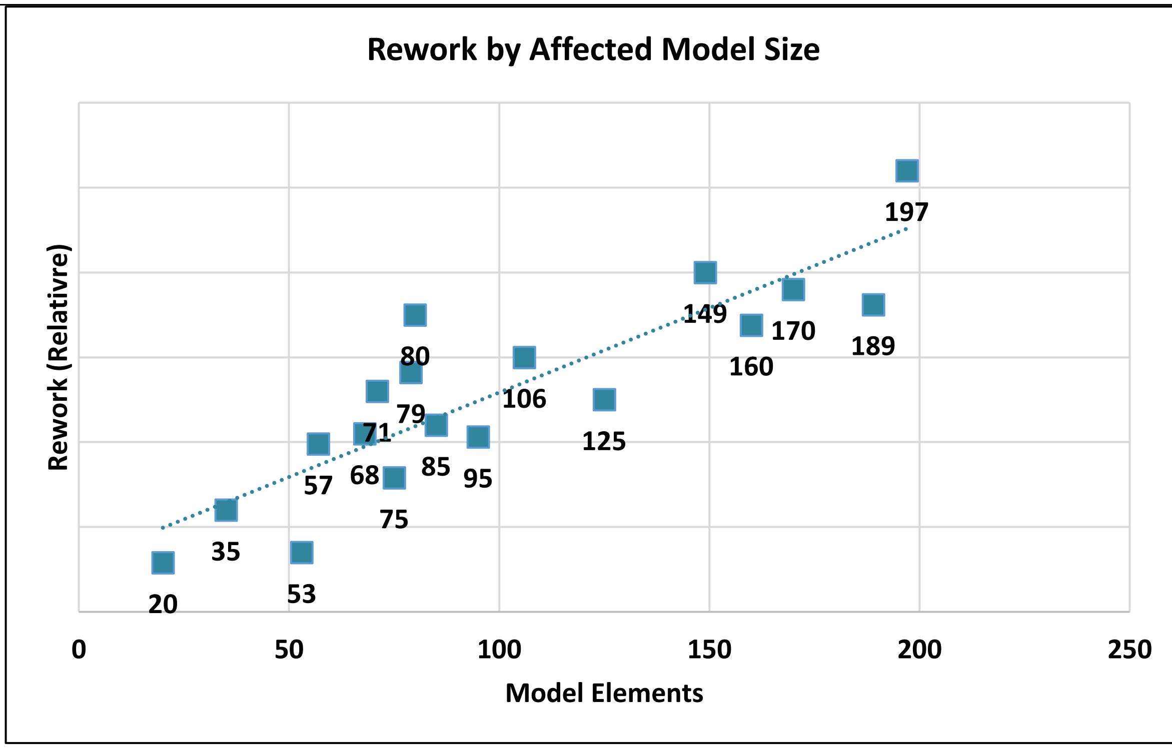


Figure 8-16: Rework by Affected Model Size

Individual changes can also be plotted and rework effort correlated with other factors or attributes. In this example, a scatter diagram is used to plot rework effort vs. the size of the change, specified in terms of product size (affected model elements). The vertical axis for the scale of rework effort has been intentionally excluded in this conceptual example since no actual model size vs. rework data is available to the authors, and to depict an actual size vs. rework effort relationship would be inaccurate and misleading. Conceptually, this type of analysis indicator might be used with actual data in the future to determine estimating relationships for rework vs. change size, or to further investigate anomalies outside expectations.

Indicator Description and Sample (cont.)

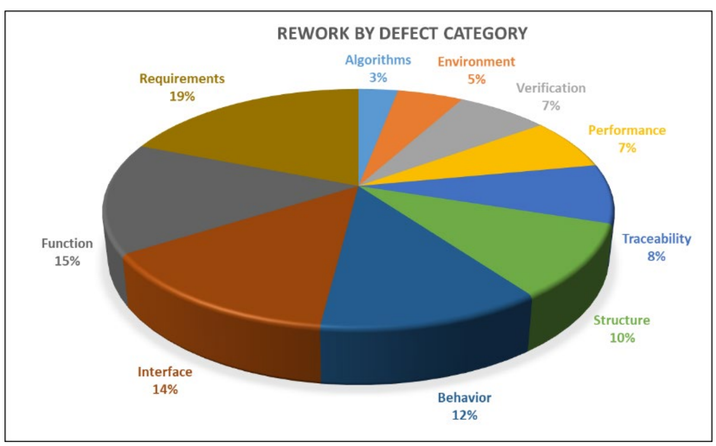


Figure 8-17: Rework by Defect Category

Capturing defects and rework by defect severity and defect category can help prioritize where to best invest effort in corrective or perfective improvement actions. In this example the project periodically analyzes defect records to identify areas with the greatest improvement impact. Analysis of defect data indicates the greatest proportion of rework relative to digital engineering is in the modeling of requirements, functional models, behavioral models, and interfaces. The project is implementing new project processes for model reviews and validation to improve model quality and reduce modeling defects in future iterations.

Analysis Model	Analyze measured change effort (rework – planned or unplanned) for the selected data set and filtered attributes. Look for correlations, trends, and indicators that can be used to investigate rework anomalies, systemic issues, root causes for improvement actions, or develop models that can be used to manage future performance and rework. Example analyses include: • Rework trends: Is the amount of rework appropriate to the size of the change effort and change type? Is the normalized relative rework increasing or decreasing? Is rework within expected bounds? • Rework distribution: Plot the distribution of rework by work product, activity, or life cycle stage. Is the distribution of rework effort as expected or are there areas that need further analysis? Is the model-driven approach leading to less rework relative to traditional development, with rework effort shifting from later to earlier life cycle activities as anticipated? • Rework categories: Analyze defect records by category to determine the areas of greatest project impact. Which defect types are most prevalent? Which are the most critical? Which cause the most rework? Determine the highest priority areas for improvement. Conduct root analysis at lower levels if needed to determine improvement actions.
Decision Criteria	Establish data set thresholds, performance targets, and tolerances for the range of expected rework based on change type and selected attributes. Measures of rework outside expected performance should trigger further investigation. Assess rework measures and trends against project plans (cost, schedule) and determine if adjustments to the plan are needed.

Additional Information	
Additional Analysis Guidance	Evaluate defect rework in conjunction with other defect measures. Other relevant and complementary measures from this PSM digital engineering framework include: • Defect Detection • Defect Resolution • Functional Architecture Completeness and Volatility • Model Traceability • Product Automation Refer to these corresponding measurement specifications for additional analysis guidance. Projects or organizations with a robust collection of historical data from past projects may be able to analyze the measurable benefits of model-based development relative to prior traditional development projects.
Implementation Considerations	Business systems, models, and tools must be configured and instrumented to collect the measures needed specific to the change effort, as tailored from this specification.

Additional Specification Information	
Information Category	Product Quality
Measurable Concept	Functional Correctness
Relevant Entities	Approved change requests, by change type
Attributes	Rework measures may vary according to the work product being modified and stage in the product life cycle. Example attributes that can be used to guide the rework analysis may include: • Product type (e.g., requirements, design, implemented work product, test) • Change type (corrective, perfective, adaptive) • Program activity or phase • Change reason (e.g., requirements change, defect) • Change request priority or severity

Data Collection Procedure	Collect change requests approved for work from a baseline change management repository or tool. Collect defects and associated attributes from the project configuration management repository or defect management tool. Collect the size of the change effort (count of affected model elements) from the project modeling tool. Collect labor measures from the project time tracking system. Labor should be tagged, categorized, or otherwise retrievable specific to the scope of the change effort. Collect cost measures from the project financial accounting system. Cost should be tagged, categorized, or otherwise retrievable specific to the scope of the change effort.
Data Analysis Procedure	Analyze aggregate rework measures and trends at regular intervals, such as monthly or quarterly, or in response to observed performance anomalies.

