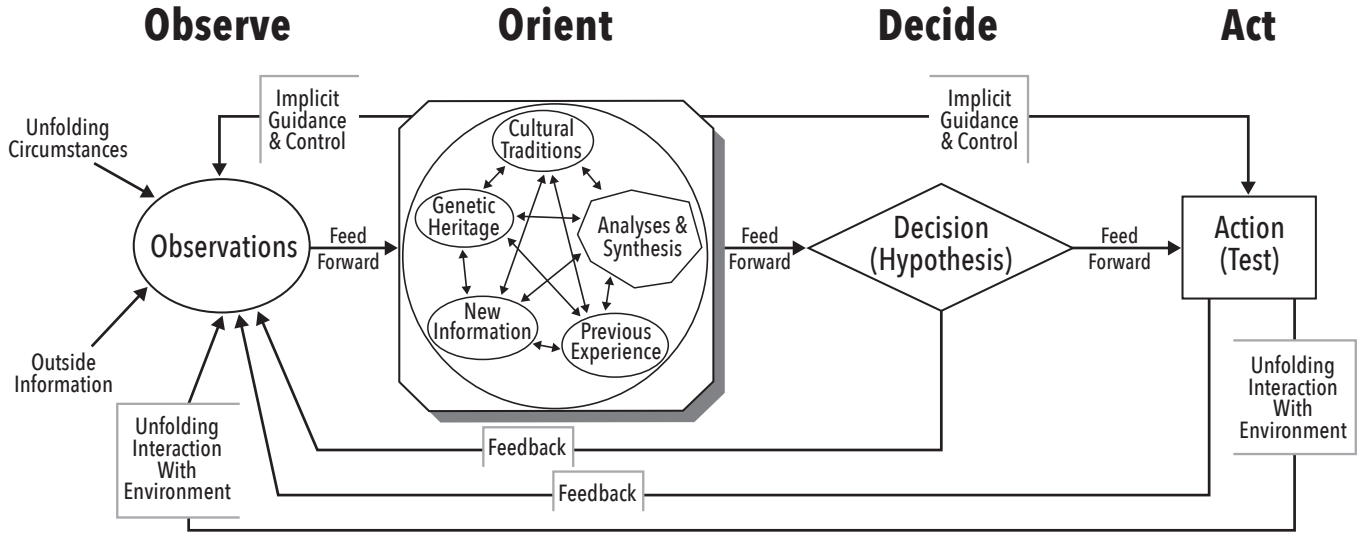
INSIGHT

This Issue's Feature:

Engineering In and For Dynamic, Uncertain Environments



OODA Loop diagram source from the 2012 paper:
Boyd's OODA Loop (It's Not What You Think)
by Chet Richards
Permission granted to INCOSE to publish and use.

OP-ED

From Silos to Services: Systems Engineering for the Digital Enterprise—Why This Matters to Our INCOSE Teammates

by Justin Fanelli, Chief Technology Officer, U.S. Department of the Navy, doncio.navy.mil

AUGUST 2026
VOLUME 29 / ISSUE 4

A PUBLICATION OF THE INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING

INCOSE

Alignment is the Advantage

Train for Skills. Align for Capabilities.

Caltech CTME designs custom learning that aligns engineers, leaders, and organizations around shared systems thinking—so teams move faster, smarter, together.

NEW CERTIFICATE PROGRAMS

- Transitioning Models to SysML v2
- MBSE Fundamentals Using SysML v2
- AI-Assisted MBSE

Caltech Center for Technology & Management Education

CTME.CALTECH.EDU

ACADEMIC EQUIVALENCY INCOSE SEP SYSTEMS ENGINEERING PROFESSIONAL

INSIGHT

A PUBLICATION OF THE INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING

AUGUST 2026 VOLUME 29/ ISSUE 4

Inside this issue

About This Publication

INFORMATION ABOUT INCOSE

INCOSE's membership extends to over 28,000 members and CAB associates and more than 200 corporations, government entities, and academic institutions. Its mission is to share, promote, and advance the best of systems engineering from across the globe for the benefit of humanity and the planet. INCOSE charters chapters worldwide, includes a corporate advisory board, and is led by elected officers and directors.

For more information, click here:
The International Council on Systems Engineering
(www.incose.org)
INSIGHT is the magazine of the International Council on Systems Engineering. It is published six times per year and

OVERVIEW

features informative articles dedicated to advancing the state of practice in systems engineering and to close the gap with the state of the art. ***INSIGHT*** delivers practical information on current hot topics, implementations, and best practices, written in applications-driven style. There is an emphasis on practical applications, tutorials, guides, and case studies that result in successful outcomes. Explicitly identified opinion pieces, book reviews, and technology roadmapping complement articles to stimulate advancing the state of practice. ***INSIGHT*** is dedicated to advancing the INCOSE objectives of impactful products and accelerating the transformation of systems engineering to a model-based discipline.

Topics to be covered include resilient systems, model-based systems engineering, commercial-driven transformational systems engineering, digital engineering, artificial intelligence, natural systems, agile security, systems of systems, and cyber-physical systems across disciplines and domains of interest to the constituent groups in the systems engineering community: industry, government, and academia. Advances in practice often come from lateral connections of information dissemination across disciplines and domains. ***INSIGHT*** will track advances in the state of the art with follow-up, practically written articles to more rapidly disseminate knowledge to stimulate practice throughout the community.

PERMISSIONS

*** PLEASE NOTE: If the links highlighted here do not take you to those web sites, please copy and paste address in your browser.**

Permission to reproduce Wiley journal Content:
Requests to reproduce material from John Wiley & Sons publications are being handled through the RightsLink® automated permissions service.

Simply follow the steps below to obtain permission via the Rightslink® system:

- Locate the article you wish to reproduce on Wiley Online Library (http://onlinelibrary.wiley.com)
- Click on the 'Request Permissions' link, under the ‹ARTICLE TOOLS› menu on the abstract page (also available from Table of Contents or Search Results)
- Follow the online instructions and select your requirements from the drop down options and click on 'quick price' to get a quote
- Create a RightsLink® account to complete your transaction (and pay, where applicable)
- Read and accept our Terms and Conditions and download your license
- For any technical queries please contact customercare@copyright.com
- For further information and to view a Rightslink® demo please visit www.wiley.com and select Rights and Permissions.

AUTHORS – If you wish to reuse your own article (or an amended version of it) in a new publication of which you are the author, editor or co-editor, prior permission is not required (with the usual acknowledgements). However, a formal grant of license can be downloaded free of charge from RightsLink if required.

Photocopying
Teaching institutions with a current paid subscription to the journal may make multiple copies for teaching purposes without charge, provided such copies are not resold or copied. In all other cases, permission should be obtained from a reproduction rights organisation (see below) or directly from RightsLink®.

Copyright Licensing Agency (CLA)
Institutions based in the UK with a valid photocopying and/or digital license with the Copyright Licensing Agency may copy excerpts from Wiley books and journals under the terms of their license. For further information go to CLA.

Copyright Clearance Center (CCC)
Institutions based in the US with a valid photocopying and/or digital license with the Copyright Clearance Center may copy excerpts from Wiley books and journals under the terms of their license, please go to CCC.

Other Territories: Please contact your local reproduction rights organisation. For further information please visit www.wiley.com and select Rights and Permissions.
If you have any questions about the permitted uses of a specific article, please contact us.

Permissions Department – UK
John Wiley & Sons Ltd.
The Atrium,
Southern Gate,
Chichester
West Sussex, PO19 8SQ
UK
Email: Permissions@wiley.com
Fax: 44 (0) 1243 770620
or

Permissions Department – US
John Wiley & Sons Inc.
111 River Street MS 4-02
Hoboken, NJ 07030-5774
USA
Email: Permissions@wiley.com
Fax: (201) 748-6008

ARTICLE SUBMISSION

insight@incose.net

Publication Schedule. ***INSIGHT*** is published six times per year. Issue and article submission deadlines are as follows:

- October 2026 issue – 1 July 2026
- December 2026 issue – 1 September 2026

For further information on submissions and issue themes, visit the INCOSE website: www.incose.org

 ISSN 2156-485X; (print) ISSN 2156-4868 (online)

ADVERTISE

Readership
INSIGHT reaches over 27,000 members and CAB associates and uncounted employees and students of more than 130 CAB organizations worldwide. Readership includes engineers, manufacturers/purchasers, scientists, research and development professionals, presidents and chief executive officers, students, and other professionals in systems engineering.

Issuance	Circulation
2026, Vol 29, 6 Issues	100% Paid

Contact us for Advertising and Corporate Sales Services
We have a complete range of advertising and publishing solutions professionally managed within our global team. From traditional print-based solutions to cutting-edge online technology the Wiley-Blackwell corporate sales service is your connection to minds that matter. For an overview of all our services please browse our site which is located under the Resources section. Contact our corporate sales team today to discuss the range of services available:

- Print advertising for non-US journals
- Email Table of Contents Sponsorship
- Reprints
- Supplement and sponsorship opportunities
- Books
- Custom Projects
- Online advertising

Click on the option below to email your enquiry to your nearest office:

- Asia and Australia corporatesalesaustralia@wiley.com
- Europe, Middle East and Africa (EMEA) corporatesaleseurope@wiley.com
- Japan corporatesalesjapan@wiley.com
- Korea corporatesaleskorea@wiley.com

USA (also Canada, and South/Central America):

- Healthcare Advertising corporatesalesusa@wiley.com
- Science Advertising Ads_sciences@wiley.com
- Reprints Commercialreprints@wiley.com
- Supplements, Sponsorship, Books and Custom Projects busdev@wiley.com

Or please contact: Marcom@incose.net

CONTACT

Questions or comments concerning:

Submissions, Editorial Policy, or Publication Management
Please contact: William Miller, Editor-in-Chief
insight@incose.net

Advertising—*please contact:*
Marcom@incose.net

Member Services – *please contact:* info@incose.org

ADVERTISER INDEX

CORPORATE ADVISORY BOARD — MEMBER COMPANIES

3DSE Management Consultants
Advanced Systems Engineering, LLC
Aerospace Corporation, The
Airbus
Albers Aerospace
Analog Devices, Inc.
ANSYS, Inc
Arcfield
Auburn University
Australian National University
Aviation Industry Corporation of China, LTD
BAE Systems
Bechtel
Becton Dickinson
BlueHalo Labs An AV Company
BMT Canada
Boeing Company, The
Bombardier
Booz Allen Hamilton Inc.
Boston Scientific Corporation
BTS Software Solutions
California State University Dominguez Hills
Caltech California Institute of Technology
Capgemini Engineering
Carnegie Mellon Univ. Software Engineering Institute
Change Vision, Inc.
Colorado State University Systems Engineering Programs
Commercial Aircraft Corporation of China, Ltd (COMAC)
Cornell University
Cranfield University
CT Engineering Group
Cubic Corporation
Cummins, Inc.
Dassault Systèmes
Deloitte Consulting, LLC
Denso Create Inc
DENTSU SOKEN INC
DigiFlight , Inc.
Drexel University
Eaton
EMBRAER
Embry-Riddle Aeronautical University
FAMU-FSU College of Engineering
Federal Aviation Administration (U.S.)
Florida Institute of Technology
Ford Motor Company
GE Aerospace
General Dynamics
General Motors
George Mason University
Georgia Institute of Technology
Hitachi Energy
Honeywell Aerospace Technologies
Huawei Technologies Co. Ltd
Idaho National Laboratory
IQNOX, LLC
ISAE - Supaero
ISDEFE
IVECO Group
Jama Software
Japan Aerospace Exploration Agency
Jet Propulsion Laboratory
John Deere & Company
Johns Hopkins University
KBR, Inc.
KEIO University
L3Harris Technologies
Lawrence Livermore National Laboratory
Leidos
LEONARDO
Lockheed Martin Corporation
Los Alamos National Laboratory
Loyola Marymount University
Magna
ManTech International Corporation
Massachusetts Institute of Technology
MBDA (UK) Ltd
Medtronic
MetaTech Consulting Inc.
Missouri University of Science & Technology
MITRE Corporation, The
Mitsubishi Electric Corporation
Mitsubishi Heavy Industries, Ltd
Modern Technology Solutions Inc
National Aeronautics and Space Administration (NASA)
National Reconnaissance Office (NRO)
Naval Postgraduate School
Nissan Motor Co, Ltd
Northrop Grumman Corporation
ONE-SYS Srl
Pacific Northwest National Laboratory
PARAMETRIC TECHNOLOGY GMBH PTC
Pennsylvania State University
Petronas International Corporation Limited
Prime Solutions Group, Inc
Project Performance International (PPI)
Purdue University
RealmOne
Redwire Space
Rolls-Royce
RTX
Saab AB
SAFRAN
SAIC
Shanghai Formal-Tech Information Technology Co., Ltd
Shell
Siemens
Sierra Nevada Corporation
Singapore Institute of Technology
Southern Methodist University
Space Dynamics Laboratory
SPEC Innovations
Stevens Institute of Technology
Strategic Technical Services LLC
Studio SE, Ltd.
Suranaree University of Technology
Swedish Defence Materiel Administration (FMV)
Systems Planning and Analysis
System Strategy, Inc (SSI)
Taiwan Space Agency
Tata Consultancy Services
Terumo BCT Inc
Thales
The Afeka Academic College of Engineering
The George Washington University
The University of Arizona
The University of Texas at Arlington
The University of Utah
Torch Technologies
TOSHIBA Corporation
Trane Technologies
Tsinghua University
UK MoD
UNCOMN
Universidade Federal De Minas Gerais
University of Alabama in Huntsville
University of Arkansas
University of California San Diego
University of Central Florida
University of Connecticut
University of Maryland
University of Maryland, Baltimore County
University of Maryland Global Campus
University of Michigan, Ann Arbor
University of New South Wales, The, Canberra
University of South Alabama
University of South-Eastern Norway (USN)
University of Southern California
University of Texas at Austin
University of Texas at El Paso (UTEP)
U.S. Coast Guard
US Department of Defense
Vector Informatik GmbH
Veoneer US Safety Systems, LLC
Virginia Tech
Volvo Cars Corporation
Volvo Construction Equipment
Wabtec Corporation
Warfighting Acquisition University
Wayne State University
Weber State University
Wichita State University College of Engineering
Woodward Inc
Worcester Polytechnic Institute (WPI)
Woven by Toyota, Inc.
Yulista Services , Inc.
Zuken, Inc

Systems Engineering: The Journal of The International Council on Systems Engineering

Call for Papers

The *Systems Engineering* journal is intended to be a primary source of multidisciplinary information for the systems engineering and management of products and services, and processes of all types. Systems engineering activities involve the technologies and system management approaches needed for

- definition of systems, including identification of user requirements and technological specifications;
- development of systems, including conceptual architectures, tradeoff of design concepts, configuration management during system development, integration of new systems with legacy systems, integrated product and process development; and
- deployment of systems, including operational test and evaluation, maintenance over an extended life cycle, and re-engineering.

Systems Engineering is the archival journal of, and exists to serve the following objectives of, the International Council on Systems Engineering (INCOSE):

- To provide a focal point for dissemination of systems engineering knowledge
- To promote collaboration in systems engineering education and research
- To encourage and assure establishment of professional standards for integrity in the practice of systems engineering
- To improve the professional status of all those engaged in the practice of systems engineering
- To encourage governmental and industrial support for research and educational programs that will improve the systems engineering process and its practice

The journal supports these goals by providing a continuing, respected publication of peer-reviewed results from research and development in the area of systems engineering. Systems engineering is defined broadly in this context as an interdisciplinary approach and means to enable the realization of successful systems that are of high quality, cost-effective, and trustworthy in meeting customer requirements.

The *Systems Engineering* journal is dedicated to all aspects of the engineering of systems: technical, management, economic, and social. It focuses on the life cycle processes needed to create trustworthy and high-quality systems. It will also emphasize the systems management efforts needed to define, develop, and deploy trustworthy and high quality processes for the production of systems. Within this, *Systems Engineering* is especially concerned with evaluation of the efficiency and effectiveness of systems management, technical direction, and integration of systems. *Systems Engineering* is also very concerned with the engineering of systems that support sustainable development. Modern systems, including both products and services, are often very knowledge-intensive, and are found in both the public and private sectors. The journal emphasizes strategic and program management of these, and the information and knowledge base for knowledge principles, knowledge practices, and knowledge perspectives for the engineering of systems. Definitive case studies involving systems engineering practice are especially welcome.

The journal is a primary source of information for the systems engineering of products and services that are generally large in scale, scope, and complexity. *Systems Engineering* will be especially concerned with process- or product-line–related efforts needed to produce products that are trustworthy and of high quality, and that are cost effective in meeting user needs. A major component of this is system cost and operational effectiveness determination, and the development of processes that ensure that products are cost effective. This requires the integration of a number of engineering disciplines necessary for the definition, development, and deployment of complex systems. It also requires attention to the lifecycle process used to produce systems, and the integration of systems, including legacy systems, at various architectural levels. In addition, appropriate systems management of information and knowledge across technologies, organizations, and environments is also needed to insure a sustainable world.

The journal will accept and review submissions in English from any author, in any global locality, whether or not the author is an INCOSE member. A body of international peers will review all submissions, and the reviewers will suggest potential revisions to the author, with the intent to achieve published papers that

- relate to the field of systems engineering;
- represent new, previously unpublished work;
- advance the state of knowledge of the field; and
- conform to a high standard of scholarly presentation.

Editorial selection of works for publication will be made based on content, without regard to the stature of the authors. Selections will include a wide variety of international works, recognizing and supporting the essential breadth and universality of the field. Final selection of papers for publication, and the form of publication, shall rest with the editor.

Submission of quality papers for review is strongly encouraged. The review process is estimated to take three months, occasionally longer for hard-copy manuscript.

Systems Engineering operates an online submission and peer review system that allows authors to submit articles online and track their progress, throughout the peer-review process, via a web interface. All papers submitted to *Systems Engineering*, including revisions or resubmissions of prior manuscripts, must be made through the online system. Contributions sent through regular mail on paper or emails with attachments will not be reviewed or acknowledged.

All manuscripts must be submitted online to *Systems Engineering* at ScholarOne Manuscripts, located at:

https://mc.manuscriptcentral.com/SYS

Full instructions and support are available on the site, and a user ID and password can be obtained on the first visit.

FROM THE EDITOR-IN-CHIEF

William Miller, insight@incose.net

We are pleased to publish the August 2026 issue of *INSIGHT* published in cooperation with John Wiley & Sons as a magazine for systems engineering practitioners. The *INSIGHT* mission is to provide informative articles for advancing the state of the practice of systems engineering. The intent is to accelerate the dissemination of knowledge to close the gap between the state of practice and the state of the art as captured in *Systems Engineering*, the Journal of INCOSE, also published by Wiley. The August *INSIGHT* theme is *engineering in and for dynamic, uncertain environments*.

We lead with the op-ed directed to INCOSE members and the broader systems and engineering communities by Justin Fanelli, chief technology officer (CTO) of the U.S. Department of the Navy (DON). Justin's op-ed "From Silos to Services: Systems Engineering for the Digital Enterprise" addresses today's challenges common to large enterprises including defense: "how to move at the speed of the problem" securely with emphasis on measurable outcomes. The Department of the Navy has substantiated, validated, and implemented Modern Service Delivery (MSD) "as a repeatable systems engineering approach, moving from program-centric silos to secure, composable services" … "making systems engineering available to more of the workforce." One of the key enablers is the deployment of DON GPT, a certified, secure, internal generative AI tool providing assured, repeatable assistance to users. The Department has been recognized with Forrester's 2025 Technology Strategy Impact Award for North America https://www.forrester.com/blogs/forresters-2025-technology-strategy-impact-award-winner-and-finalists-for-north-america/.

We thank theme editor Rick Dove and the authors of the theme articles addressing agile systems, a future of systems engineering (FuSE) project for realizing the *Systems Engineering Vision 2035*. This August *INSIGHT* issue themed on agile systems is preceded by the June 2018 *INSIGHT* featuring *enabling and practicing systems engineering agility*. The June 2023 *INSIGHT* features *agility in the future of systems* engineering that references the roadmap for near-term improvement presented at the INCOSE 2021 International Symposium, introducing synergistic linkages among nine strategic foundation concepts and four objectives. The articles in this August *INSIGHT* are as follows:

1. Rick Dove and Mike Yokell, "If You Know OODA, Like Boyd Knew OODA …."
2. Larri Ann Rosser, "Leveraging Product Line Engineering for Agile Delivery of Cyber-Physical Systems"
3. Rick Dove and Mike Yokell, "Agility as an Emergent Behavior"
4. Larri Ann Rosser, "Dynamic Decision Custody in Uncertainty"
5. Matthew Hause, "MBSE Enablers for Agile Systems Engineering–Reuse, Modular Systems & Product Lines"
6. Kenneth Kubo, "Wayfinding Through Change"
7. Shreya Sridhar, "Decomposing Complexity: A Systems Engineering View of Modern Software Aligning the V-Model with a Slicing-Based Decomposition Strategy"
8. Robin Yeman, "Agile Systems Engineering in the Age of AI: Architecting for Dynamic and Uncertain Environments".

We finish with the third article on functional-outcomes-driven tailoring (FODT) by Barend Botha titled "Implementing Functional-Outcomes-Driven Tailoring (FODT): A Strategic Roadmap for Multi-Organizational Product Development." Barend's initial article "Functional-Outcomes-Driven Tailoring in Modern Complex Engineered System Development" appears in the February 2026 *INSIGHT* and the second article "Proving the Path: Validating Functional-Outcomes-Driven Tailoring (FODT) Through Practical Applications" appears in the April 2026 *INSIGHT*.