8.7 PRODUCT AUTOMATION

Measure Introduction	
Description	Model-driven development provides opportunities to automate engineering processes and generation of work products that have often been done manually in traditional approaches. Model-based work products such as requirements, architecture, design, use cases and other views or modeling artifacts can be automatically generated and published directly from modeling tools, at significant savings in effort relative to traditional documentation-centric approaches. Model-driven automation based on an Authoritative Source of Truth (ASOT) can lead to process efficiencies, labor reductions, shorter cycle times, less rework, and earlier verification and validation of solutions. Artifacts applicable for automation may vary based on many factors, including product, requirements, domain, availability of reference models, processes, resources, tools, and business constraints. It may not be practical for projects or enterprises to expect that all artifacts are model-generated. Projects or enterprises may set objectives for the quantity or percentage of engineering products that are automatically generated in a model-centric approach. Examples of potential model-driven measures of digital engineering product automation include: • % of digital model artifacts produced via automation • % of requirements verified through automation of digital model parameters and constraints • % of labor hours spent generating digital artifacts through automated vs. manual methods The industry sees automation of digital artifacts as one of the most significant expected benefits from a digital engineering implementation, so this specification is currently focused on measuring the actual artifacts only, with the goal of inspiring progress toward a widespread practice of model-based automated artifacts and reviews. As of this writing, the authors are not familiar with representative studies substantiating consistent savings in labor, cost, rework, or reviews realized through digital model-driven vs. documentation-driven approaches. It can be difficult to perform direct comparisons since systems vary widely, and it is likely some proportion of both approaches will continue be common on projects for some time. As the industry is still generally in the early stages of digital transformation with little historical data, it is also not clear that projects can accurately estimate the quantity of artifacts needed to compare plans vs. actuals in a digital model-driven environment.
Relevant Terminology	Model-driven automated artifacts: Products or artifacts produced and reviewed directly from digital models without significant manual intervention or generation of separate documents for development and review. Examples: model-based views and diagrams for requirements, architecture, design.

Information Need and Measure Description	
Information Need	What percentage of artifacts are automatically generated from digital models? To what extent are artifacts facilitating program reviews? How much is automation contributing to meeting our performance and quality objectives?
Base Measure 1	Total Artifacts *[integer > 0]* *Total count of artifacts generated using both automated and manual methods.*
Base Measure 2	Automated Artifacts *[integer ≥ 0]* *Count of artifacts generated from automated model-driven methods.*
Base Measure 3	Manual Artifacts *[integer ≥ 0]* *Count of artifacts generated using manual (non- model driven) methods, e.g., documentation generation.*
Base Measure 4	Known Artifacts Not Yet Addressed *[integer ≥ 0]* *Count of artifacts known to be necessary, but not yet generated using either automated or manual methods.*

Derived Measure 1	% of Automated Artifacts = ((Automated Artifacts) / (Total Artifacts)) * 100 *[percentage ≥ 0%]*
Derived Measure 2	% of Manual Artifacts = ((Manual Artifacts) / (Total Artifacts)) * 100 *[percentage ≥ 0%]*
Derived Measure 3	% of Known Artifacts Not Yet Addressed = ((Known Artifacts Not Yet Addressed) / (Total Artifacts)) * 100 *[percentage ≥ 0]*

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

(Product Model-Driven Artifact Automation)</td><td>
Figure 8-18 depicts the percentage of project artifacts that are generated or verified by automated vs. manual methods. 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, as the absolute quantities of artifacts generated is likely to vary widely. The total number of artifacts changes over time as digital modeling matures across engineering requirements, design, or verification stages. The term “artifact” is used as a proxy for the quantity or size of work products or model elements plotted on the vertical axes for the system or discipline of interest, e.g., requirements, design, use cases, test cases. The total quantity of artifacts is plotted on the secondary axis for context to enable consideration of the scale and complexity of the development and automation effort. Tradeoff decisions can be made on the benefit of investing further program effort to develop new digital modeling automation tasks to increase coverage. This may include estimating the net impact on program throughput, quality, or cost.

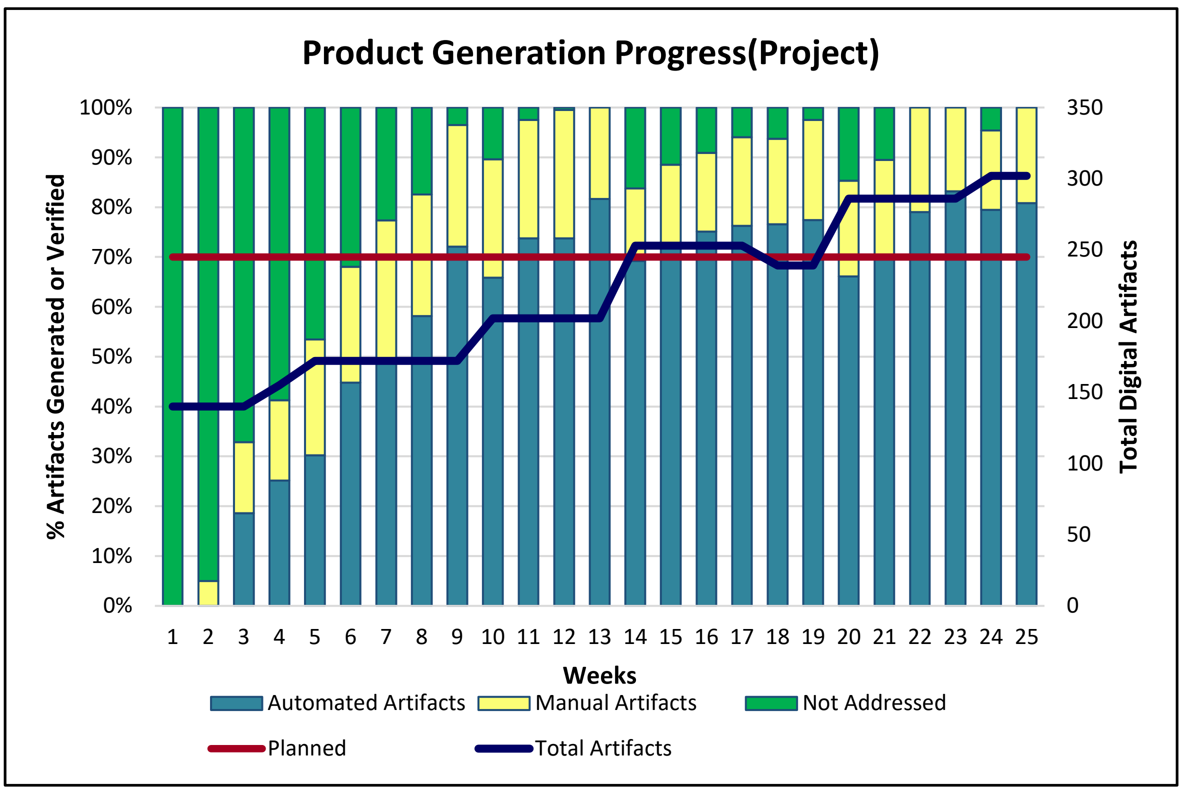


Figure 8-18: Automation Coverage (Project Level)