We thank the contributing authors. We hope you find *INSIGHT*, the practitioners' magazine for systems engineers, informative and relevant. ■

From Silos to Services: Systems Engineering for the Digital Enterprise

Justin Fanelli, Chief Technology Officer, U.S. Department of the Navy

THE MANDATE FOR CHANGE

Systems engineering earned its place in the Department of War the hard way. In the early Cold War, what had been a working set of wartime practices for managing radar, aircraft, missiles, nuclear forces, and command-and-control systems hardened into a recognized discipline. That discipline runs through each of the three major military offsets: the complexity of nuclear deterrence, the networking of precision strike, and the interoperability now demanded by AI, autonomy, and distributed operations.

Through the first two offsets, exquisite systems led the race: custom built, inside a program, and often decades in the making. That model won its era. It will not win this one by itself.

The constraints have changed. The Department of the Navy still needs exquisite platforms, but it also needs a delivery system that can absorb new technology, data, and mission needs faster. Speed is now more decisive than ever. Tens of thousands of companies produce capabilities relevant to national security. Autonomy, commercial cloud, cyber, communications, data, AI, sensors, and edge technologies move on cycles measured in months, sometimes weeks. Interfacing with new data and new capabilities can no longer wait for the next technical exchange meeting.

This is not only a naval problem. It is a systems engineering problem. Large enterprises in defense, industry, health care, energy, transportation, and government all face the same pattern: many capable parts, many responsible organizations, many local optimizations, and too few common rules for turning those parts into coherent services. The result is predictable: prolonged deployment cycles, vendor lock-in, duplicated engineering, high sustainment costs, and fragile integration at the seams.

So the uncomfortable question was simple: how does an institution this large move at the speed of the problem?

MODERN SERVICE DELIVERY: ENGINEERING THE DELIVERY SYSTEM

The Department of the Navy's answer has been Modern Service Delivery, or MSD. MSD is not a slogan for IT modernization. It lives in our Innovation Adoption Kit as a repeatable systems engineering approach for moving from program-centric silos to secure, composable services.

The idea was not handed down from an ivory tower strategy shop. It was built through trial and error, with development and delivery working hand in hand with Sailors, Marines, and vendors, new and old. Early versions matured inside program offices before becoming an enterprise-wide design. Its execution was recognized when the Department of the Navy received Forrester's 2025 Technology Strategy Impact Award for North America for speed and scale in deploying secure, mission-critical technologies. Finalists included Verizon and CBRE, and it was the first time a government organization received the award.

For the INCOSE community, the most interesting part is that MSD treats the enterprise delivery apparatus itself as a system. It defines shared interfaces, reusable services, security constraints, data expectations, automation patterns, and governance mechanisms so that new needs do not restart the engineering conversation from zero every time.

The premise is deliberately simple. Before a team builds a new capability, it should be able to answer questions in a common language: What are we currently using to accomplish this objective? Can something be reused or divested? What security controls are required by default? What must operate in the cloud, at the edge, or both? How will we show that the service is ready, secure, supportable, and valuable?

When every stakeholder signs up to that common approach, the enterprise has fewer basic conversations and better mission conversations. The debate moves from 'What architecture are we inventing?' to 'How do we apply the agreed pattern to this mission need?' To each individual team, that shift may sound modest. At enterprise scale, it fuels speed.

Table 1. Five principles, stated as engineering commitments

MSD principle	Explained
Composable by Default	Architectures must be built and consumed as collections of reusable, loosely coupled services connected via standardized application program interfaces (APIs). This modular approach eliminates duplicative engineering across programs. Instead of every new program building its own distinct database or mapping software from scratch, the enterprise can assemble new capabilities rapidly by pulling from a catalog of trusted services. This treats software as modular building blocks, enabling rapid scaling as technology evolves.
Zero Trust Always	Operating in a persistently hostile digital environment requires abandoning the outdated perimeter defense mentality and adopting an "assume breach" mindset. Security is engineered into every layer of the architecture from day one. An example is the Navy's Flank Speed cloud platform, which has successfully met 151 out of 152 Zero Trust controls (3 years ahead of deadline) while maintaining over 99.9% platform availability. This involves continuous authentication, micro-segmentation, and least-privilege access, moving defenses away from traditional networks directly to the data and users to contain threats and ensure compromised components do not lead to total system failure.
Cloud-Native and Edge-Ready	Naval operations uniquely demand resilience in disconnected, intermittent, and low-bandwidth environments. A ship at sea cannot rely on constant reachback to a centralized data center. Therefore, systems must be cloud-native from inception. The scaling of the Flank Speed platform to over 500,000 users, which has previously returned over 120 years of manpower to the fleet through productivity gains and a 15% reduction in latency, demonstrates the power of a central cloud. This architecture also ensures critical compute and data capabilities can seamlessly push to the tactical edge. When a vessel loses satellite connectivity, its systems must operate independently, synchronizing with the broader cloud once restored.
Automate Everything by Default	To increase operational tempo and reduce cognitive burden, manual processes are systematically engineered out of deployment and operational pipelines. For example, the USS Fitzgerald recently used an AI-driven system to preemptively replace a degrading part, contributing to a 15% improvement in ship availability. By making Continuous Integration and Continuous Deployment (CI/CD) pipelines and automated testing mandatory, this approach shifts human effort away from routine system administration toward strategic problem-solving and warfighting tasks.
Prefer Commercial Solutions	The Department is adopting a "Buy Before Build" approach. The Navy prioritizes Commercial Off-The-Shelf (COTS) and Software-as-a-Service solutions, wherever possible. Recently, a Marine Corps communications squadron successfully integrated non-military, COTS sensors for maritime domain awareness during a major exercise, proving again commercial tech can be used in tandem with military systems. And even more recently, early this summer, two downed Apache pilots were rescued by a USV for the first time in US military history. This occurred in the dark of night and the vehicle was COTS. By leveraging the billions commercial industry invests in R&D, the Navy accelerates time-to-value, reduces maintenance overhead, and ensures systems benefit from rapid commercial innovation.

SCALING SYSTEMS ENGINEERING BEYOND ENGINEERS

One of the least obvious benefits of MSD is that it makes systems engineering available to more of the workforce. A large enterprise increasingly struggles to rely on a small number of credentialed engineers to translate every interface, dependency, constraint, risk, and trade-off. Program managers, contracting officers, cyber authorizing officials, operators, vendors, financial managers, and mission owners all shape the system whether they call themselves systems engineers or not.

MSD gives that broader workforce a shared engineering language. Services, interfaces, data products, identity, evidence, exceptions, and reuse become common terms of delivery, not specialist jargon. The goal is not to turn everyone into a systems engineer. The goal is to let everyone participate in engineering the enterprise with enough common language to reduce translation loss and increase coordinated action.

That matters because most delays in a large enterprise are not caused by one

team failing to work hard. They are caused by many teams working from different assumptions. MSD scales the useful instincts of systems engineering: define the interfaces, make constraints explicit, verify with evidence, manage exceptions, and learn back into the baseline.

FIVE PRINCIPLES, STATED AS ENGINEERING COMMITMENTS

At the heart of the Department of the Navy's MSD transformation are five engineering commitments. They are simple enough to be understood across the enterprise and rigorous enough to guide architecture, acquisition, cyber authorization, operations, and sustainment (Table 1).

FROM PRINCIPLES TO EXECUTION: FLANK SPEED WIRELESS AND EDGE

Principles matter only when they change delivery. Flank Speed Wireless and Edge deployments illustrate the practical value of MSD.

Under a traditional program-by-program model, these types of deployments can trigger a long chain of separate unique negotiations: identity, endpoint configuration, network access, cloud connectivity, cyber controls, data flows, authority to operate, operations monitoring, and sustainment. Each conversation is reasonable in isolation. Together, they become latency. Teams spend too much time re-litigating the baseline and too little time adapting the baseline to the mission.

MSD changed the starting point. Because Flank Speed established common enterprise patterns for identity, collaboration, security controls, service management, and user experience, new wireless and edge efforts could inherit more of the operating model instead of recreating it. The question became less, 'What stack are we building?' and more, 'How does this deployment conform to the common service pattern, and where does the mission require a justified exception?'

The same logic applies at the edge. Edge computing is not simply cloud placed farther forward. It is a system-of-systems design problem involving compute, data, applications, identity, synchronization, degraded operations, telemetry, cyber defense, and human use. A cloud-native and edge-ready baseline reduces the number of bespoke design decisions each deployment must make. It also gives engineers a common vocabulary for the trade space: what must run locally, what can synchronize later, what data must be cached, what controls must persist when disconnected, and what evidence proves the system remains trustworthy.

MSD focuses engineering judgment on the hard problems. By standardizing the repeatable portions of delivery, it frees engineers to focus judgment where it matters most: mission context, risk, integration, and performance under real operating conditions.

WHY THIS MATTERS TO OUR INCOSE TEAMMATES

Systems engineers have always worked at the seams. MSD makes those seams explicit and gives the rest of the enterprise a way to see them. It asks the professional systems engineer to look beyond the closed system and architect the larger ecosystem: services, users, suppliers, data, networks, authorities, policies, and operating environments.

That shift matters because modern capability is increasingly assembled rather than invented from scratch. A warfighter, shipyard, hospital, factory, or energy operator may need a new outcome long before any one program can build a complete bespoke system. The enterprise has to compose, secure, verify, and field what already exists, often across organizational boundaries.

This is where systems engineering becomes more important, not less, and where INCOSE has a particularly important role to play. The discipline provides the methods for managing complexity without freezing the enterprise in place: architecture views, interface definitions, requirements discipline, verification evidence, risk management, configuration control, and lifecycle thinking. MSD translates enough of those methods into simple operating rules that more people can use every day.

The larger lesson is this: new game-changing technologies will continue to arrive, but smart adoption is the bottleneck. The institutions that win the next decade will not simply be the ones that invent or invest. They will be the ones that find, prioritize, and adapt the best of what is already out there, engineer it into coherent services, and adopt it faster than the problem changes. ■

ABOUT THE AUTHOR

Justin Fanelli is the Chief Technology Officer for the U.S. Department of the Navy, serves on the U.S. federal Technology Modernization Fund Board, is a fellow at the Open Forum for Artificial Intelligence, and teaches Software-Defined Warfare at Georgetown University. He works with the best team in the world for the best warfighters in the world. He holds a bachelor's degree in electrical engineering from Pennsylvania State University and a dual master's degree in electrical and systems engineering from the University of Pennsylvania.

Modern Service Delivery and the Innovation Adoption Kit can be found on the Department of the Navy CIO website, doncio.navy.mil, with periodic updates on the DON CIO LinkedIn page.

AUGUST 2026
VOLUME 29/ISSUE 4

If You Know OODA, Like Boyd Knew OODA …

Rick Dove, dove@parshift.com; and **Mike Yokell,** mike.r.yokell@gmail.com

■ **ABSTRACT**
This article examines the OODA loop as modeled by John Boyd, how it relates to eight strategic aspects of systems engineering agility, and how that perspective reveals points of leverage for amplifying desired capabilities and outcomes.

INTRODUCTION

Although Boyd never drew it, a popular representation of OODA is a loop (Figure 1), depicting it as a circular sequence of four parts – *Observe, Orient, Decide, Act*. That literal representation can provide value in certain contexts but also trivializes and misrepresents the depth of Boyd's intent (Richards 2012, 2020).

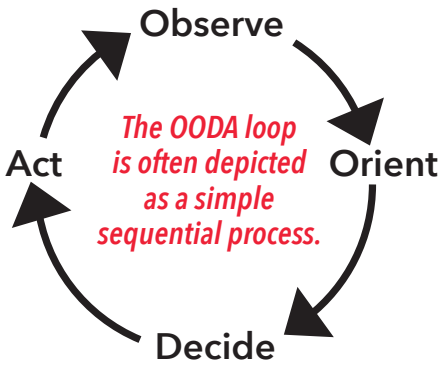


Figure 1. Mythinformation – Popular incorrect depiction (Richards 2020)

The only OODA diagram that John Boyd ever drew (Figure 2) is considerably different and richer than the ubiquitous simple circle. OODA is not about aircraft dogfights, engaging in battles, or competing in markets – though it has been productively applied to those activities. Instead, OODA is a fundamental reference architecture for how all cognitive entities confront dynamic, uncertain situations.

Boyde spoke of OODA for about 15 years (Spinney 2020) before he finally depicted it (Figure 2) as a comprehensive thoughtful diagram. He didn't invent OODA, he simply came to understand it well enough to articulate it as a coherent structured behavior. Everybody engages OODA when confronted with dynamic, uncertain situations … to variable degrees of effectiveness.

Anybody who has come to know of John Boyd's work had a first-time exposure, and varying degrees of subsequent exposure to knowledgeable source material. For those who may be reading this article in that light a few words are in order. Boyd's unique perspective on aircraft maneuverability and warfare in general has had a profound world-wide effect on aircraft design and warfare strategy (Coram 2002). Boyd's original pre-PowerPoint overhead transparency lectures and recordings chronical his research and depth of learning. Chuck Spinney and Chet Richards, who worked closely with Boyd throughout his creative period, continue clarifying and exposing Boyd's thinking since his death in 1997.

ARTICULATING THE DEPICTION OF OODA

Referring to Figure 2, *Orient* consists of beliefs, knowledge, and reasoning – the mind of a cognitive entity, be it an individual, a mission-driven team, or a vision-driven team-of-teams. It is where everything known and everything contemplated occurs. Genetic heritage, an element within *Orient*, speaks biologically to individuals and collectively to organizational hard-wired policies and infrastructures. Analysis and synthesis, the reasoning and creative part of *Orient*, is where thinking and planning happen, where the implications of what's been observed occur, and where alternatives for decisions are developed. Decisions resolve dilemmas that have alternative ways of being dealt with. It is *Orient*'s reasoning, the analysis and synthesis, that develops these alternatives. Orientation is central to OODA – it is the driver of all observation, action, and decision, and the continuous learning mechanism of capability and knowledge evolution. Positioning *Orient* as a sequential activity in the simple-circle depiction of OODA as a loop is the biggest disservice of that Figure 1 depiction (Spinney 2020).

The horizontal lines at the top of Figure 2 that exit right and left from *Orient* are labeled as Implicit Guidance and Control (IG&C). These are automatic control invocations of instinctive actions. To the right, IG&C bypasses *Decide*, invoking an action from a repertoire of if-then action patterns that have a history of success. To the left,

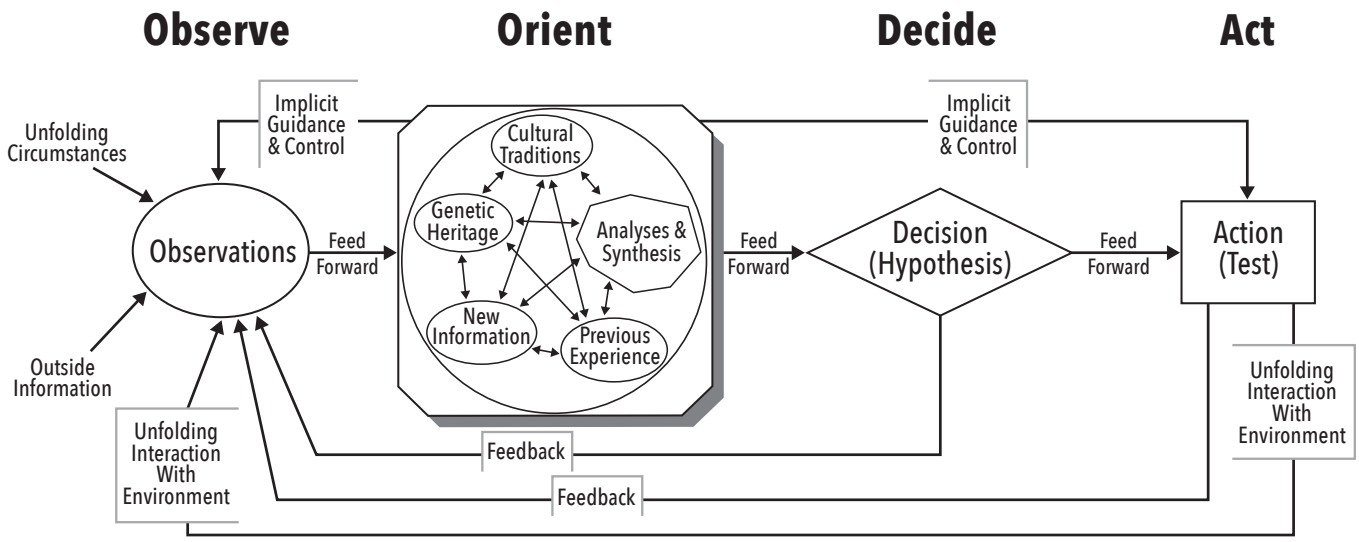


Figure 2. The only OODA diagram that Boyd actually drew (Richards 2020)

IG&C biases and filters what is observed and observable. You may have seen the video of a group passing a basketball and you are asked to count the number of times a ball is caught (Simons and Chabris 1999). Your selective attention is on the counting, and you don't see the gorilla-suited person who walks among them. This is what IG&C means in *Observe*. It is all those filtration mechanisms that inhibit the ability to see what's really going on, constrain what to look for, and what to sense, especially when something is unusual.

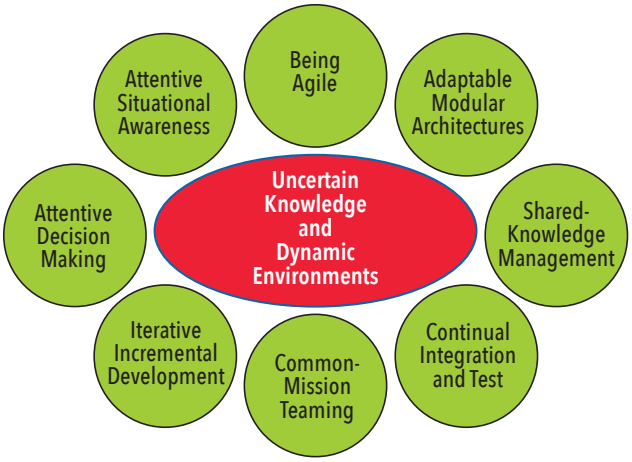
Decision is choice and timing, not analysis and synthesis. All that happens in *Decide* is selection among alternatives, and timing of invocation. Boyd marked *Decide* as a hypothesis, yet to be proven in Act as being effective. According to Richards (2020), "Most actions – our repertoire – flow from orientation (no decision necessary). … Learning manifests in the orientation area, where a repertoire of knowledge evolves."

Boyd qualified *Act* as a test, an experiment in need of validation with feedback into *Observe*. Richards says most actions are repertoire, basically a collection of decisions that have been tested and proven to be effective as situationally triggered things to do.

Observation is the perception part of situational awareness, which is classically described in three parts: perception, comprehension, and projection (Endsley 1995). *Observe* handles perception while *Orient*'s Analysis and Synthesis handle comprehension of what that means and projection into the future.

Summarizing the labeled areas in Figure 2:

- *Orient* consists of knowledge, beliefs, and reasoning.
- Within *Orient*, Analysis & Synthesis is where reasoning happens – providing alternatives for *Decide* and implications of *Observe*.
- IG&C (right side) is an if-then action link, no decision involved.
- IG&C (left side) filters (bias, expectations) on what is observable.
- *Observe* is the situational awareness perception (input) to *Orient*'s comprehension and projection reasoning.
- *Decide* is choice and timing (not analysis and synthesis) as a proposition in need of validation.
- *Act* is a Test – an experiment in need of evaluation and validation.
- Feedback connections to *Observe* are ubiquitous (and subject to Orient's IG&C filters).



Aspect	Behavior
Adaptable Modular Architectures	Compose and reconfigure configurations from evolving variations of reuseable assets
Iterative Incremental Development	Perform incremental loops of building, evaluating, correcting, and improving capabilities
Attentive Situational Awareness	Actively monitor and evaluate relevant internal and external environment factors
Attentive Decision Making	Systemically link situational awareness to decisive action
Common-Mission Teaming	Engage collaboration, cooperation, and teaming among all relevant stakeholders
Shared-Knowledge Management	Facilitate communication, collaboration, and knowledge curation
Continual Integration and Test	Demonstrate integrated test of work-in-process
Being Agile	Nurture sense-respond-evolve engagement with a dynamic environment

Figure 3. Eight strategic aspects that enable and facilitate systems engineering agility

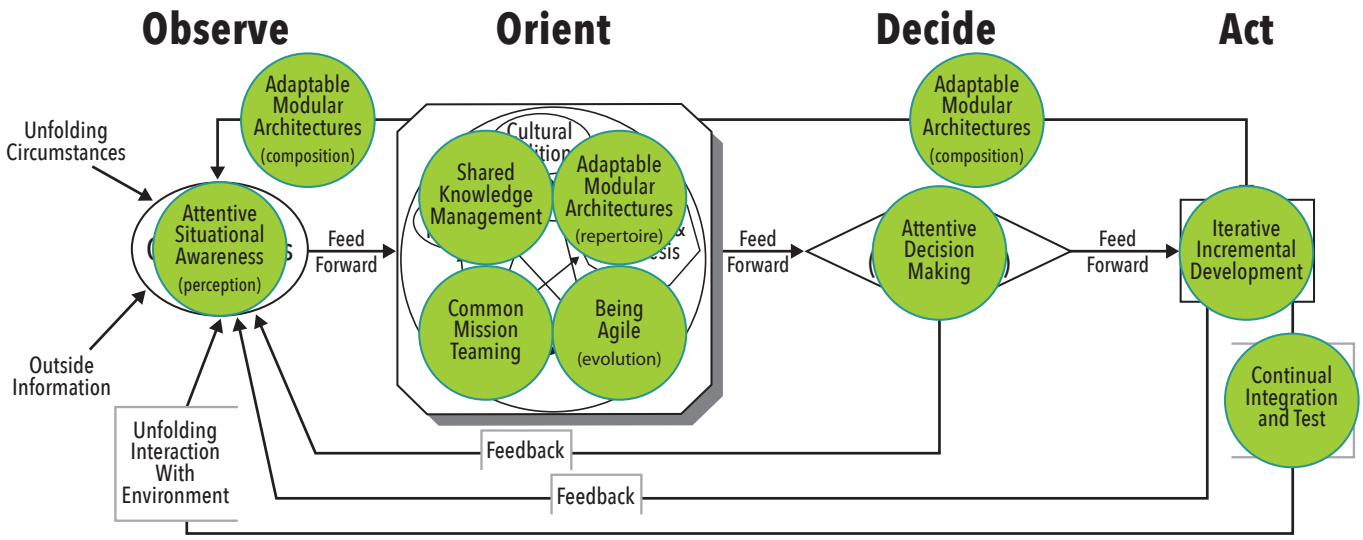


Figure 4. Overlaying systems engineering agility aspects for OODA role perspective

SEGUE TO SYSTEMS ENGINEERING

Agile systems engineering (SE) is a strategy-based method for designing, building, sustaining, and evolving purpose-fulfilling creations when knowledge is uncertain and environments are dynamic (INCOSE 2023; ISO/IEC/IEEE 2025). Ergo, OODA is also a reference architecture for systems engineering agility.