The project work scope may evolve iteratively (with additions, modifications, deletions) based on collaboration with acquirers and other stakeholders. The scope of the automation effort may also vary accordingly. By week 9, over 70% of the planned modeling artifacts are generated or verified using automated methods enabled by digital modeling tools. In week 18, as shown by the blue line, a modeling component was deleted from the work scope and the artifact count was reduced. Over time, additional automated artifacts are integrated into the digital model that reduce the dependence on manual development,
</td></tr>
</table>

<table>
<tr><td></td><td>documentation, and verification tasks. The project has generally met its objective of 70% of artifacts generated directly from digital models.</td></tr>
<tr><td>Indicator Description and Example

(Model-Driven Milestone Reviews)</td><td>
In traditional approaches extensive effort is often spent preparing products for milestone reviews with acquirers, such as exporting or packaging applicable products into presentation slides for review. In a model-based approach, many of the project products can be reviewed directly from modeling tools, saving effort and schedule from the presentation preparation and conduct, and using the Authoritative Source of Truth (ASOT) directly as a basis for reviews instead of using copies separate from the model itself.

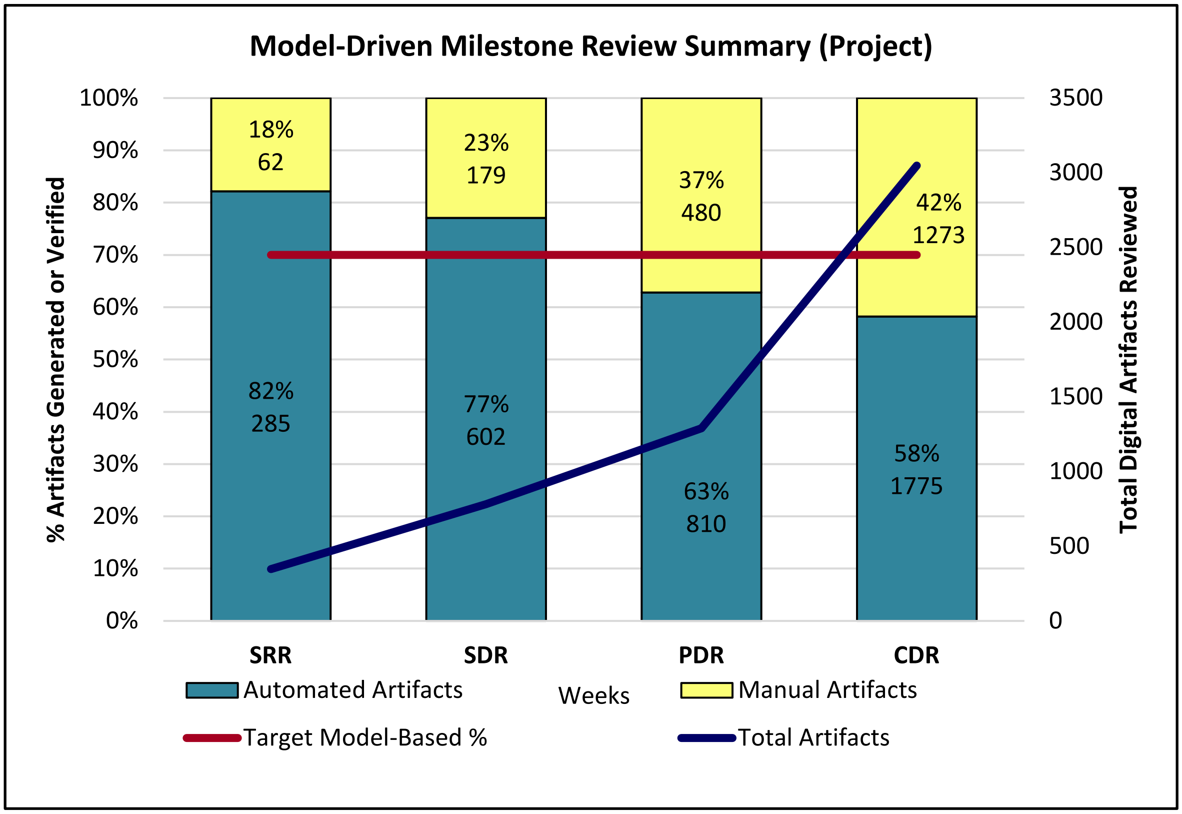


Figure 8-19: Model-Driven Design Reviews

This example depicts the relative percentage of artifacts reviewed with acquirers in a series of model-based milestone reviews. In this example, the project has established an objective of 70% of the artifacts reviewed in the milestone reviews being published directly from the digital models. This objective was met for systems engineering reviews early in the project lifecycle (System Requirements Review, System Design Review) based primarily on MBSE models. However, it proved difficult to meet this same level of model-based content for the Preliminary Design Review and Critical Design Review as the project engineering disciplines (software, mechanical, electrical) are still working toward their respective digital design transformation and integration of cross-discipline models. Further, both the supplier and acquirer base are still overcoming traditional and cultural obstacles within the acquisition system and workforce to become fully receptive to a model-based design review approach.
</td></tr>
<tr><td>Analysis Model</td><td>
% of Automated Artifacts Generated or Verified:

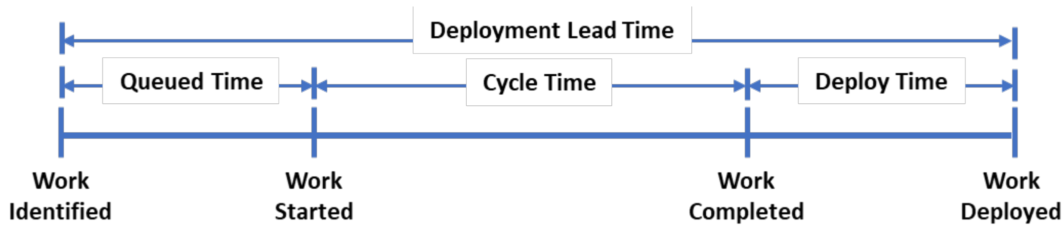
- What percentage of artifacts are automated from digital models? Is each requirement or design element fully covered by the automation, or are some aspects verified manually, or not yet verified?
- Decisions must be made on the value obtained from investing in automation. Any artifacts not generated or verified through automation must be done 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 automation.
</td></tr>
</table>

	Automated model-driven verification is a primary enabler for achieving efficiency, quality, and cost savings at both the project and organizational levels. Organizations should monitor automated verification measures in relation to achievement of their desired performance objectives.
Decision Criteria	The impact of digital modeling automation is judged best not by the quantity of artifacts generated, but by the savings in effort and schedule relative to generating and maintaining similar artifacts using manual generation and documentation-driven methods. Automation 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 DE measurement framework. Objectives for the extent of model-driven automated artifact generation may be specific to the product or domain. Automation in the range of 70%-80% is often beneficial in producing improved performance outcomes, but this may vary by domain or application.

Additional Information	
Additional Analysis Guidance	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 automation. Effort and cost measures can be correlated with digital product automation measures in order to determine the business savings and efficiencies gained.
Implementation Considerations	Relying solely on digital model artifact automation may not be wholly sufficient to exercise all functionality needed (e.g., user interfaces, quality attributes). It may be necessary to supplement automated artifact generation and verification with manual effort to adequately cover all required functionality.