OODA should not be thought of as a best practice or a choice among multiple alternatives – it is simply the nature of cognitive interaction/response shaped by a dynamic, uncertain situation. OODA is independent of the effectiveness and outcome of that behavior. Thus, when systems engineering engages with a dynamic, uncertain operating environment it will behave in OODA fashion, with or without explicit intent. Explicit intent is exhibited when systems engineering employs procedures and strategies purposely chosen for agility. In any event, OODA is necessary, but not sufficient, for agility – purpose and performance matter.

STRATEGIC ASPECTS – PERSPECTIVES ON PURPOSE AND PERFORMANCE

Eight strategic aspects (Figure 3) enable and facilitate systems engineering agility (Dove et al. 2023, SEBoK 2025, ISO/IEC/IEEE 2025). OODA and these eight strategic aspects are two compatible but different perspectives on how an engineering activity can effectively navigate in a dynamic, uncertain operational environment.

This section overlays those eight aspects on the OODA architecture in locations that can be interpreted as having similar functional objectives. The objective in doing this is to examine the aspects from an OODA-purpose perspective as a means to reveal potential performance leverage.

The OODA reference architecture provides a purpose schema for thinking about aspect performance. The objective is to posit and consider criteria for evaluating and evolving the implementation and practice of each aspect, and the interplay of aspects that might have harmonious or discordant relationships.

To alleviate possible anxiety over the order of the aspect discussion that follows, as it neither follows the graphic circular sequence nor the top-to-bottom table sequence in Figure 3, it is noted that there is no hierarchical or sequential order of importance or priority among the eight aspects. The are simply discussed here in an order that feels natural to describing their roles in the OODA architecture.

OODA is structured to evolve *Orient*'s orchestration effectiveness at outmaneuvering confrontational dynamics. But it doesn't do that automatically. The structure enables adaptation — adaptation beyond simply dealing with the situation of the moment. Adaptation to adapt, learning what works and doesn't (Figure 4).

What follows is a four-element summary of systems engineering agility in the OODA perspective – with the agility leverage element offering one conceptual example of how the OODA perspective might influence strategy for agility performance improvements.

Common-mission teaming:

- OODA purpose: provide the context foundation of *Orient*.
- Agility behavior: engage collaboration, cooperation, and teaming among all relevant stakeholders.
- OODA performance: single-minded foundation of what is and is not the objective.
- Agility leverage: appreciate the mission as an outcome rather than a set of tactics to carry out.

Shared knowledge management:

- OODA purpose: provide the content foundation of *Orient*.
- Agility behavior: facilitate communication, collaboration, and knowledge curation.
- OODA performance: single-minded knowledge foundation that evolves at the speed of *Act*.
- Agility leverage: add to, and prune, repertoires of situational responses and awareness sensors.

Attentive situational awareness:

- OODA purpose: inform *Orient* with observations.
- Agility behavior: actively monitor and evaluate relevant internal and external environment factors.
- OODA performance: look for known interests, anomalous developments, and effects of actions.
- Agility leverage: monitor actions for continued effectiveness rather than just perpetuating what worked before – situational dynamics may render a previous action ineffective.

Adaptable modular architecture:

- OODA purpose: develop and evolve IG&C repertoires of reusable *Observe* sensors and if-then actions.
- Agility behavior: compose and reconfigure configurations from evolving variations of reusable assets.
- OODA performance: self-evolve repertoires by automatically remembering things that work and pruning things that no longer work. Repertoires self-select IG&C compositions.

- Agility leverage: validate process methods for appropriateness and effect every time they are employed.

Attentive decision making:
- OODA purpose: *Decide* (hypothesis).
- Agility behavior: systemically link situational awareness to decisive action.
- OODA performance: instantiate (initially) and validate (continually) IG&C actions.
- Agility leverage: evaluate decision actions for appropriate timing (not for decision effectiveness).

Iterative incremental development:
- OODA purpose: *Act* (test).
- Agility behavior: perform incremental loops of building, evaluating, correcting, and improving capabilities.
- OODA performance: execute a decided action as a test for potential sensor and *Act* repertoire content.
- Agility leverage: accelerate learning and improvement by innovative experiments doing something different or differently with promise.

Continual integration and test:
- OODA purpose: guide unfolding interaction with environment.
- Agility behavior: demonstrate integrated test of work-in-process.
- OODA performance: deploy actions into a confrontation with the operational environment.
- Agility leverage: employ short cycle continuous integration/continuous deployment (CI/CD) with decisive feedback.

Being agile:
- OODA purpose: orchestrate resilience and evolution of *Orient* foundation.
- Agility behavior: nurture sense-respond-evolve engagement with a dynamic environment.
- OODA performance: validate and evolve OODA capability.
- Agility leverage: monitor, sustain, and improve agility that emerges from Aspect interaction.

CONCLUDING REMARKS

OODA is focused on evolving *Orient*'s orchestration effectiveness when practiced proactively, purposefully. It constantly evolves practitioner ability to be better at engaging with dynamic, uncertain situations. It is a learning machine preparing for perpetual confrontation. Being agile, the strategic aspect shown in the Figure 3 *Orient* block, is about learning and evolving as the principle means for growing agility performance excellence.

This article explored the eight strategic aspects, as a system, from an OODA perspective — a systems thinking methodology for understanding systems as relationships and behaviors. Two knowledge effects emerged:

1. Looking at an aspect as fulfilling an OODA purpose provides a specific perspective on how to evaluate, and perhaps adjust, performance to purpose.
2. Looking at aspect in the roles of OODA architectural elements reveals direct aspect-to-aspect relationship connections, identifying and encouraging opportunities for synergy and avoidance of discord among aspects. A topic for a separate article (Dove and Yokel 2026).

A variety of other perspectives on the eight aspects are being and have been explored (Dove 2025), offering a richer appreciation for each of the aspects and how they might work together more effectively toward systems engineering agility.

To conclude, if you know OODA, like Boyd knew OODA, it would be intuitive nature behind all you do. ■

REFERENCES

- Coram, R. 2002. *Boyd–The Fighter Pilot Who Changed the Art of War*. Little, Brown and Company.
- Dove, R., K. Lunney, M. Orosz, and M. Yokell. 2023. Agile Systems Engineering – Eight Core Aspects. www.researchgate.net/publication/373092973.
- Dove, R. 2025. "Agile Systems Thinking – Outside the Box." NDIA Systems and Mission Engineering Conference, Tampa, US-FL, 27-30 October. www.researchgate.net/publication/402947453.
- Dove, R. , and M. Yokell. 2026. "Agility as an Emergent Behavior." *INSIGHT 29 (4)*, International Council on Systems Engineering, August.
- Endsley. M. R. 1995. "Toward a Theory of Situation Awareness in Dynamic Systems." *Human Factors* 37(1): 32-64. March. www.researchgate.net/publication/210198492.
- INCOSE. 2023. *Systems Engineering Handbook: A Guide for System Life Cycle Process and Activities* (5th ed.). Section 4.2.2, Agile Systems Engineering. D. D. Walden, T. M. Shortell, G. J. Roedler, B. A. Delicado, O. Mornas, Y. S. Yip, and D. Endler (Eds.). San Diego, US-CA: International Council on Systems Engineering. Published by John Wiley & Sons, Inc.
- ISO/IEC/IEEE. 2025. 24748-10:2026, Guidelines for Systems Engineering Agility. February. www.iso.org/standard/90086.html.
- Richards, C. 2012. Boyd's OODA Loop (It's Not What You Think) https://slightlyeastofnew.com/wp-content/uploads/2010/03/boyds-real-ooda-loop.pdf.
- Richards, C. 2020. "Boyd's OODA Loop." *Necesse* 2020, 5 (1): 142-165. www.scribd.com/document/734847756/Boyds-OODA-Loop-Necesse-vol-5-nr-1.
- SEBoK. 2025. Agile Systems Engineering. SEBoK v. 2.13, released 17 November.
- Simons, D. J. , and C. Chabris, C. 1999. "Gorillas in Our Midst: Sustained Inattentional Blindness for Dynamic Events." *Perception* 28 (9):1059-1074. www.chabris.com/Simons1999.pdf.
- Spinney, F. C. 2020. Evolutionary Epistemology – A Personal View of John Boyd's "Destruction and Creation" and its centrality to the OODA Loop. www.youtube.com/watch?v=gdK4y6O-llE.

ABOUT THE AUTHORS

Rick Dove is an unaffiliated researcher and systems engineer focused in the systems agility and systems security areas. He is an INCOSE Fellow and chairs the working groups for Agile Systems and Systems Engineering and for Systems Security Engineering. He leads the Agility and Security project areas for INCOSE's future of systems engineering (FuSE) initiative.

Dr. Mike Yokell is a retired systems engineering leader with more than 40 years in the US aerospace and defense industry. He is the editor, co-editor, or contributor to numerous international standards on systems and software engineering. Mike is certified as an expert systems engineering professional (ESEP). He holds multiple US and European patents for model-based systems engineering and large-scale immersive virtual reality.

Leveraging Product Line Engineering for Agile Delivery of Cyber-Physical Systems

Larri Ann Rosser, Larri.Rosser@rtx.com

■ **ABSTRACT**
Agile methods have produced significant gains in productivity and delivery speed for software-intensive systems, yet their application to complex cyber-physical systems has not yielded comparable results. This gap reflects not merely cultural resistance, but fundamental structural constraints associated with hardware development, manufacturing, long-lead components, integration, certification, and system-level test. Attempts to optimize for software flow alone therefore fail to improve overall value delivery.

This article suggests that product line engineering (PLE) provides a structural mechanism for enabling meaningful agility in complex cyber-physical systems by organizing solutions into stable core assets and controlled variability that explicitly respects physical and organizational constraints. By aligning product line scoping decisions with system bottlenecks and delivery objectives, organizations can preserve throughput while enabling rapid adaptation and learning.

The article presents key challenges encountered when applying agile practices to cyber-physical systems, outlines PLE principles tailored for such environments, and introduces practical alignment patterns for integrating PLE with agile systems engineering. Illustrative practitioner examples are provided to demonstrate how these patterns support faster, more reliable delivery without sacrificing system integrity. The result is a pragmatic framework for achieving cyber-physical agility grounded in architectural discipline rather than software-centric optimization.

■ **KEYWORDS:** Agile, Hardware, Cyber-Physical, Product Lines, CCPS, Agile SE, PLE

STRUCTURAL LIMITS OF AGILE IN COMPLEX CYBER-PHYSICAL SYSTEMS

Agile methods were originally developed to address the dominant constraints of software development: requirements volatility, knowledge uncertainty, and the high cost of late defect discovery. Iterative delivery, frequent feedback, and incremental refinement allow teams to manage these forces effectively when system components are largely fungible, reproducible, and decoupled from physical production. In complex cyber-physical systems (CCPS), however, these underlying assumptions do not hold.

It is important to note that the term product line engineering (PLE) encompasses a family of strategies, not a single technique. While feature-based product lines dominate much of the software literature, cyber-physical systems frequently benefit more from layer-based, modular, and platform-oriented product line strategies. These approaches structure variation at coarser levels of granularity, reducing churn in hardware, manufacturing, integration, and certification activities. In complex cyber physical system (CCPS) environments, such strategies often provide stronger alignment with physical and organizational constraints than fine-grained feature-based approaches alone.

Cyber-physical systems are governed by structural constraints that impose hard limits on achievable delivery speed regardless of how efficiently software work is organized. These constraints include long-lead hardware components, specialized manufacturing processes, regulatory certification, supply chain dependencies, environmental qualification, and multi-level integration and test. While software changes may be implemented in days or weeks, their validation at system scale often depends on

hardware availability, physical configuration, and test infrastructure that operate on fundamentally different timelines.

As Goldratt observed in *Theory of Constraints*, improving a non-bottleneck does not improve overall system performance. In CCPS development, the system bottleneck is rarely the production of new software features. Instead, throughput is constrained by integration and test capacity, hardware readiness, certification cycles, and manufacturing cadence. Optimizing software flow without addressing these structural bottlenecks therefore produces local efficiencies that fail to translate into faster system delivery.

Organizations attempting to apply software-centric agile methods to CCPS frequently encounter a growing mismatch between planning models and physical reality. Program increments may demonstrate impressive software velocity while system-level milestones continue to slip. Backlogs grow with features that cannot be integrated or tested. Teams experience increasing rework as late hardware changes ripple through tightly coupled subsystems. The result is not improved agility but rather increased organizational stress and reduced delivery predictability.

Digital engineering techniques such as simulation, emulation, digital twins, and digital threads mitigate some of these constraints by enabling earlier verification and tighter feedback loops. However, these artifacts themselves require substantial investment to develop, validate, and sustain. They do not eliminate the underlying physical dependencies that ultimately govern system throughput.

True agility for CCPS therefore cannot be achieved through process adoption alone. It requires explicit architectural and organizational strategies that align development effort with the dominant structural constraints of the system. Without such alignment, agile practices risk accelerating the production of work that cannot yet be realized, creating the illusion of progress while system-level delivery remains bounded by unresolved bottlenecks.

PRODUCT LINE ENGINEERING AS AN AGILITY ENABLER FOR COMPLEX CYBER-PHYSICAL SYSTEMS

Product line engineering (PLE) addresses the structural limits of agile development in cyber-physical systems by explicitly organizing solutions around stability, controlled variability, and throughput preservation. Rather than treating each delivered system as a largely independent project, PLE establishes a persistent architectural foundation that supports the systematic production of multiple related system instances over time.

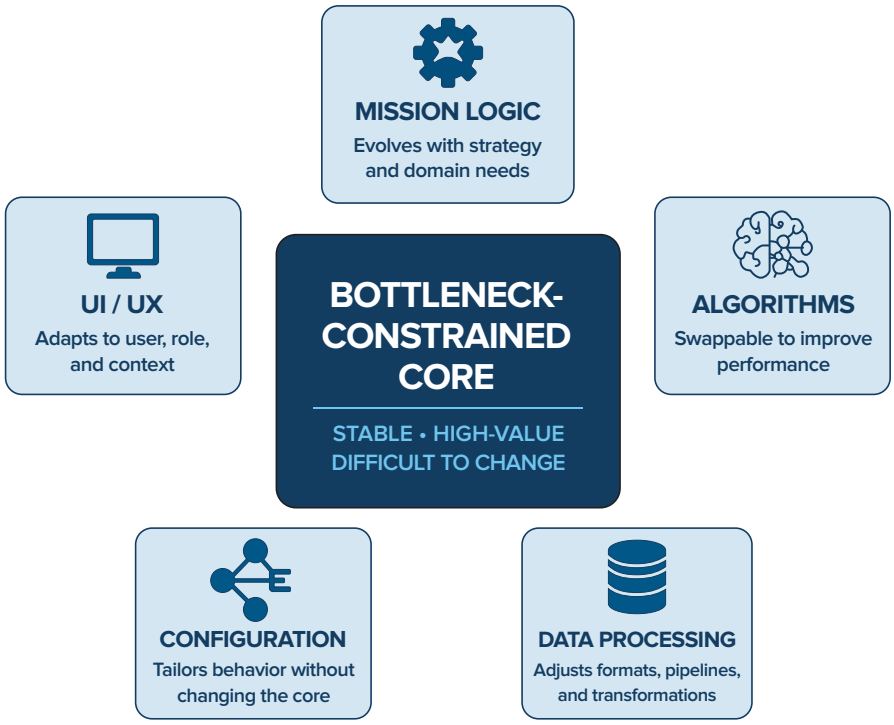


Figure 1. Engineering concentrates change in domains that can absorb it (light blue boxes) while stabilizing elements constrained by integration, certification and hardware dependencies (dark blue box)

At its core, PLE partitions a system family into two complementary domains: a stable core of assets that remain invariant across the product line, and a set of managed variation points that enable customization for specific operational needs. This separation is well understood in software product lines, where variation is typically implemented through configuration, feature selection, and conditional logic. In cyber-physical systems, however, the scoping of stability and variability must account for physical constraints, supply chain realities, and integration dependencies that dominate delivery risk and cost. In practice, this often favors modular, layered, or platform-based product line strategies over purely feature-based approaches in cyber-physical contexts.

For CCPS, the most valuable candidates for stability are not merely common functions but those elements whose change incurs disproportionate impact on integration effort, test complexity, certification scope, manufacturing disruption, or logistics burden. These may include hardware platforms (the underlying physical systems and their governing interface constraints), physical interfaces, safety-critical subsystems, environmental envelopes, mechanical tolerances, and externally governed standards. By deliberately constraining variability in these domains, PLE prevents frequent redesign of system elements that sit directly on the program's primary bottlenecks.

Conversely, PLE concentrates variability in areas where change can be absorbed with minimal impact on value delivery. These often include mission logic, control algorithms, user interfaces, data processing pipelines, and selected software-defined behaviors. When these variation points are explicitly designed and governed, teams gain the ability to respond rapidly to evolving requirements while protecting the stability of the system's physical and organizational backbone.

This architectural discipline transforms agility from a scheduling aspiration into a structural property of the system. Rather than relying on process alone to absorb volatility, PLE embeds change tolerance directly into the system design. The result is not simply faster iteration within individual teams, but improved predictability and higher sustained delivery capacity at the system level.

Crucially, PLE also enables cumulative learning. As the product line matures, reusable assets grow in capability and robustness, reducing the marginal cost of each new system instance. Over time, organizations transition from repeated reinvention to structured evolution, achieving gains in quality, speed, and confidence that no amount of process optimization can deliver in isolation.

ALIGNMENT PATTERNS FOR INTEGRATING AGILE SYSTEMS ENGINEERING AND PRODUCT LINE ENGINEERING

The successful integration of agile systems engineering with product line engineering in cyber-physical environments depends less on tooling or ceremony than on a small set of architectural alignment patterns. These patterns provide practical guidance for structuring programs so that agile practices operate within—rather than against—the dominant constraints of the system.

Pattern 1: Bottleneck-Aligned Stability

Stability should be deliberately imposed on those elements of the system that dominate delivery risk and throughput: hardware platforms, safety-critical subsystems, external interfaces, manufacturing processes, and certification artifacts. These elements form the product line's core and are evolved cautiously and infrequently.

Agile iteration is then focused on those domains where change can be absorbed without disturbing these bottlenecks—typically mission software, control logic, data processing, and user interaction layers. This preserves overall system flow while still allowing rapid response to operational needs.

Pattern 2: Variation as a First-Class Architectural Concern

Variation must be designed explicitly rather than discovered through repeated modification. Variation points should be identified early in system architecture and implemented through standardized mechanisms such as configuration parameters, modular interfaces, software-defined behaviors, and controlled hardware options.

This approach converts requirement volatility from a destabilizing force into a managed input that the system is architected to accommodate.

Pattern 3: Decoupled Development Cadences

Different parts of a cyber-physical system naturally operate on different time horizons. Hardware platforms may evolve on multi-year cycles, while mission software may change monthly or even weekly. PLE enables these disparate cadences to coexist by insulating long-cycle assets from high-frequency change.

Agile planning is thus conducted within the bounds of each domain's natural tempo rather than forcing artificial synchronization that amplifies rework and integration risk.

Pattern 4: Incremental Certification and Integration Scope Control

By constraining variability in safety-critical and regulated domains, PLE limits the scope of recertification required for each new product instance. Agile increments can therefore deliver meaningful operational value without triggering full system recertification on every release.

This pattern transforms certification from a program-stopping event into a managed pipeline activity.

Pattern 5: Asset-Centric Backlog Management

Backlogs should be organized around product line assets and variation points rather than individual projects. This enables teams to prioritize investments that strengthen the reusable core while still addressing near-term delivery needs.

Over time, this shifts organizational behavior from short-term project optimization toward long-term value delivery.

ILLUSTRATIVE EXPERIENCE FROM CYBER-PHYSICAL PROGRAMS

The following composite examples, drawn from multiple large-scale cyber-physical programs, illustrate how the preceding alignment patterns operate in practice.

In one aerospace program developing a family of mission systems, early attempts to adopt software-centric agile methods produced strong velocity at the team level but repeated delays at system integration. Each program increment introduced new software features that could not be validated until corresponding hardware configurations became available months later. Integration backlogs grew, rework accumulated, and confidence in planning steadily eroded.

The program restructured its architecture around explicit product line principles. A common hardware platform, core avionics services, and safety-critical interfaces were designated as stable assets. Change within these domains required cross-program governance and long-term planning. In contrast, mission processing, user interaction layers, and selected control behaviors were redesigned around configurable variation points. Agile teams were then aligned primarily to these high-variability domains.

Within three release cycles, system-level integration stabilized. Software increments began delivering operational value on predictable cadence, while hardware and certification activities progressed independently along longer planning horizons. The program reported reduced late-stage rework, shorter integration campaigns, and improved confidence in delivery commitments.

A similar pattern emerged in an industrial automation product line supporting multiple customer deployments. Prior to PLE adoption, each customer configuration was treated as a distinct project, leading to frequent redesign of core components and escalating test complexity. By consolidating common mechanical, electrical, and safety subsystems into a stable core and shifting customer-specific variation into software-defined behaviors and configurable modules, the organization achieved a significant reduction in manufacturing disruption and test effort per delivery while increasing responsiveness to customer requirements.

These experiences illustrate that agility in cyber-physical systems emerges not from accelerating all change, but from placing change where the system can absorb it.

IMPLICATIONS FOR AGILE SYSTEMS ENGINEERING PRACTICE

The integration of agile systems engineering with product line engineering fundamentally reframes how organizations should approach transformation in cyber-physical environments. Rather than asking how agile practices can be scaled to ever larger and more complex systems, practitioners must first ask how system structure enables or constrains agility.

This shift carries several important implications. First, architectural decisions become the primary lever for improving delivery performance. Investments in stable core assets, explicit variation mechanisms, and integration boundary design often yield greater returns than further optimization of team-level processes. Second, program governance must evolve from project-centric oversight toward product line asset-centric stewardship, ensuring that product line integrity is preserved while enabling rapid fielding of new capabilities. Third, metrics of success must extend beyond software velocity to include system throughput, integration stability, certification impact, and cumulative learning across product generations.

Most importantly, this approach restores realism to agile adoption in cyber-physical contexts. By aligning change with structural constraints, organizations can achieve genuine responsiveness without sacrificing safety, quality, or long-term sustainability.

CONCLUSIONS

Agility in complex cyber-physical systems cannot be achieved through process adoption alone. The dominant constraints that govern delivery—integration, hardware readiness, certification, manufacturing, and test—impose structural limits that software-centric agile methods cannot overcome by themselves.

Product line engineering provides the architectural foundation required to make agility real in this domain. By organizing systems around stable core assets and explicitly managed variability, PLE aligns development effort with system bottlenecks, preserves throughput, and enables sustained learning across product generations. When combined with agile systems engineering practices, this approach transforms agility from a scheduling aspiration into an inherent system property.

For organizations operating in cyber-physical domains, the path forward is therefore not to scale agile ever harder, but to architect for change intelligently. When structure and process are aligned, agility becomes not only possible, but reliable. ■

REFERENCES

- Baldwin, C. Y., and K. B. Clark. 2000. *The Power of Modularity*. MIT Press.
- Clements, P., and L.Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley.
- Clements, P., and L. Northrop. 2015. *Software Product Lines: Practices and Patterns* (2nd ed.). Addison-Wesley.
- Dove, R. 1999. *Response Ability: The Language, Structure, and Culture of the Agile Enterprise*. John Wiley & Sons.
- Dove, R. 2001. "Agile Enterprise Cornerstones: Knowledge, Values, and Response Ability." In Proceedings of the IFIP TC5/ WG5.7 International Conference on Advances in Production Management Systems (pp. 313–320).
- Dove, R., and R. LaBarge. 2014. "Architecting for Agility: Model-Based Approach." INCOSE International Symposium Proceedings, 24 (1): 281–296. https://doi.org/10.1002/j.2334-5837.2014.tb03053.x.
- Ebert, C., and J. De Man. 2008. "Effectively Utilizing Software Product Line Engineering." *IEEE Software* 25 (4): 50–58. https://doi.org/10.1109/MS.2008.79.
- Goldratt, E. M. 1990. *Theory of Constraints*. North River Press.
- Haberfellner, R., O. de Weck, E. Fricke, and S. Vössner. 2019. *Systems Engineering: Fundamentals and Applications* (2nd ed.). Springer.
- ISO/IEC/IEEE. 2015. ISO/IEC/IEEE 42010: Systems and Software Engineering — Architecture Description. IEEE.
- Johnson, S., and R. Yeman. 2023. *Industrial DevOps: Build Better Systems Faster*. IT Revolution Press.
- Meyer, M. H., and A. P. Lehnerd. 1997. *The Power of Product Platforms: Building Value and Cost Leadership*. Free Press.
- Northrop, L. M., et al. 2009. A Framework for Software Product Line Practice, Version 5.0. Software Engineering Institute, Carnegie Mellon University.
- Oppenheimer, G. M., and J. Kim. 2017. "Managing Integration and Verification Complexity in Large Cyber-Physical Systems." INCOSE International Symposium Proceedings 27 (1): 114–128.
- Pohl, K., G. Böckle, and F. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.
- Pyster, A., and D. H. Olwell. 2013. The Guide to the Systems Engineering Body of Knowledge (SEBoK). INCOSE.
- Schoen, E., and T. Menzies. 2020. "Industrial Case Studies of Software Product Line Adoption: Benefits and Barriers." *Journal of Systems and Software* 164: 110563. https://doi.org/10.1016/j.jss.2020.110563.
- Ulrich, K. T., and S. D. Eppinger. 2016. *Product Design and Development* (6th ed.). McGraw-Hill Education.

ABOUT THE AUTHOR

Larri Ann Rosser is a senior technical fellow at Raytheon. She has worked in engineering and technology for four decades as an electrical engineer, software engineer, systems engineer, and architect. She holds a B.S. in information systems and computer science from Charter Oak State College, and master of science in systems engineering from Worcester Polytechnic Institute. She holds multiple patents in the man portable systems domain and is a RTX CAP certified architect, and an advanced SAFe program consultant. She is the co-chair of the INCOSE Agile Systems and Systems Engineering Working Group, a member of the INCOSE Complex Systems Working Group, a member of the NDIA SED Architecture Committee and NDIA ADAPT Working Group. At Raytheon, she co-chairs the Raytheon Architecture Board and the RTX System & Enterprise Architecture Community of Practice. She works with programs and product lines to apply modern methods to create customer value.

Agility as an Emergent Behavior

Rick Dove, dove@parshift.com; and **Mike Yokell,** mike.r.yokell@gmail.com

ABSTRACT
In common usage, the terms agile and agility refer to the ability to move quickly and easily. In systems engineering agility is not installed or practiced; it is a behavior that emerges when eight strategic aspects engage effectively and efficiently with a dynamic situation. This article explores the nature and nurturing of emergent agility.

INTRODUCTION

Agility as a measurable quality of movement went to the dogs in 1980, when the Kennel Club of the United Kingdom recognized dog agility as a sport. Agility courses have dogs run through a set of obstacles (Figure 1) to evaluate their effectiveness, maneuverability and efficiency. Scoring for agility is based on the fastest time through the obstacles without errors. "The dog's physical ability to navigate obstacles safely and efficiently is more important than aesthetic grace (www.thetrainingofdogs.com)."

These dogs demonstrate agility as a quality of operational behavior, effectively and efficiently dealing with obstacles. In engineering, agility can be enabled by architecture and facilitated by experience, but it's manifestation is expressed as effective and efficient engagement with dynamic situations – an operational behavior.

AGILITY AS EMERGENT BEHAVIOR

Manifestation of engineering agility is shown to have eight strategic aspects (Dove, et al. 2023, SEBoK 2026), where each aspect is described as a strategy in terms of need and intent – why and what (Table 1). But as aspects of agility they are just parts of a broader concept that emerges from harmonious relationships and interactions of those Aspects in operational play (Figure 2).

An organic, archetypal example of agility emerging from the interplay of the eight aspects is revealed in "Innovation Engineering at Tesla – Agility as a Cultural Practice" (Dove et al. 2025). "At Tesla … agile engineering isn't recognized and

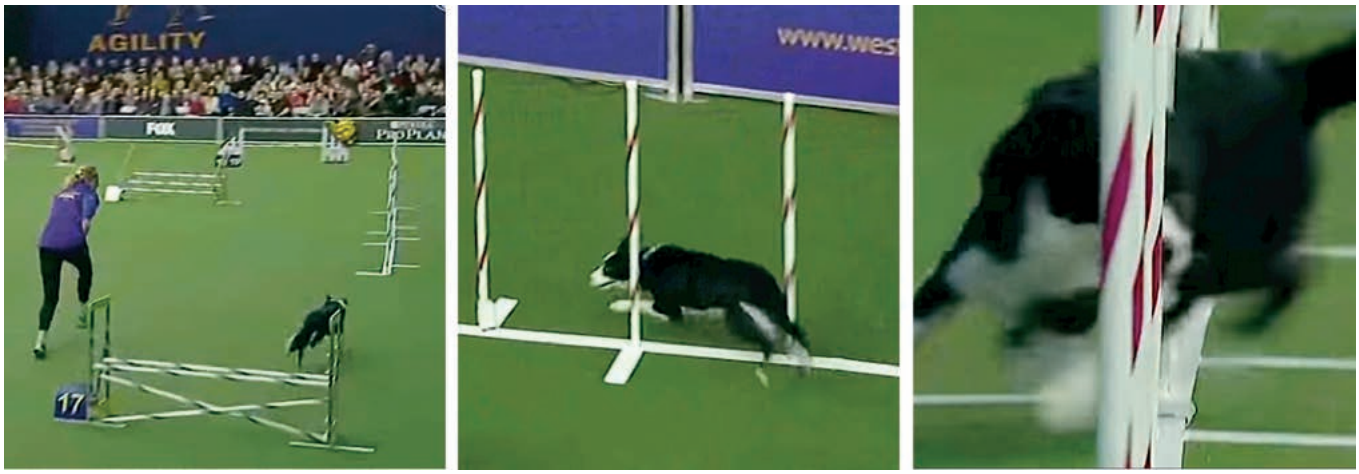


Figure 1. Border Collie weaves through pole obstacle in agility course, https://en.wikipedia.org/wiki/Dog_agility

Table 1. Engineering agility has eight strategic aspects

Aspect	Need	Behavior/Intent
Adaptable Modular Architectures (AMA)	Facilitated product and process experimentation, modification, and evolution	Compose and reconfigure configurations from evolving variations of reusable assets
Iterative Incremental Development (IID)	Minimal rework, maximal quality, facilitated innovation	Perform incremental loops of building, evaluating, correcting, and improving capabilities
Attentive Situational Awareness (ASA	Timely knowledge of emergent risks and opportunities	Actively monitor and evaluate relevant internal and external environment factors
Attentive Decision Making (ADM)	Timely corrective and improvement actions	Systemically link situational awareness to decisive action
Common-Mission Teaming (CMT)	Coherent collective pursuit of a common mission	Engage collaboration, cooperation, and teaming among all relevant stakeholders
Shared-Knowledge Management (SKM)	Accelerated mutual learning and single source of truth for internal and external stakeholders	Facilitate communication, collaboration, and knowledge curation
Continual Integration and Test (SKM)	Early revelation of system integration issues	Demonstrate integrated test of work-in-progress
Being Agile (BA)	Attentive operational response to evolving knowledge and dynamic environments	Nurture sense-respond-evolve engagement with a dynamic environment

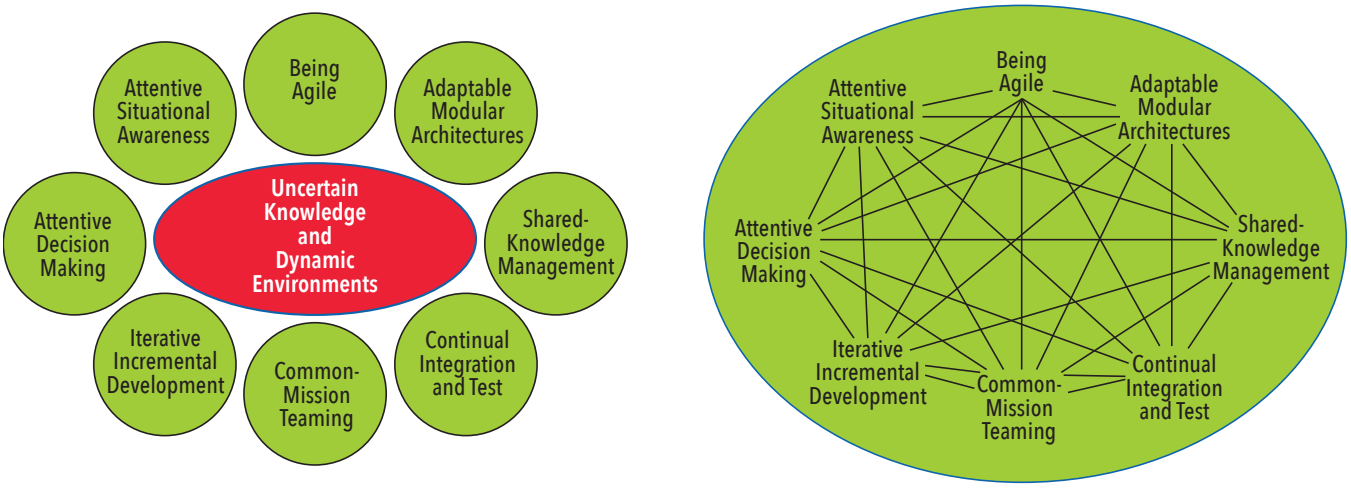


Figure 2. Agility is an emergent behavior of relationships

doesn't appear as a formal objective; the driving objective is innovation. Agility appears as an emergent characteristic from a pursuit of innovation engineering." The engineering process at Tesla nurtures that emergent agility naturally: full stack teams sensitive to efficiency sustain single-minded mission alignment, collective knowledge development, multi-perspective hands-on experimentation, test-driven validation, and encapsulated modularity as a way of being.

Another emergent-agility example predating the eight-Aspect articulation is described in a paper titled "Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group" (Dove, Schindel, and Scrapper 2016). Perhaps the title says it all. The Wave process at U.S. Navy SPAWAR System Center (SSC) Pacific is an overlapping iterative incremental evolutionary process formulated by Dahmann et al. (2011). Lead engineer Chris Scrapper implemented this process as a team- and human-focused natural adaptation of the eight Aspects – to customer critical acclaim.

Team behavior as an emergent outcome has a strong research foundation. Team behavior is definitionally emergent – good, bad, or indifferent. The concept of team cognition is particularly relevant to engineering agility as it is central to coherent efficient adaptation. "The term 'cognition' used in the team context refers to cognitive processes or activities that occur at a team level. Like the cognitive processes of individuals, the cognitive processes of teams include learning, planning, reasoning, decision making, problem solving, remembering, designing, and assessing situations" (Cooke et al. 2013). Team cognition has convergent and divergent forms (Kozlowski and Chao 2012), where convergent is a homogenous shared composition while divergent is a compiled collection of heterogenous expertise. From an agility perspective

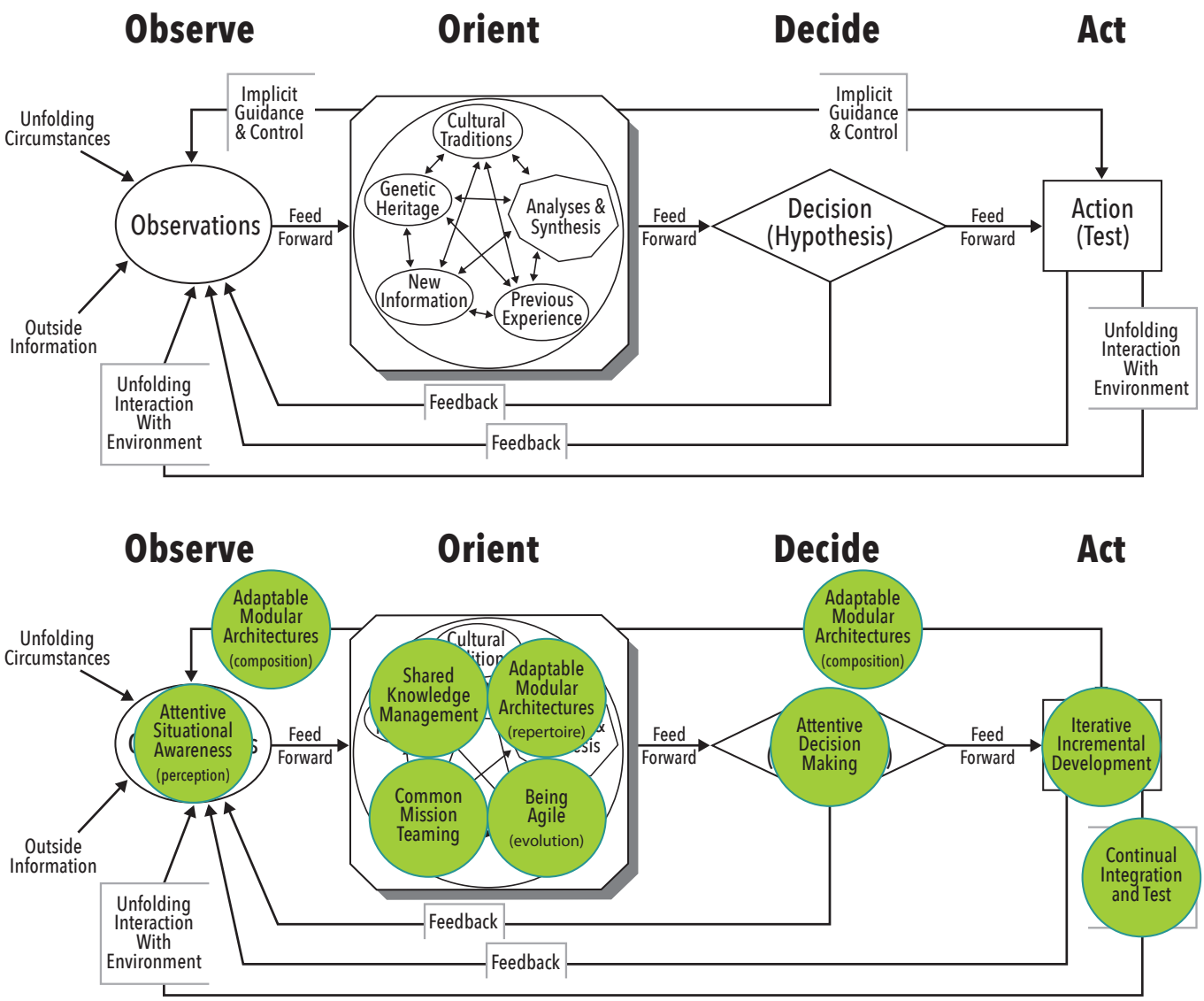


Figure 3. Overlaying aspects for an OODA role perspective reveals direct aspect-aspect relationships

an effective and efficient response to a dynamic situation may have an indistinguishable compositional aspect source or a specific aspect source well-suited to the immediate response needs.

Relationships between aspects come in three basic types: harmonious, discordant, and harmoniously discordant. Harmonious has a positive effect, discordant has a negative effect, and harmoniously discordant has a potentially beneficial balancing effect. As examples, shared knowledge management (SKM) and common mission teaming (CMT) can be harmonious when sharing multiple perspectives on mission understandings, discordant when mission knowledge is shared arbitrarily, and harmoniously discordant when need-to-know constraints determine prudent scope of sharing. These distinctions can guide the nurturing of emergent agility, sensing and mitigating discordance as well as sensing and leveraging opportunities for harmony.

CRITICAL RELATIONSHIPS

John Boyd's observe-orient-decide-act (OODA) framework is a reference architecture for how cognitive entities deal with dynamic uncertain situations, whether done purposely and to good effect, or not. As such, the OODA framework is also a reference architecture for systems engineering agility. Figure 3, from (Dove and Yokell 2026), depicts the eight aspects overlaid on the OODA framework, which illuminates an OODA perspective on aspect roles. As a side effect, this overlay identifies aspects with close direct relationships – the four that work together in the *Orient* area and those connected by a direct feed line.

No aspect among the eight is more important than any other for emergent agility, but relationships between aspects can have varying criticality. For instance, the relationship that attentive situational awareness (ASA) has with the four *Orient* aspects (SKM/CMT/AMA/BA) would seem most critical of all for relationship harmony, else *Orient* is flying blind with no sense of reality. As the orchestrator of all action, *Orient*'s four aspects would seem to be thsecond most critical set of relationships.