Additional Specification Information	
Information Category	Process Performance
Measurable Concept	Process Efficiency - Automation
Relevant Entities	Digital modeling artifacts
Attributes	Quantity of automated artifacts generated and verified (planned and actual)
Data Collection Procedure	Data is typically collected directly from digital engineering modeling tools. Results are recorded in team tracking tools. Summaries of automated artifact generation and verification results can often be collected automatically using scripts or collected on demand.
Data Analysis Procedure	Data is reviewed and analyzed to ensure adequate quality for each candidate product. Discrepancies in process effectiveness, product quality, or coverage not meeting threshold targets may indicate updates to code or test scripts are necessary.

8.8 Deployment Lead Time

Measure Introduction	
Description	Deployment Lead Time is a measure of how rapidly authorized requests for system capabilities and work products can be engineered, developed, and delivered for use in their intended operational environment. Deployments may be related to a single product, multiple iterations of that product, or across multiple comparable products or programs. By systematically measuring the duration of processes and workflow steps in product development over time, decision makers are enabled to analyze process performance efficiency and act on bottlenecks to reduce the deployment lead time for new capabilities. Attributes characterizing the relative work performed (e.g., product requirements, model elements, product size, complexity) can be used to normalize and synthesize comparable work performed under similar defined conditions. Deployment Lead Time in aggregate generally consists of major workflow stages and milestones as depicted in Figure 8-20. Major workflow stages are Queued Time, Cycle Time, and Deploy Time. Major milestones are Work Identified, Work Started, Work Completed, and Work Deployed. Deployment Lead Time and its elements are used to evaluate efficiencies in deploying work products and as a predictor for estimating future product deployment times.  **Figure 8-20: Stages and Elements of Deployment Lead Time** These general concepts are similar to those common in many manufacturing and development domains (e.g., software agile methods). For digital engineering, the overarching objective of this specification is to characterize the process efficiency for developing and deploying digitally engineered products or models relative to traditional methods.
Relevant Terminology	Queued Time: The time a received and approved work request sits idle. Queued time includes the up-front effort needed to define and prepare the work to be implemented, such as backlog, prioritization, planning, and authorization to start work. Cycle Time: The elapsed time from when development work is started until the time development work has been completed and is ready for deployment. This time includes activities such as planning, requirements analysis, design, implementation, and testing. Cycle Time is typically targeted at measuring repeatability and predictability of team performance for well-scoped work so that results are comparable across multiple similar efforts. Deploy Time: The elapsed time to deploy completed development work for operational use. Deploy Time includes the time needed to schedule and obtain access to the operational environment for deployment to commence. Deployed means available for use as part of mission operations. If the work is deployed to multiple sites, deploy time is the time that the work is deployed at the site(s). Deployment Lead Time: The total time from when an approved request for a new capability is received until the capability is completed, deployed, and available for use in the operational environment.

Information Need and Measure Description

Information Needs (Deployment Lead Time)	How long does it take to deploy an identified feature or capability? How long does it take to deploy a viable product for operational use after a request is received? Where is the deployment bottleneck; in planning/backlog, implementation, or deployment of the implemented capability?
Information Need 2 (Cycle Time)	How long does it take to develop a digital engineering model or product?
Base Measures 1	Process Timestamps: *date* - start and end dates bounding the duration of process workflow events • **Identified Date**: timestamp when a system requirement or capability request is received and validated. The work may be queued until it is prioritized and resources are available. • **Started Date**: timestamp when the system capability request is prioritized and authorized to begin development. • **Completed Date**: timestamp when authorized work completes development (design, implementation, integration, testing) and is authorized for deployment. • **Deployed Date**: timestamp when work is deployed for use in its operational environment. Timestamps and durations of workflow events are typically in days, but projects may use different scales as appropriate.
Base Measure 2	**Product Size**: units may vary; refer to the Product Size measurement specification *Product Size is used to normalize the process workflow durations for the amount of work performed.*
Derived Measure 1	Queued Time = (Started Date - Identified Date) [integer: days]
Derived Measure 2	Cycle Time = (Completed Date - Started Date) [integer: days]
Derived Measure 3	Deploy Time = (Deployed Date - Completed Date) [integer: days]
Derived Measure 4	Deployment Lead Time = (Deployed Date - Identified Date) = (Queued Time + Cycle Time + Deploy Time)) [integer: days]

Indicator Specification	
Indicator Description and Sample (Deployment Lead Time)	In Figure 8-21, notional data for Deployment Lead Time of deployed capabilities is depicted as a stacked column with Queued Time, Cycle Time and Deploy Time shown for each capability. The height of the stacked column is Deployment Lead Time.

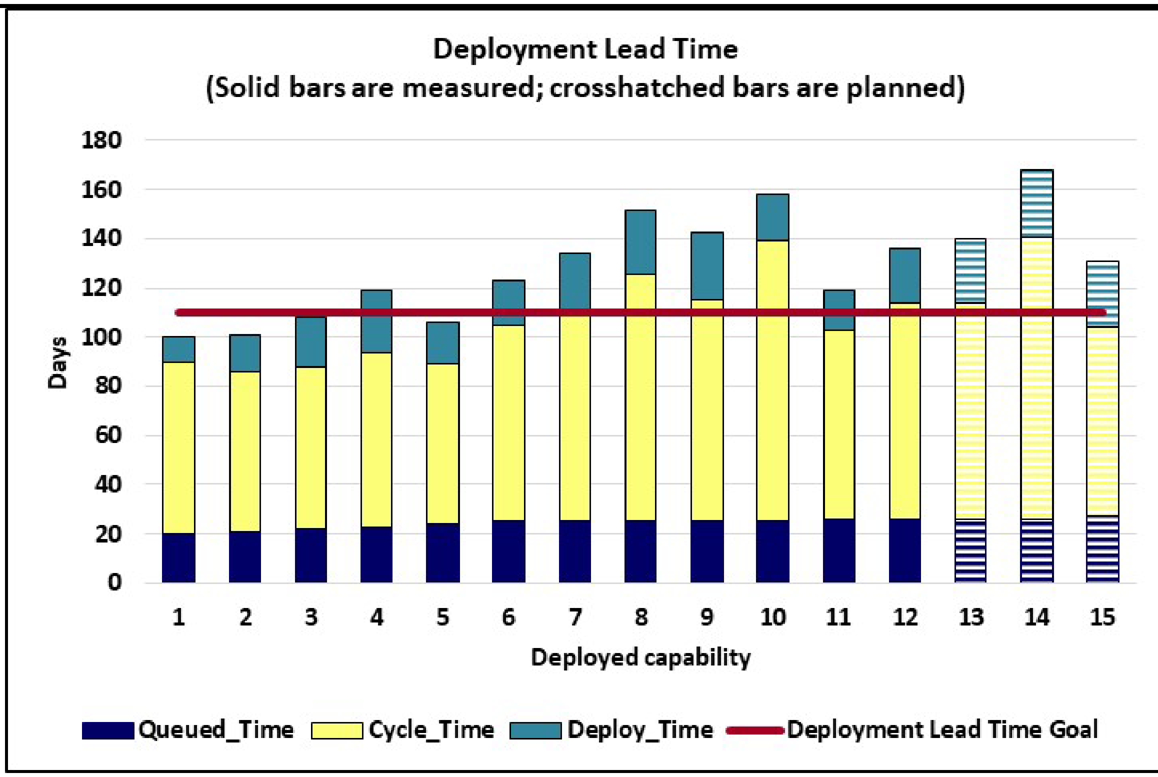
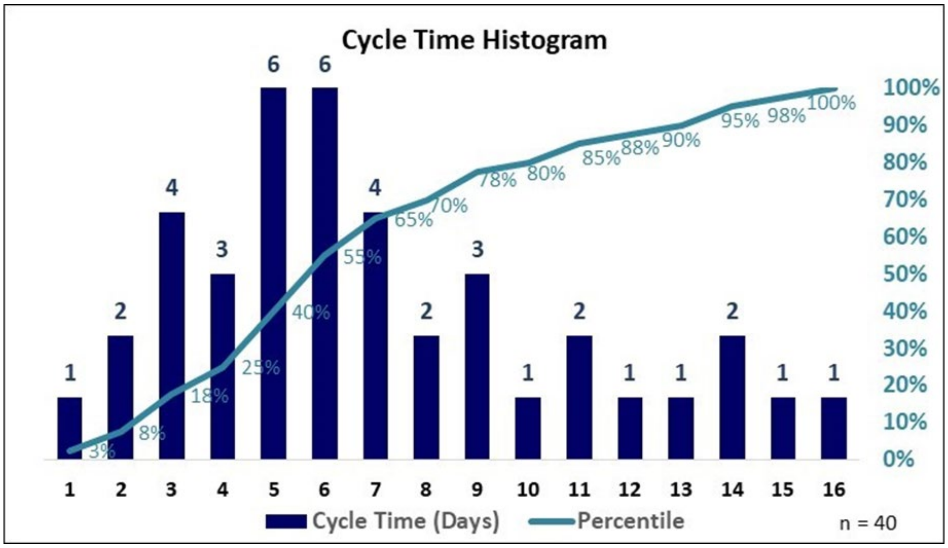