Systems thinking is sensemaking activity that explores the effect of relationships among entities. It builds models to represent how these entities are connected to each other, with beliefs of what those connections enable and constrain as holistic system behavior. Figure 4 isolates the edges from Figure 2 for the two sets of relationships considered in this article as most critical. How and how well the aspects relate to each other is dependent

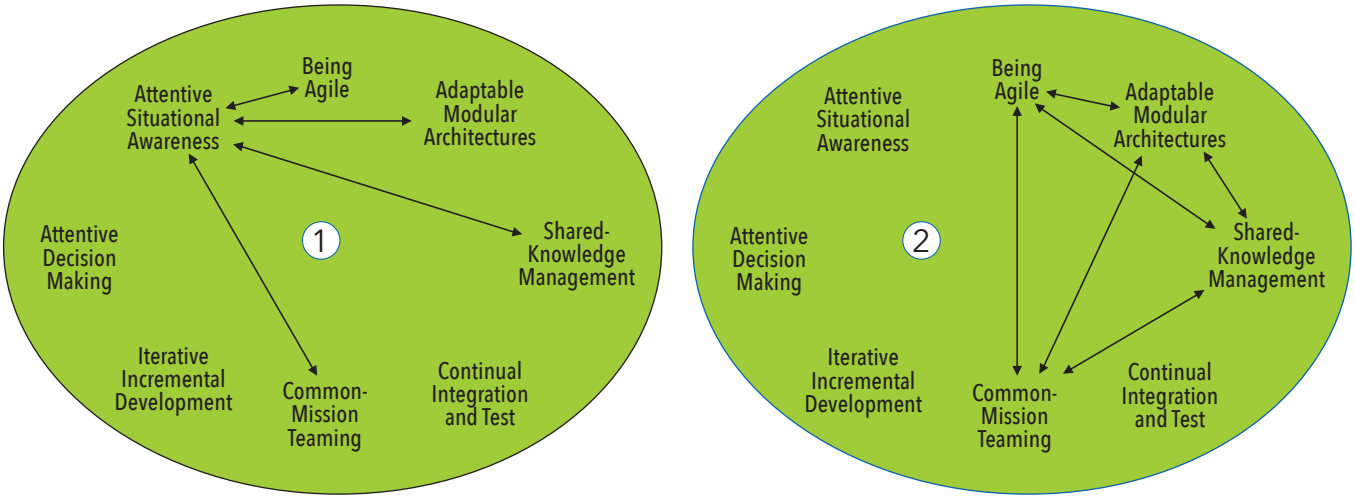


Figure 4. Bidirectional relationships

upon specific team and project context; but if agility is a valued behavior, the being agile aspect will mitigate relationship discordance and nurture relationship harmony.

IN SUMMARY

Agility is a quality of engagement with dynamic uncertain situations. That quality manifests as effectiveness and efficiency of engagement.

Agility emerges in engineering engagements from harmonious interactions among eight Strategic Aspects. Harmony in Aspect interactions can be nurtured. Nurturing leverages sensitivity to those interactions to mitigate discordance and amplify harmony, though success is not guaranteed. ■

ACKNOWLEDGEMENT

The three graphic panels in Figure 1 are stills captured from a Fox News video in the Wikipedia entry for Dog Agility, with Fox News recognized for Creative Commons attribution license.

REFERENCES

- Cooke, N. J., J. C. Gorman, C. W. Myers, and J. L. Duran. 2013. "Interactive Team Cognition." *Cognitive Science* 37: 255–285. https://onlinelibrary.wiley.com/doi/epdf/10.1111/cogs.12009.
- Dahmann, J., G. Rebovich, R. Lowry, J. Lane, and K. Baldwin. 2011. "An Implementers' View of Systems Engineering for Systems of Systems." IEEE International Systems Conference, Montreal, CA, 4-7 April. https://www.researchgate.net/publication/252011984.
- Dove, R., B. Schindel, and C. Scrapper. 2016. "Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group." INCOSE 2016 International Symposium. International Council on Systems Engineering. Edinburgh, GB, 18-21 July. https://www.researchgate.net/publication/308083752.
- Dove, R, K. Lunney, M. Orosz, and M. Yokell. 2023. "Agile Systems Engineering – Eight Core Aspects." INCOSE. https://www.researchgate.net/publication/373092973.
- Dove, R., and M. Yokell, M. 2026. If You Know OODA, Like Boyd Knew OODA. *INSIGHT* 29 (4), International Council on Systems Engineering, August.
- Kozlowski, S. W. J., and G. T. Chao. 2012. "The Dynamics of Emergence: Cognition and Cohesion in Work Teams." *Managerial and Decision* Economics 33 (5-6): 335–354. Wiley Online Library. https://www.academia.edu/30837626.
- SEBoK. 2026. "Systems Engineering Agility" in The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 2.14, N. Hutchison (Editor in Chief). https://sebokwiki.org/wiki/Agile_Systems_Engineering.

ABOUT THE AUTHORS

Rick Dove is an unaffiliated researcher and systems engineer focused in the systems agility and systems security areas. He is an INCOSE Fellow and chairs the working groups for Agile Systems and Systems Engineering and for Systems Security Engineering. He leads the Agility and Security project areas for INCOSE's future of systems engineering (FuSE) initiative.

Dr. Mike Yokell is a retired systems engineering leader with more than 40 years in the US aerospace and defense industry. He is the editor, co-editor, or contributor to numerous international standards on systems and software engineering. Mike is certified as an expert systems engineering professional (ESEP). He holds multiple US and European patents for model-based systems engineering and large-scale immersive virtual reality.

AUGUST 2026
VOLUME 29/ISSUE 4

Dynamic Decision Custody in Uncertainty

Larri Rosser, Larri.Rosser@rtx.com

■ **ABSTRACT**
In dynamic, uncertain environments, humans are not the weakest link—they are the last resort: the agents of judgment, improvisation, and ethical choice when automation reaches its limits. As engineered systems evolve, we have reached a point where architecture and concepts of operation must adapt to center human decision-making when it is needed most. This article explores current patterns of work that are becoming brittle, and new patterns that emerge when systems are designed for ambiguity-aware decision custody.

INTRODUCTION

Six and a half decades ago, in 1960, J. C. R. Licklider envisioned a future in which humans and machines worked together in symbiosis—each contributing their strengths and compensating for the other's limitations in pursuit of human goals. He acknowledged that such a partnership was distant, requiring advances in time-sharing, memory, programming languages, and interactive input/output before it could be realized (Licklider 1960). Since then, computing technology—and the ways humans interact with it—has evolved dramatically: from batch processing to time-sharing, from desktop applications to today's cloud-based, networked, and AI-assisted systems. The technological barriers that once constrained dynamic human–machine partnership have largely fallen. Yet the architectural and workflow patterns forged under those earlier constraints remain embedded in how we design systems and allocate authority.

As the pace, scope, and volume of work continue to accelerate, it is time to re-examine how humans and machines collaborate—not to automate more, but to architect partnership differently so we can address the complex adaptive challenges of today and tomorrow. Decades ago, Rick Dove envisioned the concept of a naturally agile organization or system "… able to transform itself with competence into whatever the new situation requires" (Dove 1999). His research on response-ability focused on structural properties that enable systems to adapt under both proactive strategy and reactive necessity. If we extend Dove's question to engineered human–machine systems, we must ask: how do we design decision architectures that are naturally agile—able to reallocate decision execution competently as conditions change?

THE LEGACY OF BATCH THINKING

In the early days of computing, the separation between human activity and computer tasks was clear and expansive. A person composed a job for the computer to execute, using an imperative language such as ALGOL. The programmer typically drafted the instructions on paper, reviewed and corrected errors, and then handed the finalized description to a punch operator, who translated each statement into a stack of punch cards. Those cards were submitted to a central computing center, queued for execution, and run by a computer operator. The results were printed and eventually returned to the requester.

Over time, this process was streamlined. Direct terminal access replaced punch cards; graphical interfaces replaced typed commands; validation tools reduced input errors. Today, users interact directly with computing systems through interactive applications that make it easy to describe and visualize intended outcomes (Pew 2002).

Yet despite these dramatic improvements in accessibility and speed, the underlying interaction pattern remains surprisingly familiar: humans formulate intent, refine it until it is "ready," and then commit the task for execution. Technology has evolved, but the authority boundary—the moment when the machine takes over completely—remains largely intact. The graphical user interface did not eliminate the batch mentality; it merely moved error correction earlier in the process.

ARCHITECTING FOR AGILE DECISION MAKING IN UNCERTAIN ENVIRONMENTS

Current architectural and workflow patterns assume static boundaries between human and machine responsibilities. Authority is typically binary: if condition A is met, the system executes; if condition B arises, control passes on to a human. Deci-

sion gates are predefined and deterministic.

As data volume increases, work cycles compress, and digital capabilities expand, this zero-or-one allocation of decisions becomes increasingly fragile. Fully automated systems can move too quickly through unresolved ambiguity. Fully human-controlled processes cannot sustain the required scale and tempo.

To avoid making the human a bottleneck or the machine an unchecked actor, we must design for dynamic sharing of decision custody. Custody should shift based on context—expanding toward automation when throughput demands it, and returning to human judgment when ambiguity, value tradeoffs, or ethical considerations emerge. This is not simply "human-in-the-loop" design. It is decision custody-in-motion design. Authority may remain formally defined, but custody of a decision should shift dynamically based on context. Authority defines who is ultimately accountable. Custody defines who is executing the decision at a given moment.

Our current default model of static pre-defined decision custody is stable and efficient when we understand all relevant input conditions, throughput, distribution and behavior options fully. However, unanticipated changes in the model can result in failures through threshold inversion, throughput collapse, assumption invalidation, coupled instability or regime shift. (Perrow 1999; Hollnagel et al. 2006; Hollnagel et al. 2011). In practical terms, this means that rare cases can suddenly become common, automated pathways can overwhelm human oversight, or underlying assumptions about workload and behavior can quietly break.

As operating contexts become more complex and adaptive, the emergence of these and other failure paths becomes more frequent, and our architectures must evolve to be more resilient to dynamic contextual changes.

Decision custody—not authority—is what must be architected for sharing. Certain decisions will always rest with humans, particularly those involving ethical judgment, value tradeoffs, or ultimate accountability. Other activities will remain fully allocated to machines, especially those that are high-volume, deterministic, or continuously repetitive. But in complex adaptive environments, a third class of work emerges: decisions whose custody should move between human and machine based on defined principles and situational context. Designing for this dynamic custody—rather than static allocation—is the architectural shift now required. The question, then, is not whether humans or machines should decide, but under what conditions custody should transfer between them.

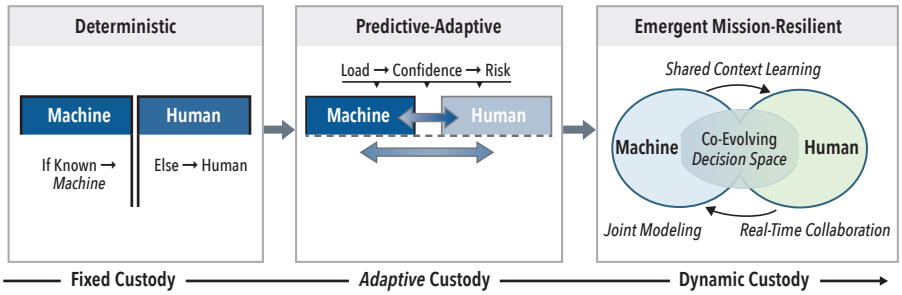


Figure 1. A spectrum of decision custody models for knowledge work

A DECISION CUSTODY EXAMPLE

To clarify this concept, let's consider a well-known and rapidly evolving decision workflow: malicious message triage. We will review both the deterministic form that is generally used today and in the more advanced decision custody architectures that could be applied.

In a deterministic custody model—the approach common in many systems today—decision boundaries are predefined. If a message matches a known threat signature or rule set, it is automatically quarantined. If it falls outside those rules, it is routed to a human analyst for review. The allocation of custody is binary and fixed: the machine handles what it recognizes; the human handles what it does not.

A predictive-adaptive custody model introduces measured variability. Here, custody shifts based on system state and confidence signals. If message volume exceeds human throughput capacity, the system expands automated classification while escalating only low-confidence cases. If anomaly detection metrics spike—indicating unfamiliar patterns—the system increases human review proportionally. The transfer of custody is not hard coded at design time but adjusts algorithmically based on load, confidence, and risk indicators.

An emergent mission-resilient custody model extends further. In this mode, the system not only adjusts thresholds but participates in reshaping the decision landscape. When novel attack patterns appear, the machine may surface clusters or hypotheses that do not match existing rules. Human analysts interpret these signals, redefine threat models, and validate new filtering logic. Custody becomes collaborative rather than sequential: the machine surfaces structure; the human reframes intent; together they evolve the triage model in response to mission conditions. The boundary between analysis and execution blurs; both partners participate in reshaping the decision framework itself.

The difference among these models is not simply technological sophistication. It lies in how decision custody is architected—whether it is fixed, conditionally adaptive, or designed to evolve under uncertainty. In systems of systems, no single component fully controls the operating context. Behavior emerges from interaction among semi-autonomous elements. Under these conditions, decision custody cannot be statically assigned without risking brittleness. Custody must be architected as a dynamic property of the whole system, not as a fixed allocation within a subsystem. The three triage models above illustrate this progression in a single domain—but the principle scales. As systems grow more interconnected and decision cycles compress, the case for emergent custody becomes not just desirable but necessary.

NATURAL AGILITY IN DECISION CUSTODY

The need for emergent mission-resilient custody becomes more apparent as we move beyond isolated applications and into complex adaptive systems and systems of systems. In these environments, behaviors arise from interaction effects among independently evolving components. Patterns shift not merely in volume or confidence, but in structure. What was once an edge case can rapidly become dominant behavior.

As digital interconnection increases and decision cycles compress, emergence becomes less rare and more routine. Static custody allocation—whether fully automated or fully human-controlled—cannot respond effectively to structural change. Architectures must therefore anticipate not only variability, but evolution. Designing for emergent custody is not a feature enhancement; it is a resilience strategy.

If we take agile principles seriously, they cannot reside solely in development methods. Systems themselves must embody agility in their operational architecture. Dynamic decision custody is one mechanism for doing so. By allowing custody of certain decisions to shift between human and machine partners based on context, systems can decentralize decision execu-

tion without relinquishing accountability. Rather than concentrating all authority at a single control point—human or automated—the system distributes decision activity to the level where it can be exercised most effectively under current conditions. This distribution supports shorter learning cycles. When machines handle high-volume, high-confidence decisions, humans are freed to focus on ambiguity, ethical judgment, and reframing of intent. When machines surface patterns or anomalies beyond predefined rule sets, they create opportunities for rapid human learning and model evolution. Each transfer of custody becomes an opportunity for feedback.

In this way, dynamic custody aligns system operation around value rather than control. Decisions are made where they can best preserve mission outcomes, not where historical boundaries happened to place them. Over time, the system adapts—not only in its parameters, but in how work itself is allocated. An agile system, then, is not merely one that was developed iteratively. It is one whose architecture enables decentralized decision execution, rapid feedback between partners, and reconfiguration of work in response to emerging conditions. ■

REFERENCES

- Dove, R. 1999. *Response Ability: The Language, Structure, and Culture of the Agile Enterprise*. Wiley.
- Hollnagel, E., D. D.Woods, and N. Leveson. (Eds.). 2006. *Resilience Engineering: Concepts and Precepts*. Ashgate. https://www.researchgate.net/publication/50232053_Resilience_Engineering_Concepts_and_Precepts.
- Licklider, J. C. R. 1960. "Man–Computer Symbiosis." *IRE Transactions on Human Factors in Electronics* HFE-1 (1): 4–11. https://doi.org/10.1109/THFE2.1960.4503259.
- Perrow, C. 1999. *Normal Accidents: Living with High-Risk Technologies* (2nd ed.). Princeton University Press. https://press.princeton.edu/books/paperback/9780691004129/normal-accidents.
- Pew, R. W. 2002. "The Changing Nature of Human-Computer Interaction." In J. A. Jacko and A. Sears (Eds.), *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (pp. 13–28). Lawrence Erlbaum Associates.
- Woods, D. D., and M. Branlat. 2011. "Basic Patterns in How Adaptive Systems Fail." In E. Hollnagel, J. Pariès, D. D. Woods, and J. Wreathall (Eds.), *Resilience Engineering in Practice: A Guidebook* (pp. 127–144). Ashgate. https://www.researchgate.net/publication/284324002_Basic_patterns_in_how_adaptive_systems_fail.

ABOUT THE AUTHOR

Larri Ann Rosser has worked in engineering and technology for four decades as an electrical engineer, software engineer, systems engineer and architect. She holds a B.S. in information systems and computer science from Charter Oak State College, and a master of science in systems engineering from Worcester Polytechnic Institute. She holds multiple patents in the man portable systems domain and is a CAP certified architect, a SAFe program consultant and a Raytheon six sigma expert. She is the co-chair of the INCOSE Agile Systems and Systems Engineering Working Group, a member of the INCOSE Complex Systems Working Group and a member of the NDIA SED Architecture Committee and the NDIA ADAPT working group.

At Raytheon, she works with programs and product lines to apply modern methods to system realization.

MBSE Enablers for Agile Systems Engineering — Reuse, Modular Systems & Product Lines

Matthew Hause, MHause@Systemxi.com

ABSTRACT

The INCOSE future of systems engineering (FuSE) initiative is a collaborative effort to realize the *Systems Engineering Vision 2035*, pivoting from traditional, process-driven methods to a more agile approach working with the different engineering and systems engineering disciplines, It focuses on addressing complex societal challenges through holistic, sustainable, and AI-enabled systems engineering, creating actionable roadmaps. The The Agile Systems and Systems Engineering working group has identified eight strategic aspects, two of which are the following:

- Shared-knowledge management – facilitated communication, collaboration, and knowledge curation
- Adaptable modular architectures – composable and reconfigurable designs from evolving variations of reusable assets

Model-based systems engineering (MBSE) has progressed to the point that it is now ubiquitous in systems engineering. It is therefore logical that MBSE be used together with systems engineering agility in order to leverage its benefits. This article will examine how the OMG Reusable Asset Specification (RAS), the Modular Open Systems Approach (MOSA), product line engineering (PLE) used as part of MBSE, and agile can address these strategic aspects to enable systems engineering agility.

INTRODUCTION

When developing systems, uncertainty is inherent and prevalent in the environment including requirements, stakeholder needs, technology, solution spaces, future trends, and unknown and unforeseen emergent properties. Given the dynamic nature of the world and technology, static development methods starting with fixed requirements and resulting in a set solution with a fixed architecture are no longer practical from either a business or technology viewpoint. "Agile systems engineering is a principle-based method for designing, building, sustaining, and evolving systems where knowledge is uncertain and environments are dynamic" (Systems Engineering Agility Primer). The Agile Systems and Systems Engineering "working group views agility as a sustainable system capability, enabled and constrained fundamentally by system architecture. This architecture delivers agile capability as reconfiguration, augmentation, and evolution of system functionality, enabling the system to respond to new and immediate situational requirements effectively. Effectiveness of response is measured in response time, response cost, response predictability, and response scope sufficient to sustain the system's functional intent." (INCOSE Agile Systems and Systems Engineering Working Group 2026). The digital thread is a formalized, seamless digitization and connection of system knowledge across the entire development and system lifecycle. It integrates model-based systems engineering (MBSE) with software, mechanical, and operational domains to establish authoritative traceability and configuration management. We need to move away from static, document-based engineering into living data repositories. These distributed "authoritative sources of truth" (ASOT) can no longer be silos but must be connected so all stakeholders can reference the exact same data point regardless of where it is used. This includes linking upstream requirements to architecture, downstream design, simulation, and manufacturing. Systems engineering agility must be integrated into the MBSE digital thread in order ensure agile systems, processes, and systems.

MODEL-BASED SYSTEMS ENGINEERING (MBSE)

RAS, MOSA, and PLE are all aspects of MBSE. None of them are process or methodology specific and can be used as the engineer sees fit. Each topic would require an entire paper or possibly a book to fully explain them. Instead, the author will summarize each topic, provide references for the reader to explore, and demonstrate how each one enables systems engineering agility.

"MBSE is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing through-out development and later life cycle phases" (INCOSE 2007). Some engineers have limited MBSE to model-based tools using the Systems Modeling Language (SysML) or other modeling languages (OMG 2023). MBSE is best employed as part of a digital engineering ecosystem (DEE) that includes tools for requirements management, enterprise architecture, physics-based modeling and simulation, and product lifecycle management (PLM) to connect systems models to physical parts, and throughout the lifecycle. A successful development team requires multiple members with different areas of expertise. In the same way, each of these tools in the ecosystem will provide a digital thread that will assist the engineers in creating a solution making use of multiple viewpoints, analysis and simulation capabilities, to ensure a wholistic solution. Interoperability and interchange between the tools are currently ad hoc, something which SysML v2 hopes to rectify by providing a common application programming interface (API) (OMG 2025).

MBSE development processes such as the INCOSE object oriented systems engineering methodology (OOSEM) and the Dassault Magic Grid emphasize creating architectures made up of modular structural and functional components to provide a system, or systems of systems to meet the stakeholder needs. The MOSA strategy aims to formalize this approach.

Modular Open System Approach (MOSA)

MOSA is a "U.S. Department of Defense (DoD) strategy for designing, acquiring, and sustaining complex systems using modular, interchangeable components and open standards. It aims to enhance interoperability, competition, and innovation while reducing costs and enabling faster technology upgrades, mandated by law for defense acquisition programs" (Defense Standardization Program) (DoW 2026). MOSA compliant developments:

- Employ a modular design that uses modular system standards-based interfaces.
- Are subjected to verification to ensure that relevant modular system interfaces comply with widely supported and consensus-based standards including the delivery of:
  - Software-defined interface syntax and properties, showing how values are exchanged between major subsystems and components.
  - Definition of the relationship between the delivered interface and existing common standards.
  - Documentation with functional descriptions of software-defined interfaces.
  - A system architecture that allows severable major system components at the appropriate level to be incrementally added, removed, or replaced throughout the life cycle of a major system platform to afford opportunities for enhanced competition and innovation.

Summarized from (Defense Standardization Program).

MOSA Benefits include:

- Significant cost saving or avoidance
- Schedule reduction and rapid deployment of new technology
- Opportunities for technical upgrades and refresh
- Avoiding "vendor lock-in"
- Interoperability, including system of systems interoperability and mission integration
- Other benefits during the sustainment phase of a major system.

(Defense Standardization Program)

MOSA directly supports the strategic benefit of adaptable modular architectures by specifying composable and reconfigurable designs. The benefits listed above for MOSA will also be realized for systems engineering agility and its use of adaptable modular architectures. They will also enable reconfiguration of the system due to the implementation of standard interfaces. For evolving variations of reusable assets, PLE and RAS will be required.