Figure 8-21: Deployment Lead Time for Operational Capabilities

This chart allows simple comparison of deployments of multiple work products and planned deployments. The table shown below provides a sample of observations that may be drawn from this chart and potential actions associated with the observations.

Observation	Analysis and Actions
The Deployment Lead Time goal was not met for last 7 completed deployments	Note that early deployment met goals. Note that large variances in Deploy Time are mostly due to variances in cycle time. Analyze root causes of Cycle Time variance.
Cycle Time variance is the largest contributor to Deploy Time variance	Analyze root causes of Cycle Time variance. Is it due to lower productivity, increased product size, or inaccurate estimates?
The planned deployments (13, 14 and 15) exceed the Deploy Time goal	Note that large Cycle Times are the main contributor. Consider ways to reduce Cycle Time, such as adding resources or deferring functionality.
Queued Time is slowly trending upward.	Analyze the root cause of increasing Queued Time. Is the increasing Queued Time indicative of an increasing backlog of approved requests?
Deploy Time has significant variation.	Determine the root cause of Deploy Time variations.*

*Oftentimes, deployment requires coordination with the acquirer or operational environment outside the supplier's control. From the supplier's perspective, potential delays in scheduling access to the operational environment can greatly affect overall Deployment Lead Time. For these reasons, measures based on Deploy Time can be interesting and useful to some extent but may be not as repeatable or actionable as Cycle Time which is more under direct project control.

Indicator Description and Example (Cycle Time)	 **Figure 8-22: Cycle Time Analysis** Cycle time performance is frequently analyzed in histograms for a set of related products, as depicted in Figure 8-22. In this example, 90% of the project's digital engineering product releases are completed in 13 days or less, 65% within 7 days. Cycle time reflects the 'Voice of the Process' from actual results. When conducted for a set of products with similar attributes (domain, scope, product size, complexity, etc.), analyses such as this can be used to characterize process capability (what is achievable?) and the likelihood of meeting project objectives or acquirer expectations ('Voice of the Customer').
Analysis Model	Shorter deployment lead times and cycle times can indicate more efficient delivery/deployment flow and quicker response to business objectives or mission needs. Longer deployment lead times and cycle times are often correlated to the scope, product size, and complexity of work products. Product Size, as an example, may be used to filter or scale the source data set for analysis of root causes of process anomalies, or to obtain greater confidence in estimates or predictions for future work. Teams should implement improvements to bring capability and performance in alignment with the mission need. Deployment lead times can often be optimized by managing depth of the work backlog, improving timely access to the operational environment for deployment, or applying additional resources to perform more work concurrently. Cycle times can also be improved by adding resources or through other approaches, such as improvements to processes, automation, or tooling. Analysis of deployment lead time and cycle time 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 deployment lead times or throughput) • Predictability for future performance (narrowing or widening standard deviation in deployment outcomes) The analyst may consider questions such as: • Are the deployment lead time and cycle time consistent across iterations? • Are durations increasing or decreasing? Why? • Does the process performance meet business objectives or the mission need? • How predictable are deployment lead time and cycle time? Can we reliably estimate future performance? • What are the process outliers? • What are the root causes for process variance? • What actions should be taken to bring performance in line with expectations?
Decision Criteria	Investigate root causes for variations. For example, review samples that are more than 10% from the average deployment time or work time.

	When Deployment Lead Time surpasses established objectives then the delays affect the operational environment. This could lead to not fielding capabilities in the operational environment within schedule.

Additional Information	
Additional Analysis Guidance	As Deployment Lead Time is analyzed, it is important to analyze its components (Queued Time, Cycle Time and Deploy Time). Each of these times will likely have different drivers as described below: • Queued Time - may be driven by backlog, release cycle and priority • Cycle Time - may be driven by work complexity and product size • Deploy Time - may be driven by operational system constraints and procedures "Completed" is expected to be defined within the project's context and criteria, e.g., definition of done. Under consistent conditions, deployment lead time can be used as a measure of team capability and throughput that can be used in lieu of traditional size-based productivity measures. Reductions in deployment lead time measures indicate faster delivery to the acquirer, which yields additional potential business benefits such as: • Identification of innovation opportunities • Higher user satisfaction and employee satisfaction • Increased productivity
Implementation Considerations	Cycle time and lead time measures can be automatically collected and analyzed by many common tool suites, or by other implementations. Data may reside in different repositories and may need to be combined for analysis.

Additional Specification Information	
Information Category	Process Performance - Process Effectiveness
Measurable Concept	Deployment Lead Time
Relevant Entities	Elapsed time duration
Attributes	Time stamps Unique Identifier for each deployed capability Identification of team and project for each work product
Data Collection Procedure	Measurement of milestone timestamps should be collected from project management and workflow tools, or from other implementations. Operational deployment milestones may be collected from acquirer business systems. Product size, if used to normalize the amount of work is collected as described in the Product Size measurement specification.
Data Analysis Procedure	Data is analyzed at the end of each deployment by the team and considered during planning sessions for the follow-on deployments. Performance trends 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 and mission objectives.

Advanced Topic - Statistical Measures for Digital Engineering Efficiency and Prediction	Digital engineering process efficiency, effort, or time-based measures, such as deployment lead time or cycle time, are enablers to characterize and act upon current performance, predict future performance, or commit to schedules for estimating future work. Hence, projects and organizations will want a higher level of confidence in the integrity and representativeness of their data sets for decision making. That likely involves more detailed analysis such as: • Applying statistical methods to gain a deeper understanding of process performance, capability, and predictability

- Analyzing sources of variation (common causes, special causes), especially anomalies or outliers outside typical ranges of process performance
- Decomposing process elements and dependencies to deep dive into key issues or bottlenecks

Example methods might include:

- Statistical measures: mean, median, standard deviation, inter-quartile ranges, outliers, etc.
- Analyses: frequency distribution curves, scatter plots, box, and whisker plots, etc.