The Reusable Asset Specification (RAS)

Models are typically created from scratch for each project resulting in wasted time, reinvention of the wheel, and repetition of errors and the learning process. (Rhodes 2019; Rhodes and Ross 2015; Hause et al. 2026) Each module is custom created for the target system with no thought of reuse and the benefits it provides. The paradigm needs to change from building models from scratch to assembling models from reusable components as industry has done for the last 200 years. Eli Whitney applied the concept of reusable and interchangeable parts to musket production and the cotton gin. Henry Ford made automobiles affordable to the general public by using interchangeable parts and the assembly lines. Prior to this, each part was unique and made by hand requiring skilled craftsmen and plenty of time. Software engineering adopted modular methods in the 2000s that were originally conceived by Dijkstra. He emphasized that complex, large-scale systems must be broken down into manageable, independent components (modules). His philosophy, which he called "separation of concerns," focused on designing software where modules have well-defined, abstract interfaces, allowing them to be understood and verified in isolation. This enabled modularization and reuse as best practice. MBSE has not yet achieved this in practice on any scale.

The Object Management Group (OMG) originally created the RAS in 2000 to define a standardized approach to software asset description and management. It was not widely adopted as software engineers preferred configuration management systems and other mechanisms. The OMG is working on an update for RAS to support both software and systems engineers. In addition to asset description and management, RAS 3.0 will also support cataloging, and access through structured metadata, searchable catalogs, and supporting curation mechanisms to assets in curated repositories. There are no restrictions on repositories. They can take the form of SQL databases, GITHUB or CM applications, teamwork cloud environments, file systems or any format desired. Repositories will be maintained by the OMG to hold standardized elements for SysML extensions, private industry to encourage reuse by internal project teams, and by the government to ensure an authoritative source of truth for government reference architectures. The key to asset discovery and reuse is the standardized catalogs listing the available assets. Each asset card in the catalog has a standard set of attributes such as:

- Identity & intent – artifact ID, name/title, description/purpose, keywords
- Stewardship & access – contact organization, license type, repository/access URI
- Version & lifecycle – version, lifecycle status (e.g., draft, released, deprecated)
- Governance & provenance – parent asset ID, lineage relationship type, version history, related assets.

These provide a variety of ways for asset discovery. The asset attributes can be extended as required. Assets can be stored in multiple formats including tool vendor formats, XML, JSON, or other. There is a standard API for catalogue search and asset create, read, update, and delete (CRUD) actions as well as others. There is a standard format for asset packaging to provide exchange for assets. Major MBSE tool vendors are planning to implement the RAS 3.0 standard when it is released later this year. (Hause et al. 2025) Assets contain project and program "knowledge" that can be stored, retrieved, shared, and reused. This assists in the systems engineering agility shared-knowledge management benefit as it facilitates communication, collaboration, and knowledge curation in an easily useable format. PLE provides an organizing set of principles regarding the organizing and configuration of the assets.

Product Line Engineering (PLE)

PLE is the engineering and management of a portfolio of related products using a shared set of assets, a feature catalogue, and a configurator or engine (INCOSE PLE 2019). PLE assets are not just software assets. Assets can include both system and software assets and involve all aspects of engineering including software, electrical, electronic, mechanical, chemical, and procedural. Instead of treating products as separate, "clone-and-own" projects, PLE manages variations and similarities across a product line, significantly improving efficiency, reducing time-to-market, and enhancing quality. Model-based PLE or MBPLE combines these elements together. Key aspects of MBPLE include:

- Shared assets and variability: PLE utilizes a shared platform of assets—such as requirements, design models, test cases, and software code—that contain built-in variability to accommodate different product needs.
- Feature-based management: Products are defined by selecting specific features from a feature model, which automatically configures the shared assets to create the desired product variant. Features could also be stored in a RAS catalog.
- Product line model: The product line model contains the various available assets for a product (different engine types, transmission types, components). Assets are associated with the various features that they support. This is known as the 150% model.
- A PLE "engine": Advanced tools are often used to automate the generation of products from these shared assets in the product line model. Engineers choose their required features, and a product model is created from the product line model. The authoritative source of truth (ASOT) aspect of RAS ensures that the authoritative sets of assets and features are used in the configurations.

Generated configurations can relate to fixed manufactured products as well as dynamic configurations. For agile systems, the conversion between configurations can take several forms of various levels of time and effort:

- A software "switch" that enables or disables system capabilities. Features in passenger vehicles such as heated seats may require a subscription to enable them. This can be done remotely by the manufacturer. For military systems, this can consist of changing the mission profile software directly by the war fighter.
- Adding or removing equipment. Again, for passenger vehicles this could be adding or removing luggage racks or safety equipment, changing tires, towing trailers as needed depending on the journey. For military vehicles, weapons, drones, or other equipment could be added or removed depending on the mission.
- Structural reconfiguration. For passenger vehicles, this could be converting a normal pickup truck to a monster truck by changing the chassis, adding giant tires, updating the engine. For military vehicles, the vehicle could be repurposed to move heavy equipment or by providing heavy duty armor. For both examples, the change in the configuration will involve considerable time and expense.

For systems that will be involved in multiple missions, configurations and usages, studies will need to be done to determine the time and cost for adapting from one purpose to another. Based on these studies, the decision may be made to provide multiple vehicles of different configurations as opposed to a reconfigurable vehicle. This will also be driven by the amount of time each configuration is required.

FURTHER CONSIDERATIONS

Of course, implementing this will require additional effort. Creating a reusable component takes more time than creating single use components. Reusable assets require deeper upfront architecture, flexible interfaces, and extensive documentation. Reusable components require robust state management and defensive programming to handle varied data inputs. Detailed documentation and thorough error handling are required so users understand exactly how to manage and extend the asset. Though this takes more initial planning, it drastically reduces long-term maintenance and accelerates future development. Cost benefit analysis should be done to ensure that the asset is worth curating. For existing assets, the benefit comes from the development costs saved by not having to recreate it each time. Reusable assets can be used on new projects as well as existing projects. The mindset needs to change from building models from scratch to assembling models from reusable assets.

CONCLUSION

Systems engineering agility and MBSE are recognized as best practice in INCOSE as well as in the greater systems engineering domain. Both promise to provide better, faster, and cheaper development of systems as well as responsiveness to changing requirements, environments, technology development timescales, and user needs throughout the lifecycle. Systems engineering agility can involve the MBSE development project to ensure that the stakeholder needs are met and that the system evolves as the environment evolves. MBSE coupled with RAS, PLE, and MOSA can enable SE to provide a responsive, adaptive, and agile system as well as the development process. The combination of these will provide better systems, better systems engineering processes, and better and more effective systems engineers. ■

REFERENCES

- INCOSE Agile Systems and Systems Engineering Working Group. 2026. From the Agile Systems and Systems Engineering Working Group website. https://www.incose.org/group/agile-systems-systems-engineering-working-group/.
- Rhodes, D. 2019. "Model Curation: Requisite Leadership and Practice in Digital Engineering Enterprises." 17th Annual Conference on Systems Engineering Research (CSER).
- Rhodes, D. H., and A. M. Ross. 2015. Interactive Model-Centric Systems Engineering Pathfinder Workshop Report. MIT. Cambridge, US-MA.
- Hause M., S. Krishnan, M. Shearin, T. Vileiniskis, and M. Petrotta. 2026, "Making Reuse Real in MBSE – An Overview of RAS 3.0." Presented at the INCOSE International Symposium, Japan.

> continued on page 45

Wayfinding Through Change

Kenneth Kubo, angliken316@gmail.com

■ **ABSTRACT**
As systems grow larger, more complex, and begin to incorporate expectations of the deployment environment, infrastructure, and multiple subsystems, a disciplined approach is essential to preserving the understanding of the product vision. Once the purview of big industrial development houses, the practices of systems engineering and a systems thinking mindset are enablers of development productivity, ensuring clear elaboration of stories and acceptance criteria, incorporation of overarching requirements such as regulatory compliance, ongoing management of system margins, and understanding and maintaining the right level of testing. Regardless of your choice of scaling framework or tools, you can evolve and tell the story of the system so that all members of all teams can drive to a common valuable result. In particular, this presentation draws on the recent work of the NDIA Systems Engineering Division to update how technical capabilities are acquired, managed, and evaluated to responsively field system functionality at the speed of relevance.

INTRODUCTION

The wayfinders of Polynesia guided voyaging canoes across vast distances, allowing commerce and exploration across trackless oceans between far-flung islands. They did not have the benefit of modern GPS, yet they navigated accurately between these distant points. The Wayfinders took input from the stars, the seas, and the marine ecosystem to adjust course based on where they had already travelled, where they were, and where they needed to go. With their skills, they helped their fleets make the journey from start to end through feedback, adaptive thinking, and ongoing course correction of incremental progress.

In the journey of product development, systems engineers are the wayfinders, understanding where the product is at any time, how it got there, and where it's heading. Systems engineers own the journey of the system – past, present, and future – and must communicate that consistently and collaboratively to ensure that everyone from stakeholders to program managers to developers shares the same understanding of the system. The value proposition of good systems engineering is to evolve a system that meets the needs of its users – to build the right system and to build the system right. It's key to remember that the function of systems engineers is not to write specs or maintain requirement databases or develop test approaches – it is to use those practices (and more!) to ensure that the products their teams are developing reach their desired destinations and deliver value to their stakeholders. This responsibility becomes increasingly important in an innovation-driven environment, as new techniques are needed to efficiently accommodate fluid system intent and increased development discovery.

CHARTING A COURSE IN AN INNOVATION AGE

The journey for large contracted system development is perilous! The challenges of the modern age drive tighter program constraints, tighter technical constraints, and the need to deal with rapidly changing missions and technology (Clements et al. 2019).

Current programs are struggling with budgetary limitations and project allocations, requiring priority calls and portfolio optimization for project approval. This raises challenges with whether the desired capabilities can be developed within the available budgets. This is coupled with aggressive (and inflexible) deployment timelines which apply pressure to traditional development methods. This has led to an increase in the use of fixed-price development contracts to bind tightly defined development scope to a constrained budget. However, if we fix price and scope, we increase risk in holding schedule.

Technical change has always been a source of risk in large-scale product development. The need to reduce time and cost is leading to greater consideration of commercially provided, off-the-shelf components as well as the integration of emerging suppliers who have not previously worked in industrial contracting. These new sources are thus not necessarily familiar with the rigorous quality standards or engineering practices, especially in the case of government contracting. This increases potential integration and performance risk. With commercial providers of infrastructure, design trades face challenges in balancing space, weight, and power (SWP). Moving from a development world where all components are bespoken to integrating with stock elements creates greater risk of discovery when implementing. Cybersecurity compliance also continues to be a

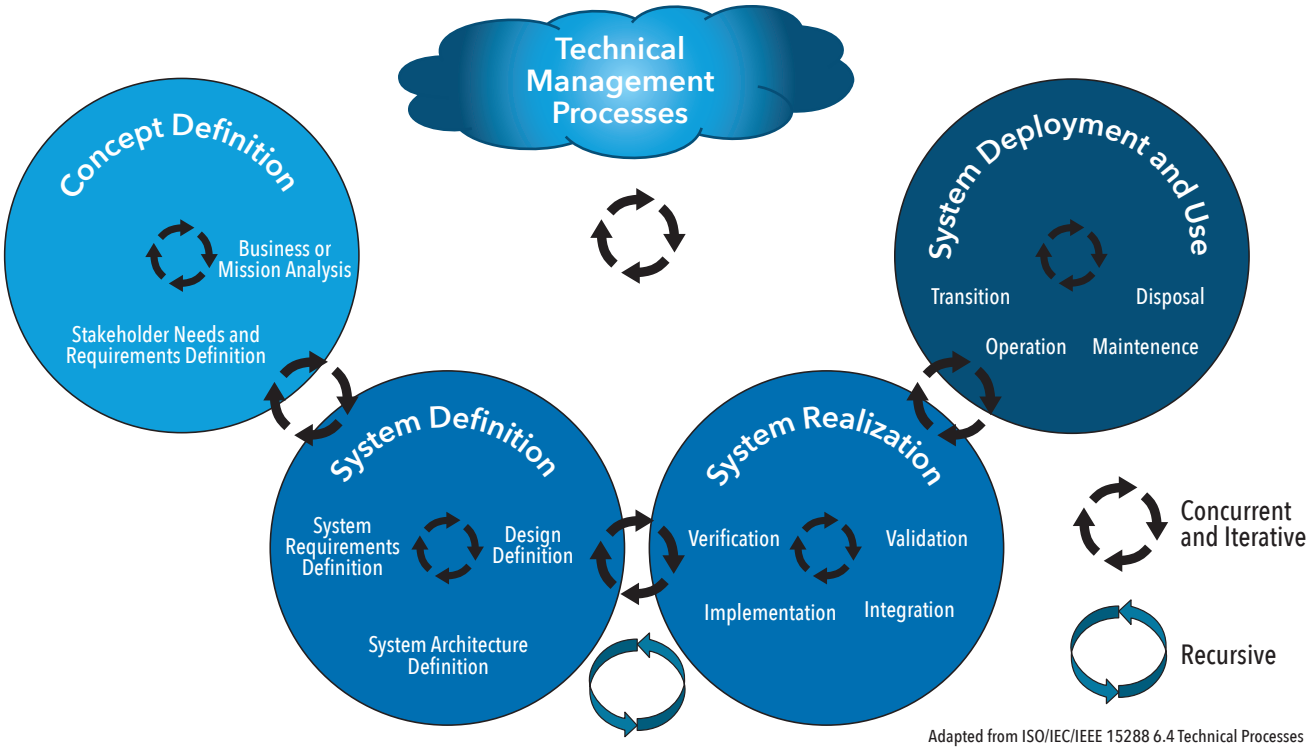


Figure 1. ISO/IEC/IEEE 15288 6.4 Technical Processes

dependable source of change, impacting how systems are designed, developed, and accredited.

The rapid pace of mission evolution impacts deployment timelines and creates pressure on development. Delivering capabilities to an environment of changing needs at the speed of relevance is increasingly challenging. The rate of change increases the risk of discovery because of the integration of less mature technologies and less fully developed designs. With the pivot to incremental product development methods, especially in the software pipeline, developers must respond to a series of minimum viable products (MVP) rather than a fully realized end-to-end system.

One such example is the call from the US DoW to "modernize systems engineering: build and sustain our weapon systems better, faster, and cheaper by using advanced acquisition and physics-based models" (US DoW 2025). Rapidly changing customer needs and the ongoing quest for "better, faster, and cheaper" certainly signal some rough seas ahead for the journey. What does "modern" systems engineering have to become in the environment just described in order to safely bring our programs to a safe harbor?

To frame the changing needs of the discipline, the current version of ISO/IEC/IEEE 15288 adds concurrency and recursion to the familiar systems engineering "V" (Figure 1). While the "V" is still visible in this reimagined layout, there is no consistent progression through the processes. This is no longer a series of phases with clean handoffs between them. This merely acknowledges the fact that few programs of any size or complexity have been able to perfectly step through the phases without adjustment based on later discovery. After all, systems engineering is (and has been since its inception) an empirical discipline where some level of incremental learning is expected. It is key to understand that all the systems engineering practices are still here – good engineering is not going out of style – but the timing, the scope, and the maturity of systems engineering products need to change because the environment is changing. So, if all systems engineering processes can occur concurrently amid changing (and even stormy) seas, what practices need to evolve to avoid steering into dangerous waters?

INCREMENTAL SPECIFICATION – REFINING THE DESTINATION

One of the primary tasks of a wayfinder is knowing where the craft needs to go. An evolving mission and technology space means that it is no longer practical to fully define the end-to-end system at the onset of development. The final destination is generally known, but some of the details remain over the horizon. The default environment is now one of incremental specification, where the highest priority functionality (or defined customer MVP) dictates which parts of the system must be elaborated to support development. Incremental specification drives progressive design which enables incremental development and deployment. In this kind of development context, this means that the systems engineering job is to incrementally translate customer-prioritized functionality into actionable work descriptions and acceptance criteria. These drive the elaboration of the value hierarchy, preserving the complete and consistent breakdown of work (for example, agile capabilities to features to stories) so that clear verification intent drives developer understanding. Good systems engineering occurs a little before its time, with the work of refining developer guidance completed before the developers need it and with the most current information. This requires systems engineers to focus on what information will be needed for the next iteration while keeping an eye on the overall heading of the product.

- Systems engineering should be refining developer guidance in the previous iteration; during each developer iteration, systems engineering refines the work for the following iteration while it supports development for the current one. Leveraging ongoing collaboration with stakeholders/customers/users, the focus is to define for that next iteration: What are we building and why? (system functionality) Where will it be delivered and why? (lab for testing, representative system for integration, actual operations) How do we know it's done? (verification

approaches) What dependencies do we have? (giver-receivers, suppliers, team capabilities)

- Systems engineering should maintain bi-directional traceability of requirements to developer work for shared understanding of which requirements are being addressed in each increment of the product. For agile programs, this means understanding what features and stories cover a requirement, so that the completion of that work tracks to the verification of that requirement.
- Systems engineering acceptance criteria for developers should include the verification approach so that developers clearly understand what their products have to do.
- Systems engineering should maintain an ongoing set of architectural trades and enabling decisions extending multiple iterations ahead to ensure that these are addressed at the appropriate responsible moment – before the development teams need an answer. This provides for a balance of intentional architecture against incrementally emergent architecture.
- Systems engineering should drive a modular open systems approach (MOSA); in addition to frequently being a contract requirement, MOSA is a key enabler of progressive design and incremental deployment. This allows capability to be built up and upgraded with pieces of functionality separated with well-defined interfaces, and it lowers the cost of change of late discovery or mission evolution.
- And note that this kind of response generally requires a systems engineering staffing profile that stays more level across program execution, since incremental specification drives incremental requirements analysis and design.

Being a good product wayfinder also means incorporating systemic feedback to enhance understanding of where system development has been as well as where it currently is, enabling an efficient vector to the (near) future.

INCREMENTAL INTEGRATION — FORCING DISCOVERY

In addition to the change in approach caused by incremental specification and development, modern product development is also affected by the incorporation of "not invented here" components. This increase in the amount of outside project components drives the need for a focused plan to buy down the cost of discovery. The goal is to "shift left" integration activities as much as feasible in order to uncover potential rework while there is still opportunity to address it efficiently. A standard approach to this is to use an iterative cadence to progressively integrate the system in order to discover any issues quickly. Integrate across as much of the system as reasonable on a regular rhythm. Focus especially on those areas of highest risk – typically interfaces and anything new (newly developed or new to the team). Whether components or subsystems are physical or digital, being able to put the pieces together and measure performance on an iterative basis helps identify where the system really is and where course correction may be needed. Involving the entire development team – including external suppliers – in both incremental planning and integration activities ensures that everyone pivots together. This exposes any issues with timelines and priorities as well as technical assumptions. An agile mindset encourages reducing risk on an ongoing basis. Prioritizing work of highest risk or greatest uncertainty drives rapid incremental learning to reduce systemic unknowns and mature designs. Applying a DevOps mindset as part of early and incremental integration helps to maintain an ongoing satisfaction of compliance and ecosystem requirements. As the system is incrementally built up, ongoing verification of important "non-functionality" ensures that the system is not accruing operational risk.

- Systems engineering should identify integration points across the system needed to verify functionality for each MVP; this includes both developed content and the integration environments. These should be planned into the development roadmap. As an example – integrating subsystems every iteration, integrating all software subsystems every two to three iterations, integrating the hardware-software system every six iterations.
- Systems engineering should identify and define interfaces and establish where and when these interfaces will be tested. Systems engineering should ensure collaboration between developers and stakeholders on all sides of the interface.
- Systems engineering should maintain both the logical and physical views of functionality; logical to inform the description and acceptance criteria for development, physical to inform integration and verification approaches and definition of interfaces.
- Systems engineering should ensure that expectations (statement of work) provided to suppliers include support of early and incremental integration with delivery of components of increasing fidelity – models, emulators, prototypes, engineering units.
- Systems engineering should maintain the "-ility" requirements and the compliance verification approach for each level of integration.

In all cases, the intent is to find issues while the cost of change is relatively low. Keeping the course changes small allows pivoting with new information without disrupting the team. Remember that a wayfinder needs to understand the journey to date – performance and compliance trends, margins and trades especially across interfaces – as well as monitor where the system is.

INCREMENTAL EVALUATION—TAKING A HEADING CHECK

> …the traditional requirements development, preliminary design review, and critical design review events may be replaced by incremental design reviews, and, if needed, system-level reviews.
>
> GAO 20-590G

The empirical nature of systems engineering in a rapidly changing environment means that the journey is not defined by its endpoints. Setting sail from Hawai'i to Tahiti is not executed as a straight line without ever checking in. Product development is expected to require course correction. Progressive design and incremental development require a supportive approach to verification. Requirements, design, and the system itself must be verified on an ongoing basis rather than at arbitrary major milestones. The pace of evaluating the system against objectives must match the change rhythm of system intent from iteration to iteration as well as the frequency of system integration and deployment. Verification is thus planned incrementally, MVP by MVP. Given a model-based development environment, this allows more use of model-based testing and earlier performance analysis. Using physics-based models allows evaluation of system behavior that is more representative of performance in the real world. These can provide a reusable simulation environment for multiple types of systems operating in particular theaters. Testing can begin at the modeled level and then provide stimulation to systems of increasing fidelity—models to prototypes to high fidelity models to engineering units to fieldable units. Leveraging an integrated test facility that is designed to allow the interaction of components of

Review	Purpose	Intent
System Requirements Review (SRR)	Assesses the system requirements captured against the mission needs	Confirm requirements align with stakeholder expectations
System Functional Review (SFR)	Evaluates the systems' functional and performance requirements to ensure that they are fully defined and aligned with user needs	Ensure functional baseline captures how the system should work.
Preliminary Design Review (PDR)	Evaluates the initial design against mission needs to ensure that it meets requirements	Ensure allocated baseline is valid and meets requirements within acceptable risk.
Critical Design Review (CDR)	Evaluates the matured design against mission needs to ensure that it meets requirements	Ensure product baseline meets requirements within acceptable risk
Test Readiness Review (TRR)	Evaluates the system's readiness for testing	Ensures the test plan and state of the system are adequate to minimize risk

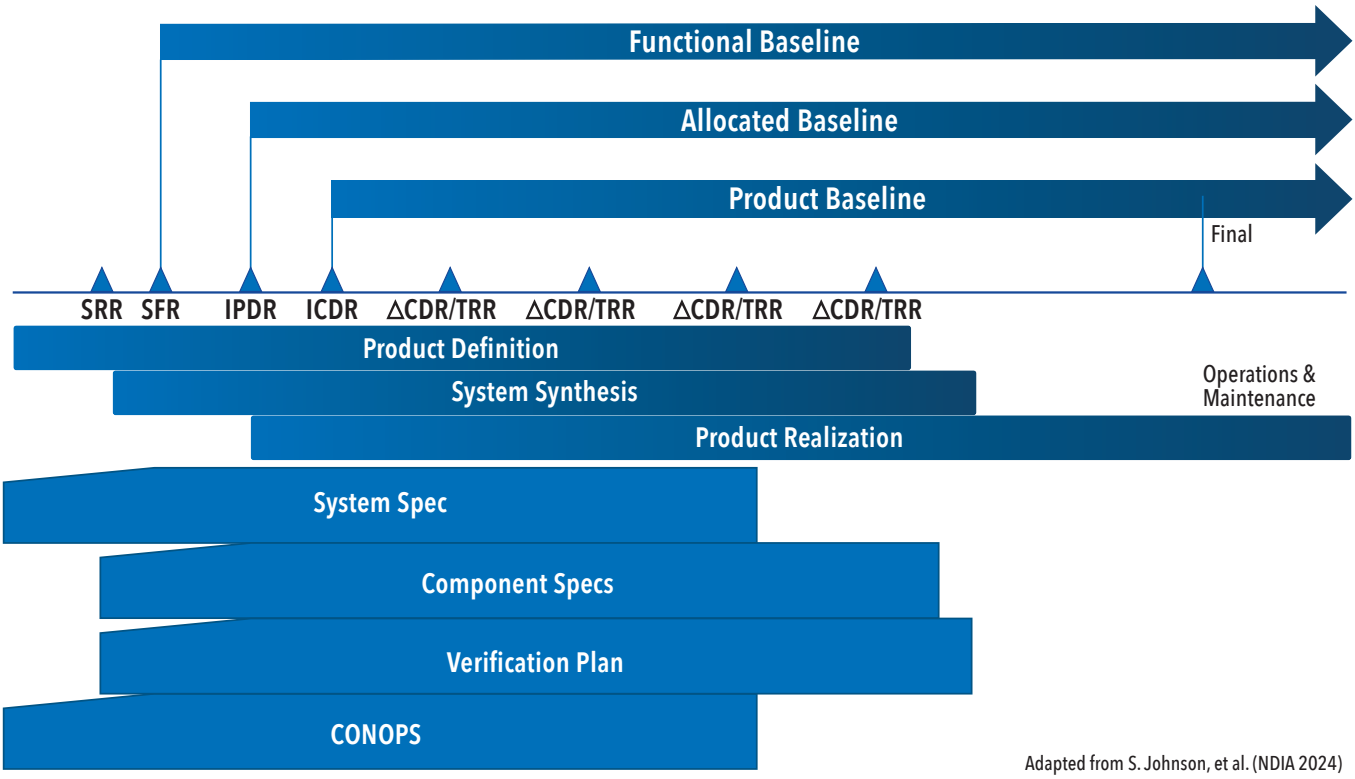
Adapted from S. Johnson et al. (NDIA 2024); derived from ISO/IEC/IEEE 15288.2

different fidelities provides a virtual system integration and verification capability essential to rapidly mature end-to-end solutions.

Incremental specification also drives the distribution of traditional systems engineering technical reviews (SETR). With a system that is evolving at all parts of the systems engineering process, incremental reviews allow the appropriate evaluation of the system as it is being constructed. With the standard SETRs, the course of the system was progressively set, focusing on reducing development risk through comprehensive maturity gates (see Table).

As with systems engineering practices in general, the intent of these reviews is still valuable, but the timing and the breadth of products evaluated must evolve to accommodate requirements subject to change. Consider that at SRR, the decision is made to sail to Tahiti. At SFR, we plot the course. At PDR, we identify the vessel and crew needs. At CDR, we have the vessel, crew, provisions, and course. And off we sail with no further feedback until we are late in arriving or run into Okinawa. Practically, program execution has assumed that these reviews imperfectly assure success because with any sufficiently complex system in a changeable environment, baselines will change. Figure 2 shows an adaptation of the traditional reviews to manage this reality.

The functional, allocated, and product baselines do not lock down at their corresponding reviews. The functional baseline still locks down before the allocated baseline before the product baseline, but there is overlap during which the baselines continue to evolve with each increment of development. As the baselines mature, this also leads



Adapted from S. Johnson, et al. (NDIA 2024)

Figure 2. Incremental review cadence

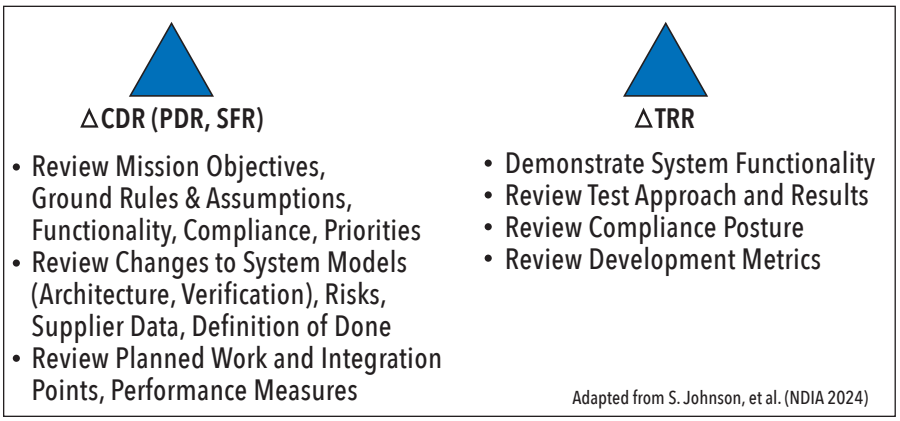


Figure 3. Incremental review content

to ongoing maturation of specifications and verification plans. Incremental specification naturally leads to the need for incremental verification and thus to the need for incremental delivery and customer acceptance.

As shown in Figure 3, these incremental reviews still serve to confirm that the baselines are at the appropriate level of maturity to proceed with development. However, each delta review focuses on the developed product(s) for one incremental cycle. This works most effectively if there is continued context for the journey to this point. These reviews provide the opportunity to evaluate whether development is still on course and decide the direction of the next leg of the journey. Advanced acquisition on the customer's side means in part being able to accept – and fund – incrementally developed scope through incremental evaluation.

- Systems engineering should assist in determining the value-added integration and test environments and when they will be needed as early in the program life cycle as possible.
- Systems engineering should establish and support the rules of engagement for incremental evaluation and customer acceptance of developed functionality.
- Systems engineering with the development teams should identify and track technical debt in the system from issues exposed through integration and test, delayed or blocked work, or short-term solutions.
- Systems engineering should incorporate feedback and changes to priorities or environment and adjust the technical plan accordingly.

CONCLUSION

The Polynesian wayfinders and systems engineers share the requirement to respond to uncertainty and complexity through domain knowledge, integration of multiple sources of data, and empirical methods. Their calling is to bring their teams through a journey and safely to their desired goal. To successfully navigate an environment of incremental specification requires systems engineering techniques that support progressive design, incremental development, and iterative evaluation. This includes managing the technical baseline – both the definition of scope and the means of verifying it–through ongoing maturation. Systems engineering is the discipline of absorbing complexity and rendering clear, constantly communicated direction so that product development is always heading in the right direction. The practices of the traditional systems engineering "V" model remain relevant – the timing and needed maturity at any point in development are what needs to evolve to match the new execution environment. In a journey across a changeable seascape of complexity, uncertainty, and unknowns, systems engineering remains the vital discipline to guide program development from where it was, where it is, to where it needs to go.

When people come together around common vision, they can accomplish great things.

—Nainoa Thompson, wayfinder and president, Polynesian Voyaging Society ■

REFERENCES

- Clements, J., T. Murphy, L. Jasper, and C. Jacka. 2019. "Tailored Systems Engineering Processes for Low Cost High Risk Missions." Small Sat Portfolio. AFRL.
- IEEE/ISO/IEC 15288. 2023."Systems and Software Engineering-System Life Cycle Processes." IEEE Software & Systems Engineering Standards Committee, March.
- Johnson, S., G. Kranz, K. Kubo, S. Thyagarajan, R. Yeman, and P. Zajac. 2024. "Moving from Predictive Planning to Empirical Planning for Systems Engineering." NDIA Systems Engineering Division, March.
- Johnson, S., et al. 2025. "IEEE 15288 Meets Lean Agile." NDIA Systems Engineering Division, June.
- US Department of War (DoW). 2025. "Acquisition Transformation Strategy: Rebuilding the Arsenal of Freedom."
- US Government Accountability Office. 2020.
- "Agile Assessment Guide: Best Practices for Agile Adoption and Implementation." Report 590G.

ABOUT THE AUTHOR

Ken Kubo is the current lead of the Space Sector Lean-Agile Center of Excellence for Northrop Grumman and an evangelist for systems engineering in an agile context. He has over 30 years of service with Northrop Grumman, with experience in software development, network security, systems engineering, and satellite operations. Ken supports project teams making their agile transitions as coach and trainer. He is a member of the INCOSE Los Angeles Chapter and the NDIA Systems Engineering Division and continues to be active as a speaker at internal and industry conferences. Ken is a scaled agile framework (SAFe®) practice consultant, an ICAgile certified agile coach, and a certified scrum product owner and scrum master; he holds an MS in management science from California State University/Fullerton and a BS in engineering/English from Harvey Mudd College.

Decomposing Complexity: A Systems Engineering View of Modern Software — Aligning the V-Model with a Slicing-Based Decomposition Strategy

Shreya Sridhar, shreya.sridhar@medtronic.com

■ **ABSTRACT**

Modern software-intensive systems–particularly in regulated domains such as medical devices -have reached a level of complexity that challenges traditional approaches to system decomposition. While the V-model has long served as a foundational systems engineering framework for ensuring traceability and verification rigor, its practical application in large-scale software programs often reveals a disconnect between how systems are decomposed and how they are ultimately built, integrated, and verified.

A growing need has emerged for a decomposition approach that preserves the intent of the V-model while adapting to the realities of iterative development, cross-functional dependencies, and continuous integration. One such approach is slicing, a systems-oriented decomposition strategy that reframes how complexity is structured shifting from component-centric thinking to behavior-driven, end-to-end system realization.

This article explores the parallel between the V-model and slicing, arguing that slicing does not replace the V-model, but rather operationalizes it in a way that is more aligned with modern software development.

THE V-MODEL: STRENGTHS AND STRUCTURAL LIMITATIONS

The V-model remains one of the most enduring representations of systems engineering discipline. Its strength lies in its explicit pairing of decomposition and verification, ensuring that every requirement at each level of abstraction is ultimately validated through corresponding test activities. Stakeholder needs translate into system requirements, which are further refined into subsystem and component definitions. On the opposite side of the "V," integration and verification activities confirm that each layer satisfies its originating intent.

However, in practice—especially within software-intensive systems—the decomposition side of the V-model is frequently interpreted through a structural or component-based lens. Teams organize around subsystems, modules, or services, each owning a portion of the architecture. While this supports parallel development, it introduces several challenges:

- Fragmentation of system behavior across organizational boundaries
- Late integration risk, as end-to-end functionality emerges only during system-level testing
- Loss of traceability clarity, particularly when features span multiple subsystems
- Misalignment with user workflows, which are inherently cross-cutting.

In effect, while the V-model defines what should be preserved—traceability, verification alignment, and requirement integrity—it does not prescribe how to structure decomposition in a way that ensures these qualities are maintained in complex software environments.

INTRODUCING SLICING: A BEHAVIOR-CENTRIC DECOMPOSITION PARADIGM

To address the growing gap between traditional system decomposition and the realities of modern software development, an alternative approach has emerged: slicing. At its core, slicing is a systems engineering decomposition strategy that structures a system around end-to-end behavior, rather than static architectural components.

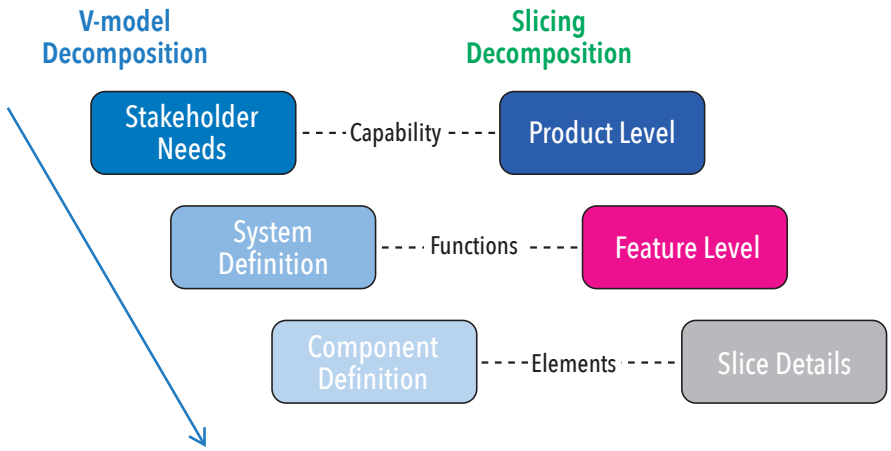


Figure 1. V-model and slicing as complementary decomposition strategies

Unlike conventional decomposition methods that partition systems into subsystems or modules owned by individual teams, slicing defines the system as a collection of cohesive, vertically integrated increments. Each increment—referred to as a slice—represents a complete thread of functionality that spans multiple layers of the system, from user interaction through processing logic to data handling and integration (Figure 1).

A slice is therefore not a component, nor is it merely a feature in the traditional sense. It is a behaviorally complete unit of system realization, designed to preserve the relationship between what the system is intended to do and how it is implemented.

Slicing typically operates across two levels of abstraction. At the higher level, product-level slices represent meaningful system capabilities derived directly from stakeholder needs. These slices define how value is incrementally delivered, ensuring that each increment corresponds to a coherent and testable outcome from a user or system perspective.

At a more detailed level, feature-level slices decompose each capability into smaller, implementable units while maintaining end-to-end continuity. These slices cut across functional layers—such as user interfaces, algorithms, data models, and integration logic—ensuring that each unit of work contributes to a fully realizable system behavior rather than an isolated technical artifact.

Figures 2 and 3 illustrate the distinction between product-level slices, which organize delivery around stakeholder-visible capabilities, and feature-level slices, which decompose those capabilities into smaller implementable units.

This dual-level structure allows slicing to bridge a critical gap in modern systems engineering: it provides a mechanism to connect high-level intent with low-level implementation without losing behavioral coherence. Rather than decomposing the system into independent parts and attempting to reassemble them later, slicing ensures that each part is inherently aligned with the overall system function from the outset.

Importantly, slicing does not eliminate architectural thinking or component design. Instead, it recontextualizes components within the flow of system behavior, allowing architectural elements to emerge in response to functional needs rather than dictating them prematurely. This shift enables teams to reason about the system in terms that are closer to stakeholder intent, operational workflows, and real-world usage.

In doing so, slicing establishes a foundation for integrating systems engineering rigor with modern development practices. It preserves the discipline of structured decomposition while enabling a more dynamic, iterative, and behavior-driven realization of complex software systems.

SLICING AS A SYSTEMS ENGINEERING DECOMPOSITION STRATEGY

Slicing addresses this gap by introducing a decomposition paradigm that aligns more naturally with system behavior. Instead of breaking the system into static components, slicing decomposes it into end-to-end functional increments that cut across multiple layers of the system.

At a high level, slicing operates across two complementary dimensions:

- Product-level slices, which represent meaningful, end-to-end capabilities delivered incrementally
- Feature-level slices, which further decompose these capabilities into cohesive units spanning user interface, algorithms, data structures, and integration logic.

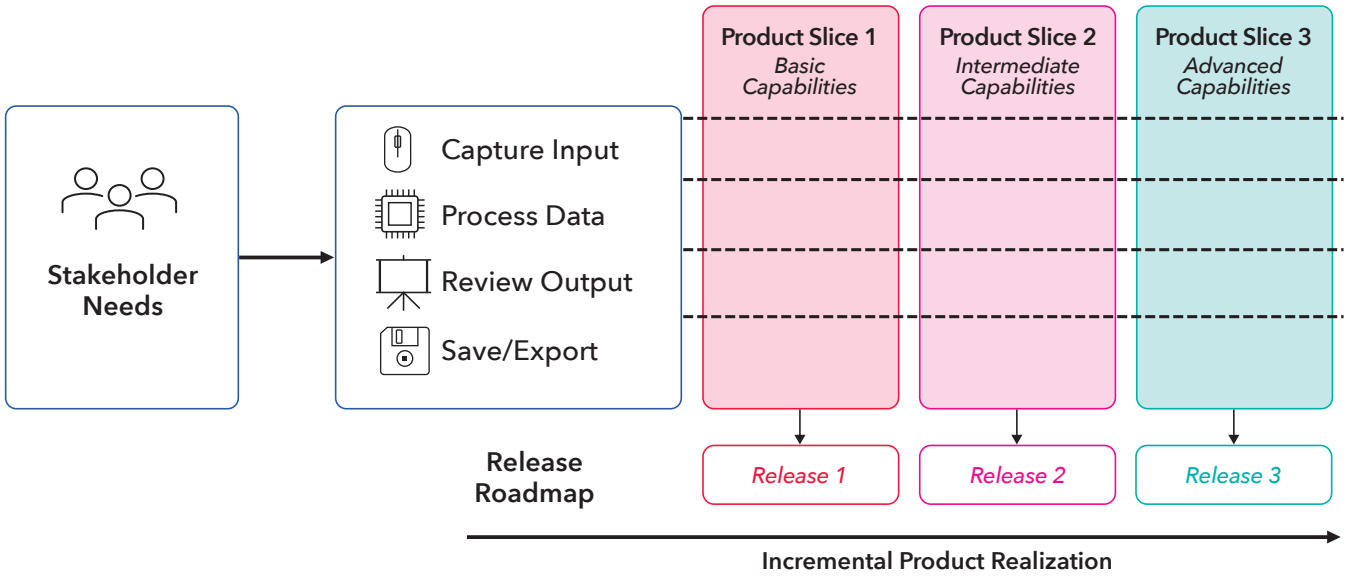


Figure 2. Product-level slices as end-to-end capability increments

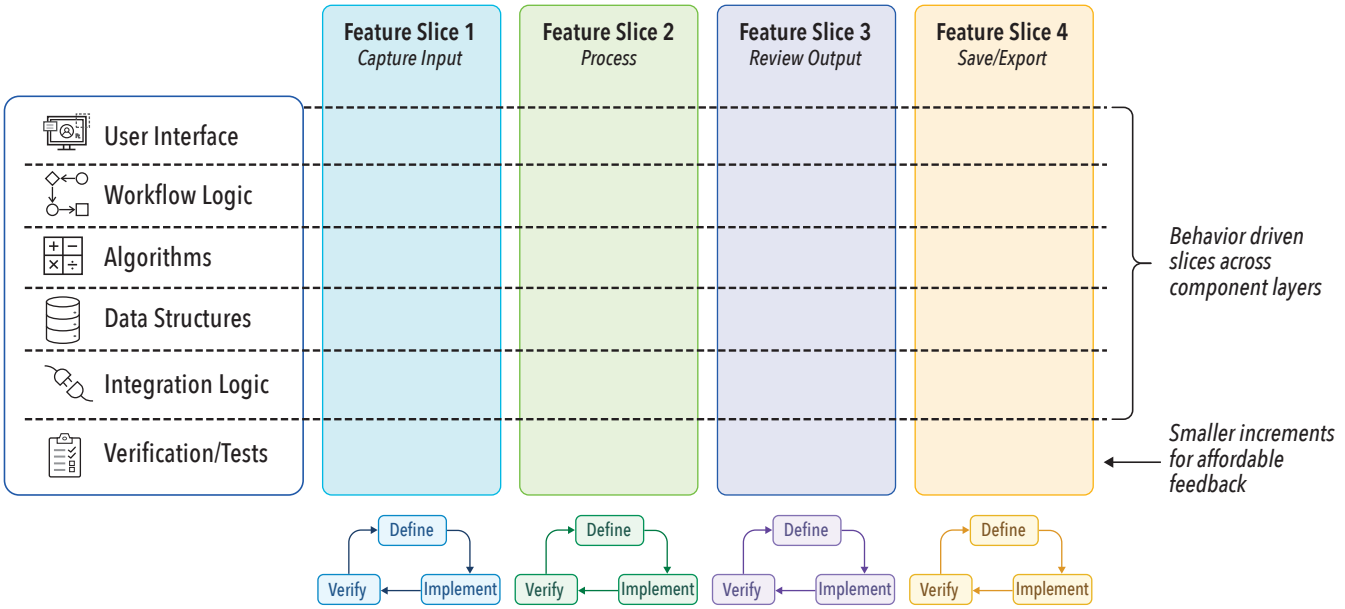


Figure 3. Feature-level slices as implementable end-to-end units

This approach reframes decomposition not as a hierarchy of components, but as a network of behaviorally complete slices, each representing a traceable path from stakeholder intent to system implementation.