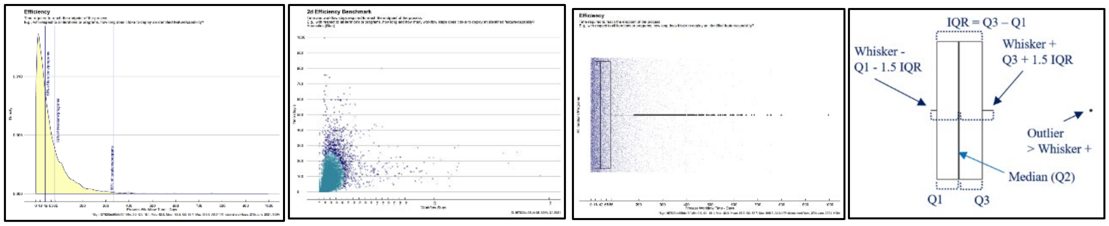


Figure 8-23: Plots and Advanced Analyses

In the context of this measurement specification, this could support more advanced analyses with greater insight, such as:

- Plotting cycle time or deployment time vs. a separate measure of interest (e.g., workflow steps, defects)
- Plotting absolute frequency of workflow steps vs. start/end boundaries, or durations
- Identifying distinct workflow patterns
- Monitoring trends in the cumulative flow of work across process states, stages, or milestones

Armed with this insight, stakeholders and projects can make better informed decisions based on a detailed understanding of their process capability.

This discussion introduces high level concepts, only. A more detailed description with measures and mathematical analyses is beyond the scope of this digital engineering measurement framework document. Further description and details are published in a white paper on the PSM website, https://www.psmsc.com/Prod_TechPapers.asp

8.9 Runtime Performance

Measure Introduction	
Description	Ensuring the efficient performance of deployed operational systems is fundamental to meeting business requirements or satisfying a mission need. Performance analysis is critical to early requirements development, architecture, and design processes to ensure the ultimate target solution is feasible. This is generally done through sophisticated models, simulations, and prototypes to validate applicable algorithms or ranges of performance prior to final implementation and deployment in the operational environment. In a digital engineering environment nearly all artifacts are digital, and integration of the tech stack enables stakeholders to maintain and collaborate around an Authoritative Source of Truth (ASOT) for engineering design, review, and validation. The tech stack hosts models that form a digital twin. The runtime infrastructure and performance become crucial concerns in this environment, enabling applicable cross-functional elements to converge on trade-off analyses toward a feasible optimized solution. Runtime performance is a particular concern for models that tax the computing infrastructure, where data latency or sluggish infrastructure performance can have significant adverse effects on the digital design effort. Runtime performance is the amount of time, or duration, that it takes a software system to perform or execute one of its capabilities. By systematically measuring the modeled or implemented runtime performance of alternative solutions, suppliers are able to analyze the likelihood of best meeting operational performance requirements and respond early, as required. During the design phase, analysts can plot performance analyses based on historical data to tailor future capabilities to their expected environments and workloads. This specification introduces summary concepts for measuring runtime performance in a digital engineering environment. Details are beyond the scope of this specification but are described in a separate Digital Engineering Addendum white paper on the PSM website.
Relevant Terminology	Please refer to Vol. 1 of the DE paper.

Information Need and Measure Description	
Information Need	• What is the runtime performance of the capability or system? • What is the likelihood that runtime performance will meet operational requirements? • Where are the runtime performance bottlenecks, and how can operational performance be optimized?
Base Measure(s) 1	Runtime performance timestamps: date and time • Runtime Performance Start - start timestamp for a runtime performance (interval) • Runtime Performance End - end timestamp for a runtime performance (interval)
Base Measure(s) 2	Additional Technical Measures within the Runtime Ecosystem: definition and units vary Often design decisions depend not only on the measured runtime performance but also on a relationship with one or more dependent measures, such as consumption of other computing resources (e.g., memory utilization and bandwidth). The combination of measures can be analyzed in trade off analyses to determine an optimized solution.
Derived Measure 1	Elapsed Time = (Runtime Performance End) - (Runtime Performance Start) (interval) *Duration between start and end of a performance interval*

Derived Measures 2	Statistical analysis: measures of runtime performance across a set of measured time intervals. e.g., min; max; mean; median; standard deviation; percentiles; and outliers These common derived statistical measures and analyses are well defined in practice and are not detailed in this measurement specification. Example statistical graphs and indicators include box plots, scatter graphs, distribution profiles, histograms, etc. Refer to the separate Digital Engineering Efficiency white paper on the PSM website for further description and examples: https://www.psmsc.com/Prod_TechPapers.asp
Derived Measures 3	Runtime performance benchmarks: time interval Time required to compute or perform a capability, process, subprocess, or activity. May include a set of iterations (1.. n) or weightings to create a linear combination.
Derived Measures 4	Multivariate analyses: varies by selected parameters in relationships with runtime performance. Correlations between Base Measures 1 and Base Measures 2 in runtime performance results.

Indicator Specification

Indicator Description and Sample

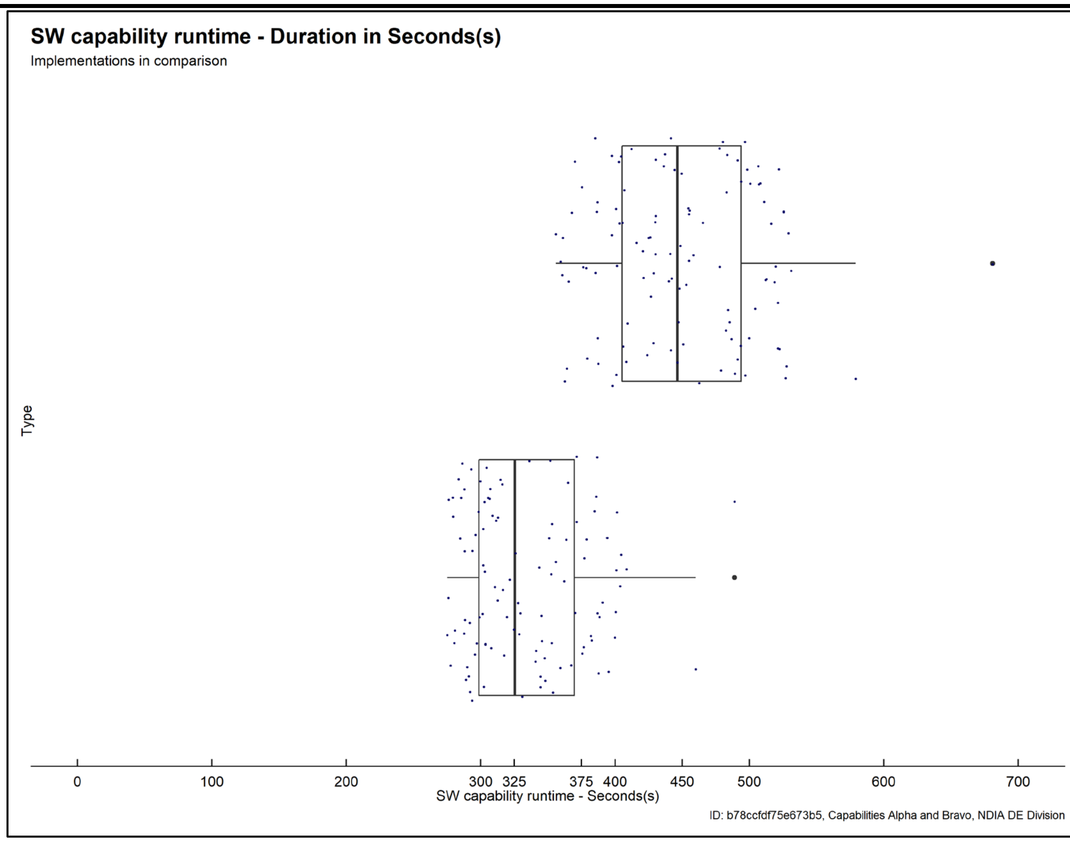


Figure 8-24: Runtime Performance Plot [1]

This indicator uses box plots to contrast the runtime performance results for two alternative implementations. Multiple runtime performance samples for each alternative are summarized in box plots, which include statistical depictions of the sample median, Interquartile Ranges (IQRs), dispersion of measured data points, and outliers. In this example, module Alpha (bottom) features faster median runtime performance, 325.4 seconds, than module Bravo (bottom), 446.4 seconds, due to the utilization of refactored code.

1 Copyright 2021 by Richard Halliger. Reprinted with permission.

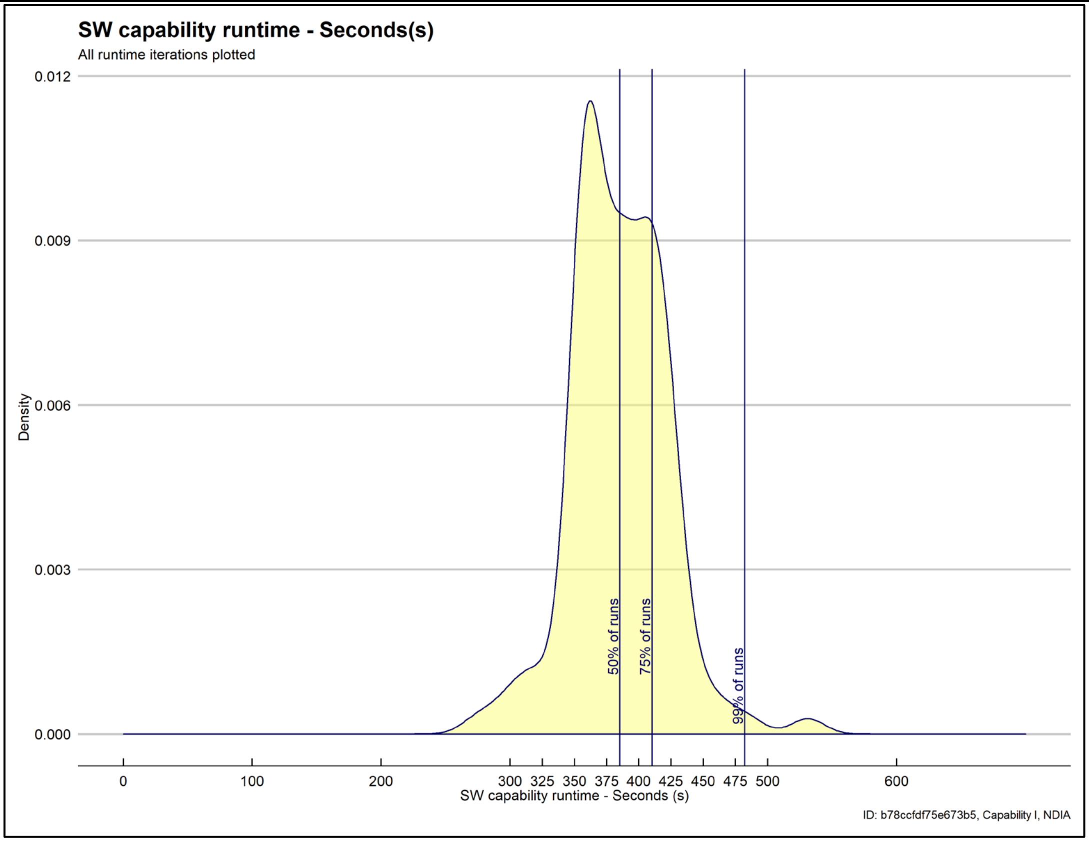


Figure 8-25: Runtime Performance Density Distribution [2]

This indicator plots the density distribution of runtime performance samples, with percentiles for the probabilities of performance within performance ranges. In this example, 50% of runtime performance samples are measured at ≤ 385.3 seconds, 75% of samples are ≤ 410.3 seconds, and 99% of samples are ≤ 482.1 seconds. The program team can use analyses such as these to consider the likelihood that alternative solutions meet operational performance objectives.

2 Copyright 2021 by Richard Halliger. Reprinted with permission.

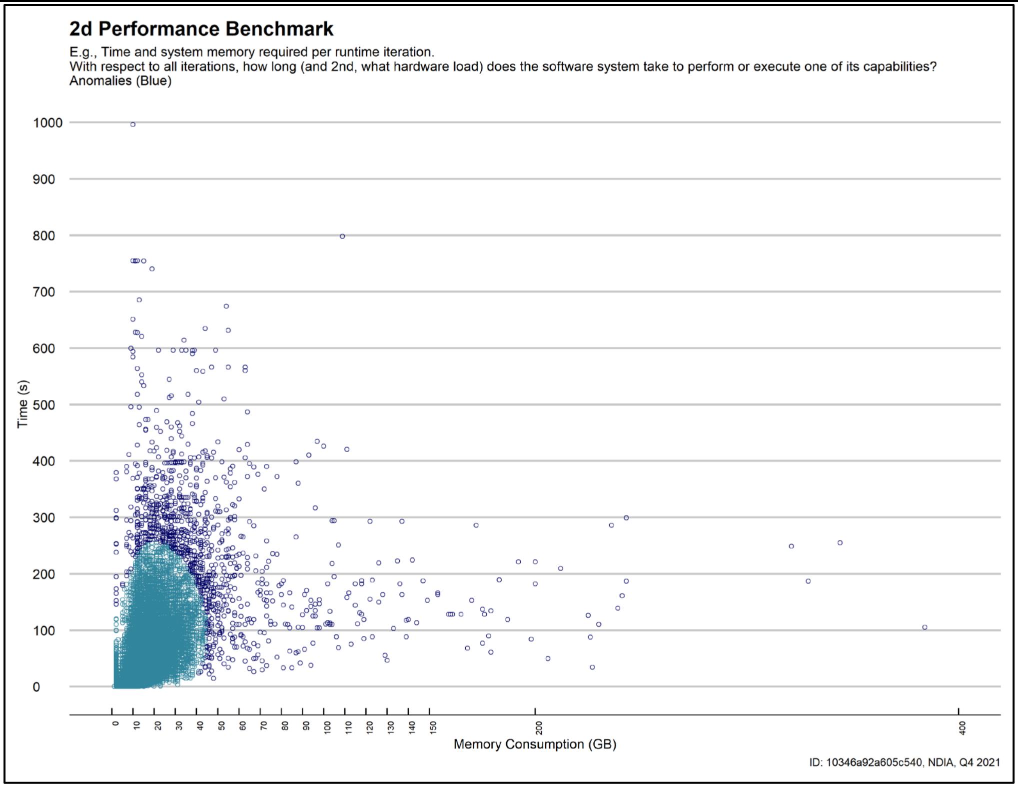


Figure 8-26: Anomaly Analysis [3]

This indicator uses a two-dimensional scatter graph to depict both the measured runtime performance against a second technical measure of interest within the runtime ecosystem. The analysis of both measures in combination can be used to determine an optimized solution. In this DE environment example, there seems to be a relationship between runtime performance and the memory consumption. Distinct runtime iterations anomalies are depicted via purple points. All other points seem to be clustered in the 0 - 200 seconds and the 1 - 50 GB memory consumption range.

Analysis Model

Figure 8-24 depicts the utilization of common, derived statistical measures that form runtime performance box plots. These allow the analyst to conduct a comprehensive, first-glance runtime assessment of the

- "best case" runtime,
- "worst" runtime,
- "center of gravity" of the runtimes,
- "spread", i.e., degree of dependability for the end user,
- "best" and "worst" quarters of runtimes, and
- runtimes the end user might expect over multiple iterations.

With respect to Figure 2, analysts might utilize customizable percentiles, e.g., 75%, to assess whether the DE capability meets the specific performance requirements of the acquirer. Additionally, extreme runtimes might be identified.

While specific tests for outlier detection (See Derived Measure 2) might be utilized at the beginning of an analysis, advanced algorithms enable the analysis of >10k runtimes in a reasonable amount of time (e.g., 5 to 60+ seconds). Figure 3 visualizes the results of an anomaly detection run. Analysts might assess the range and typical clusters of the two measures of interests at first glance. Additionally, this graph supports continuous

	monitoring, e.g., one might visually assess the results of interventions, e.g., hardware changes, by comparing a pre-change and post-change anomaly run visualization.
Decision Criteria	**Figure 8-24:** Outliers, i.e., points outside the whiskers of the box graphs, or extensive whiskers exist: The runtime of the DE capability might not be reliable or specific case cases lead to extensive resource and time consumptions. When one of the compared box graphs features one or more of the following possible observations, the decision maker shall favor this DE capability and the associated supplier, with: • a lower median score, • smaller IQRs, • less outliers, or • shorter whiskers. **Figure 8-25:** DE capability does not meet the specific performance requirements: Each quarter or milestone, the acquirer might brief the DE capability supplier about the (objective) status quo of the capability and request more contextual data of these extreme runtime cases for further analysis. Moreover, the supplier might elaborate on these extreme cases. **Figure 8-26:** Anomalous runtime and the Additional Technical Measure (See Base Measure 2) combinations that are feature over 100% higher runtime or Additional Technical Measure readings have been identified: The decision maker shall order the replacement of software modules and an in-depth analyses of the specific DE capability runs that feature these extensive resource and time consumptions.

Additional Information	
Additional Analysis Guidance	Customized measures, e.g., statistical tests for outlier detection might enable more sophisticated analysis. On the capability level, i.e., complex system, one might account for module interaction effects. These apply to complex systems that feature constrained hardware or network resources, e.g., due to undersized microelectronics or design decisions. Please refer to: https://www.psmsc.com/Prod_TechPapers.asp
Implementation Considerations	Suppliers/analysts might integrate specific modules or lines of code that benchmark specific parts of the call stack. For instance, one might calculate the time taken of a method that loads structured data into memory. Via monitoring efforts, e.g., via logging, suppliers gain an understanding of the runtime of their code.

Additional Specification Information	
Information Category	Technology Effectiveness
Measurable Concept	Technology Performance
Relevant Entities	Runtimes of the DE capability Runtime measurements and associated information (See Attributes)
Attributes	Time stamps (See Base Measures) *[mandatory]* ID *[optional]* Additional Technical Measures *[optional]* Operational environment or contextual information *[optional]* Additional information of analytical interest *[optional]*

Data Collection Procedure	Common elements of the collection process: Existing log files might serve as a starting point. However, to enable runtime monitoring and analysis, the data collection needs shall be defined by stakeholders and analysts. Collect duration timestamps using performance monitoring implementations or tools.
Data Analysis Procedure	The analyst might assess specific capabilities, modules, or submodules by filtering. Aggregations enable further computations. Analyses are built-in capabilities of the performance monitoring tools or statistical packages. The actual implementation of the analyses vary, e.g., some might utilize built-in capabilities of performance monitoring tools, additional source code, or statistical packages.

BIBLIOGRAPHY

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

[2] Office of the Deputy Assistant Secretary of Defense (Systems Engineering) [ODASD (SE), "DAU Glossary: Digital Engineering," Defense Acquisition University (DAU), 2017. [Online]. Available: https://www.dau.mil/glossary/pages/3626.aspx.

[3] ISO/IEC/IEEE draft DIS 24641:2021(E) standard: Systems and software engineering – Methods and tools for Model-based systems and software engineering (note this document version was released 9/2021 for comment to the working group and is not yet released for use).

[4] J. Lubell, K. Chen, S. Frechette, J. Horst and P. Huang, "NIST Technical Note 1753: Model Based Enterprise / Technical Data Package Summit Report," August 2012. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/TechnicalNotes/NIST.TN.1753.pdf.

[5] https://modelbasedengineering.com/faq/

[6] ISO/IEC/IEEE 15939:2017 Systems and software engineering — Measurement process; https://www.iso.org/standard/71197.html

[7] https://www.psmsc.com/

[8] Department of Defense Digital Engineering Strategy June 2018; https://ac.cto.mil/wp-content/uploads/2019/06/2018-Digital-Engineering-Strategy_Approved_PrintVersion.pdf.

[9] https://www.omgwiki.org/MBSE/doku.php?id=mbse:deix

[10] INCOSE Model-Based Enterprise Capability Matrix and User's Guide, Version 1.0, January 2020; https://connect.incose.org/Pages/Product-Details.aspx?ProductCode=MBCM.

[11] http://www.psmsc.com/UG2019/Workshops/w02.zip

[12] https://connect.incose.org/Pages/Product-Details.aspx?ProductCode=TechGuideLeadInSoft

[13] http://www.psmsc.com/CIDMeasurement.asp

[14] INCOSE Model-Based Enterprise Capability Matrix and User's Guide, Version 1.0, January 2020

[15] https://sercuarc.org/wp-content/uploads/2020/03/SERC-SR-2020-001-Benchmarking-the-Benefits-and-Current-Maturity-of-MBSE-3-2020.pdf

[16] https://sercuarc.org/wp-content/uploads/2020/06/SERC-SR-2020-003-DE-Metrics-Summary-Report-6-2020.pdf

[17] Henderson, K., Salado, A., McDermott, T., and Van Aken, E. "Measurement Framework for MBSE," American Society for Engineering Management 2021 International Annual Conference and 42nd Annual Meeting, 27 - 30 October 2021.

[18] L. Delligati, SysML Distilled: A Brief Guide to the Systems Modeling Language, Addison-Wesley 2013.

[19] https://en.wikipedia.org/wiki/Single_source_of_truth.

[20] https://www.omgwiki.org/MBSE/doku.php?id=mbse:authoritative_source_of_truth.

[21] https://www.omgwiki.org/MBSE/doku.php?id=mbse:authoritative_source_of_truth.

[22] ISO/IEC/IEEE draft DIS 24641:2021(E) standard: Systems and software engineering – Methods and tools for Model-based systems and software engineering

[23] L. Delligati, SysML Distilled: A Brief Guide to the Systems Modeling Language, Addison-Wesley 2013.

[24] https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/sysml_model_elements.html

[25] https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=models-uml-model-elements

[26] ISO/IEC/IEEE 15288:2015 Systems and software engineering — System life cycle processes

[27] portions adapted from https://www.ibm.com/docs/en/urbancode-release/6.1.1?topic=lifecycles-phases-gates