MAPPING THE V-MODEL TO SLICING

The relationship between the V-model and slicing becomes clear when examining how each stage of decomposition corresponds to slicing constructs.

At the top of the V-model, stakeholder needs define the desired capabilities of the system. In a slicing framework, these map directly to product-level slices, each representing a deliverable capability that can be independently realized and verified.

Moving down the V-model, system functions emerge as refinements of stakeholder intent. These align with feature-level slices, which encapsulate the functional behaviors required to enable each capability. Unlike traditional functional decomposition, feature slices are not confined to a single subsystem; they intentionally span the architecture to preserve end-to-end coherence.

At the lowest level of the V-model, component definitions describe the structural elements of the system. In slicing, this level corresponds to slice implementation details, where specific elements—such as services, data models, and algorithms—are defined in the context of a slice rather than in isolation.

This mapping preserves the hierarchical intent of the V-model while introducing a critical shift: decomposition is anchored to behavior rather than structure.

MAINTAINING END-TO-END FUNCTIONAL INTEGRITY

One of the most significant advantages of slicing is its ability to maintain end-to-end functional integrity throughout the development lifecycle. Because each slice represents a complete path through the system—from user interaction to backend processing—teams are continuously working with integrated functionality rather than isolated components.

This has profound implications for systems engineering. Integration is no longer a late-stage activity but an inherent property of how work is structured. As a result, many of the risks traditionally associated with system integration—interface mismatches, incomplete assumptions, and emergent defects—are surfaced earlier and addressed incrementally.

In this sense, slicing can be seen as a mechanism for bringing the right side of the V-model closer to the left, collapsing the gap between definition and verification.

RESPONDING TO DYNAMIC AND UNCERTAIN ENVIRONMENTS

Software-intensive systems seldom evolve in perfectly stable conditions. Requirements mature as users interact with early system behavior, integration assumptions change as interfaces are exercised and design constraints become clearer as the system is built.

Slicing helps mitigate this uncertainty by shortening the feedback cycle between definition, implementation, integration, and verification. Because each slice represents an end-to-end behavior, teams can evaluate whether a meaningful system outcome is emerging correctly before the full system is complete. This creates opportunities for affordable course correction: requirements can be refined, interface assumptions can be tested, and verification evidence can be generated incrementally rather than deferred until late-stage integration.

In this way, slicing provides a practical mechanism for engineering in dynamic environments. It does not eliminate uncertainty, but it makes uncertainty visible earlier and at a smaller scale. For systems engineers, this is the critical value–slicing turns evolving knowledge into structured feedback loops that preserve traceability while allowing the system design to adapt.

ANCHORING DECOMPOSITION TO SYSTEM BEHAVIOR

Traditional component-based decomposition often reflects organizational structure rather than system behavior. Teams own modules, and system functionality is distributed across these ownership boundaries. While efficient from a resource allocation perspective, this approach can obscure the actual behavior of the system as experienced by the user.

Slicing inverts this paradigm by anchoring decomposition to system behavior. Each slice is defined by what the system must do—not by where that functionality resides. This ensures that the decomposition remains aligned with real-world usage, clinical workflows, and user interactions.

For systems engineers, this shift reinforces a core principle: the system should be understood and structured in terms of its behavior and purpose, not merely its architecture.

PRESERVING TRACEABILITY ACROSS THE LIFECYCLE

Traceability is a cornerstone of the V-model, particularly in regulated environments where demonstrating alignment between requirements and verification is essential. However, maintaining traceability in complex software systems is often burdensome, as relationships between requirements, design elements, and tests become increasingly fragmented.

Slicing simplifies this challenge by embedding traceability within each slice. Because a slice encapsulates stakeholder intent, functional behavior, and implementation details, it naturally forms a traceable unit. Requirements, acceptance criteria, and verification artifacts can all be linked within the context of the slice.

This approach transforms traceability from a documentation exercise into a structural property of the system decomposition, making it more robust and easier to maintain.

ENABLING INCREMENTAL VERIFICATION: MICRO-V CYCLES

Perhaps the most compelling parallel between slicing and the V-model lies in the concept of incremental verification. In the traditional V-model, verification occurs at defined stages—unit testing, integration testing, system testing—often after substantial portions of the system have been developed.

Slicing enables a more granular approach, where each slice undergoes its own miniature V-cycle, or "micro-V." Within each slice, requirements are defined, implemented, and verified in a tightly coupled loop. This allows for continuous validation of system behavior as it is developed.

The result is a development process that retains the rigor of the V-model while achieving the responsiveness of iterative methodologies. Verification is no longer deferred; it becomes an ongoing activity embedded within each increment of work.

IMPLICATIONS FOR MODERN SYSTEMS ENGINEERING

The alignment between slicing and the V-model offers a pathway for systems engineering to remain relevant in an era of increasingly complex software systems. Rather than abandoning established frameworks, slicing demonstrates how their core principles can be reinterpreted and operationalized in a modern context.

For organizations developing software-intensive systems—particularly those subject to regulatory scrutiny—this approach provides several key benefits:

- Reduced integration risk through continuous end-to-end realization
- Improved alignment between system design and user workflows
- More maintainable traceability structures
- Earlier and more frequent verification cycles.

For systems engineers, the value of slicing is not limited to software decomposition; it also provides a way to coordinate system behavior across software, hardware, interfaces, and verification activities that mature at different rates. Hardware elements often progress through longer lead-time activities, physical prototyping constraints, supplier dependencies, manufacturing considerations, and formal qualification activities that may not move at the same cadence as software. The value of slicing is that it provides a behavioral structure for coordinating these different tempos. A slice can define the end-to-end system behavior that software is intended to enable, while also making visible the hardware interfaces, assumptions, constraints, and dependencies that must be satisfied for that behavior to be realized.

This is especially important in systems where software and hardware mature at different rates. Even when the hardware design cannot change as rapidly as software, slicing can help teams test assumptions earlier through simulations, prototypes, interface definitions, test fixtures, representative data, or incremental integration environments. In this sense, slicing does not force hardware into a software cadence; rather, it gives systems engineers a way to align software iteration with hardware maturity, identify integration risks earlier, and preserve traceability between system behavior, physical constraints, and verification evidence.

Ultimately, slicing enables teams to realize the intent of the V-model more effectively, ensuring that decomposition, integration, and verification remain tightly coupled throughout the lifecycle.

CONCLUSION

The challenge of decomposing complexity in modern software systems is not merely a technical problem—it is a systems engineering problem. The V-model provides a powerful conceptual foundation, but its effectiveness depends on how decomposition is executed in practice.

Slicing offers a complementary approach that aligns closely with the principles of the V-model while addressing the realities of contemporary software development. By anchoring decomposition to system behavior, maintaining end-to-end integrity, and enabling incremental verification, slicing transforms the V-model from a static framework into a dynamic, executable strategy.

In doing so, it provides a compelling vision for the future of systems engineering—one where rigor and agility are not in conflict, but are instead achieved through thoughtful alignment of structure, behavior, and verification. ■

REFERENCES

- Boehm, B. 2006. "A View of 20th and 21st Century Software Engineering." ICSE 2006: Proceedings of the 28th International Conference on Software Engineering.
- Bogard, J. 2018. Vertical Slice Architecture. https://www.jimmybogard.com/vertical-slice-architecture/.
- Forsberg, K., and H. Mooz. 1991. "The Relationship of System Engineering to the Project Cycle." National Council on Systems Engineering (NCOSE) and American Society for Engineering Management (ASEM), Chattanooga, US-TN, 21-23 October.
- INCOSE. 2015. *Systems Engineering Handbook*.
- ISO 14971: Medical Device Risk Management.
- IEC 62304: Medical Device Software Lifecycle.
- ISO 9241-210: Human-Centered Design.
Leffingwell, D. 2011. *Agile Software Requirements*. Pearson Education, Inc.
- Rechtin, E., and M. Maier. 2009. *The Art of Systems Architecting*. CRC Press.

ABOUT THE AUTHOR

Shreya Sridhar is a lead technical systems engineer at Medtronic, where she leads the development of software-intensive medical device systems with a focus on system architecture, risk management, and verification strategy. Her work centers on bridging systems engineering principles with modern software development practices, particularly in regulated environments. She has developed a structured approach to system decomposition—two-level vertical slicing—to improve traceability, integration, and incremental verification in complex systems. Shreya is an active member of INCOSE and serves on the board of directors for the INCOSE New England Chapter, where she contributes to advancing systems engineering practices in healthcare.

Agile Systems Engineering in the Age of AI: Architecting for Dynamic and Uncertain Environments

Robin Yeman, robinyeman@gmail.com

■ **ABSTRACT**
Dynamic operational environments and accelerating technological disruption are reshaping how systems engineering must be practiced. Traditional lifecycle models assume stability in requirements and context; however, modern socio-technical systems particularly those incorporating artificial intelligence operate under persistent uncertainty, rapid feedback cycles, and evolving mission demands. This article argues that systems engineering must shift from a control-centric discipline to one grounded in disciplined adaptability. Integrating agile principles, model-based systems engineering (MBSE), modular open systems architecture (MOSA), and AI-enabled capabilities, the paper proposes an adaptive systems engineering operating model for sustained mission relevance. It differentiates the constraints of hardware, software, and AI-driven systems, examines AI as both engineering tool and system component, and outlines infrastructure requirements including digital threads, continuous validation architectures, and adaptive governance. A cross-industry aerospace case study demonstrates how modularity, concurrent engineering, and digital twins enable iterative delivery in safety-critical domains. Practical guidance is provided for systems engineering practitioners seeking to anticipate and respond effectively to dynamic and uncertain operating environments.

INTRODUCTION

A dynamic environment is one in which the state of the operating domain evolves over time, often due to changing external conditions, system interactions, or agent behaviors. In such environments, the system's future context can shift in ways that affect objectives, constraints, or available resources (Dove 2023). Modern engineered systems no longer operate in static contexts. Instead, they function within dynamic environments where conditions shift continuously due to technological evolution, economic forces, and mission demands. This unpredictability requires systems and the methods used to engineer them to adapt quickly to maintain performance and relevance. At the same time, *uncertainty* permeates both the external environment and the systems themselves. Uncertain environments are those in which key variables and interactions are only partly known or cannot be precisely forecast, compelling engineers to anticipate unknowns rather than rely on fixed specifications. Such uncertainty arises from technological disruptions, emergent behaviors in complex systems, incomplete data, and changing stakeholder needs all of which challenge traditional, plan-driven engineering models. These challenges are particularly acute in large, software-intensive systems and cyber-physical systems where software and AI components dominate functionality. These systemic pressures are perhaps most visible in large-scale government acquisition programs where the cost of failure is high, and the pace of change is accelerating. The U.S. Department of Defense (DoD) struggles to deliver capabilities that meet contemporary threats and mission needs within acceptable cost and schedule bounds. Recent Government Accountability Office assessments show that major defense acquisition programs now average

nearly *12 years* to reach initial capability, an outcome that is increasingly mismatched with the pace of operational and technological change (Office, U.S. Government Accountability Office 2025). In response, engineering organizations are attempting to adopt agile and iterative practices that inherently embrace change and uncertainty. Agile development emphasizes incremental delivery, frequent feedback, and adaptation mechanisms that help manage evolving requirements and dynamic contexts (Zimmerman 2019). However, adoption remains uneven, and measurement challenges persist even where agile is practiced (Borg 2022). Artificial intelligence further complicates the engineering landscape by introducing systems whose behavior can be probabilistic, data-dependent, and non-deterministic. AI components require new modeling, assurance, and validation approaches to ensure they behave reliably across a wide range of operational conditions and data distributions (Adeyeye 2024). Managing this uncertainty pushes traditional systems engineering practices beyond static analyses and linear lifecycle models. Taken together, agile principles, AI capabilities, and systems engineering disciplines offer complementary mechanisms for *anticipating change, adapting designs*, and *continuously delivering validated value* making them essential for engineering in dynamic, uncertain environments.

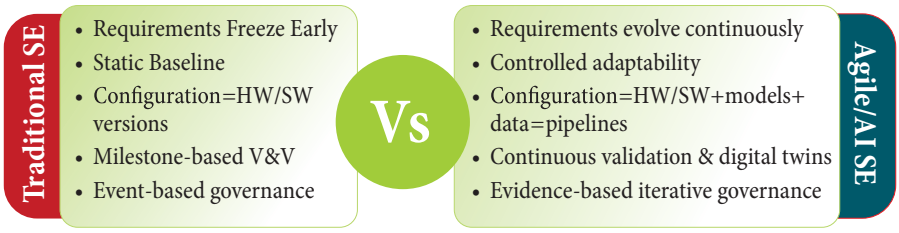


Figure 1. Shift from traditional systems engineering to agile/AI systems engineering

UNDERSTANDING THE NATURE OF DYNAMICS AND UNCERTAINTY

Types of Uncertainty

Uncertainty in systems engineering arises from both incomplete knowledge and inherent variability. Epistemic uncertainty reflects gaps in knowledge, unclear stakeholder needs, immature technologies, or insufficient modeling fidelity. Aleatory uncertainty, by contrast, stems from inherent variability in operating conditions such as environmental disturbances, user behavior, or network latency (Zarei 2024). Both forms complicate prediction and decision-making across the lifecycle. This manifests as requirement volatility the degree to which requirements change during development. Volatility is often driven by evolving missions, stakeholder reprioritization, or even simple misinterpretation. Another critical form of uncertainty is emergent behavior. In complex socio-technical systems, interactions among components produce behaviors not predictable from the properties of individual elements. Emergence becomes especially pronounced in systems-of-systems, where components cross both organizational and technical boundaries (Zarei, Expert judgment and uncertainty in sociotechnical systems analysis 2024). Finally, operational unpredictability reflects uncertainty in how systems will be used or challenged once deployed. In adversarial domains such as cybersecurity or defense operations, intelligent opponents adapt in response to system behavior, creating feedback loops that invalidate static assumptions (Angelelli 2022).

Sources of Dynamics

Rapid technology evolution continuously reshapes what is technically feasible, affordable, and operationally advantageous. Advances in computing architectures, communications infrastructure, materials science, autonomy, cybersecurity, and digital engineering alter performance ceilings and design trade spaces over time. Systems conceived under one technological paradigm may face obsolescence pressures before deployment (Mohamed 2024). Continuous deployment ecosystems shorten feedback loops, enabling more frequent updates. DevSecOps pipelines and cloud-based infrastructures allow capabilities to evolve incrementally (Claps 2015). While this increases responsiveness, it also creates persistent baseline change and requires architectures designed for modularity, interface stability, and automated integration. Adversarial environments introduce adaptive dynamics. In defense, cybersecurity, and competitive commercial domains, opponents and competitors respond strategically to deployed capabilities (Tao 2024). This creates feedback loops in which system effectiveness depends not only on internal design quality but also on how external actors adapt. Finally, technology is currently outpacing regulatory policy in safety, cybersecurity, privacy, and compliance requirements which creates regulatory flux where constraints are continuously altering how systems must operate (Shandilya 2024). Certification standards, export controls, and AI governance frameworks continue to mature, requiring systems to remain adaptable to new oversight expectations throughout their lifecycle.

Implications for Systems Engineering

Persistent uncertainty and dynamics reshape how systems engineering must be practiced. Traditional lifecycle models assume uncertainty declines over time and that architectures can be stabilized early through disciplined requirements decomposition. In dynamic environments, however, uncertainty persists beyond deployment, and assumptions degrade faster than development cycles complete. The result is baseline instability. Freezing requirements too early produces brittle systems; allowing uncontrolled change undermines integration. Systems engineers must shift their approach as shown in Figure 1 from traditional to adaptive systems engineering. They are required to move from pursuing permanent baselines to managing disciplined adaptability stabilizing interfaces and architectural guardrails while enabling component evolution (de Oliveira 2025). This shift increases configuration management complexity. Modern configuration items extend beyond hardware and software to include models, datasets, training artifacts, and infrastructure definitions that evolve asynchronously. Effective control requires a digital thread, automated traceability, and version management spanning requirements, models, code, test artifacts, and data lineage (Akerele 2024). Validation complexity also rises. Emergent behaviors and probabilistic elements challenge deterministic verification. Milestone-based qualification is insufficient when systems update continuously. Continuous integration must be paired with continuous validation, automated regression, simulation, and digital twins that assess performance across changing operational envelopes (Rosenberg 2017). Consider an autonomous aerial system: flight software updates quarterly, sensors refresh on separate hardware cycles, perception models retrain as environments shift, and regulatory requirements evolve. Without traceable updates and continuous validation, technically sound changes can degrade safety or mission performance. Finally, governance tension emerges. Sequential approval processes assume stability, while dynamic systems demand iterative evidence and incremental assurance. Systems engineering therefore shifts from enforcing stability to architecting resilience, traceability, and controlled adaptability ensuring agility does not compromise coherence, safety, or mission assurance (Yeman 2025).

AGILE ENGINEERING AS A STRATEGY (NOT A METHOD)

Agile as a Strategic Posture

Agile engineering is often reduced to team-level mechanics sprints, standups, and burn-down charts. In dynamic environments, however, agile must be understood as a strategic posture, not a procedural toolkit. At its core, agile emphasizes responsiveness, incremental value delivery, and structured learning loops that progressively reduce uncertainty through evidence and feedback (Hubbart 2025). A strategic agile posture operates across multiple planning horizons. Near-term horizons focus on sprint execution and immediate deliverables. Mid-term horizons address capability increments, architectural runway, and integration milestones. Long-term horizons preserve mission intent, system vision, and investment alignment (Johnson 2023). Rather than locking down detailed plans years in advance, organizations maintain intent at the strategic level while allowing execution detail to adapt as knowledge evolves. This multi-horizon approach prevents two common failures: short-termism and rigidity. Without long-range intent, rapid iteration can drift from mission outcomes. Without short-cycle adaptation, long-range plans become brittle under changing conditions. Agile strategy therefore synchronizes horizons linking backlog priorities to capability roadmaps and architectural evolution. Agile also reframes uncertainty. Instead of attempting to eliminate it before execution, Agile treats uncertainty as information to be discovered (Pendzik 2023). Incremental releases are not merely delivery events; they are experiments that generate operational insight. Feedback loop technical, operational, and stakeholder-driven become central governance mechanisms. This stands in contrast to procedural "agile theater," where ceremonies are adopted but processes, and architectural control remain static. In such environments, teams sprint locally while enterprise decision cycles remain slow (Berry 2025). True agility requires alignment of architecture, implementation, validation, and oversight with iterative delivery across multiple planning horizons.

Agile Systems Engineering

For systems engineers, agile must extend beyond software increments to the technical baseline itself.

Continuous architecture replaces early design freeze with evolving design intent. Stable interfaces and architectural guardrails are established early, while detailed implementations mature incrementally. Maintaining architectural runway enables adaptation without destabilizing the system (Huss 2023).

Incremental verification and validation (V&V) embed testing within development. Evidence of performance, safety, and integration readiness is produced continuously through automated regression, model-based simulation, and integration environments (Dubey 2026). Backlog-driven requirements evolution treats requirements as living artifacts. System-level intent decomposes into prioritized backlog items linked to models, acceptance criteria, and test cases, preserving traceability while enabling responsiveness (Yeman 2025a). DevSecOps integration connects development, security, and operations. Automated pipelines support frequent deployment, continuous monitoring, and rapid feedback from operational environments into engineering decisions. Together, these practices sustain system coherence while enabling change.

Lessons Learned from the field

The most significant lesson from practice is straightforward: agile as a software-only approach does not work in complex systems environments. When agile is confined to development teams but not implemented at the system level across architecture, integration, cybersecurity, contracting, and governance it produces local speed but enterprise friction (Dahlke 2023).

Fictional Vignette

An agile software team supporting a mission-critical communications platform completes a sprint early, delivering a functional module that enables encrypted message routing. They log the release in their CI/CD dashboard, mark the user story complete in Jira, and prepare for a demo. However, the systems engineering team, operating in Cameo, does not update the interface control document (ICD) to reflect the change so downstream hardware teams continue building against an outdated model. Meanwhile, the cybersecurity team is unaware of the sprint delivery because it hasn't been submitted as part of the ATO package, which is still in draft. The program manager, expecting formal milestone documentation, is unaware of the release because it doesn't align with the integrated master schedule (IMS). Finally, the contracting officer cannot authorize further work because the functional delivery was not part of the contract line items (CLINs), which are milestone-bound. The result: functioning software sits undeployed, feedback is delayed, and momentum is lost not because the teams aren't working hard, but because they're working in different worlds.

This vignette exposes three recurring pitfalls.

1. **Software-only agility creates structural misalignment:** Without synchronized architecture, contracting, compliance, and integration processes, iterative software delivery cannot translate into operational capability.
2. **Rapid iteration without coordinated system oversight accumulates architecture debt:** Interfaces, data models, and integration assumptions diverge across teams, increasing long-term complexity and rework.
3. **Scaling agile requires system level synchronization:** Multiple teams operating at different cadences demand shared architectural vision, aligned planning horizons, and governance mechanisms that accept incremental evidence rather than milestone-bound artifacts.

Agile succeeds only when implemented as a system-level strategy. It must integrate digital engineering, configuration control, accreditation, funding mechanisms, and oversight processes into the same iterative cadence. Otherwise, agility becomes theater motion without mission impact.

AI AS BOTH ENABLER AND ENGINEERING CHALLENGE

AI as an Engineering Tool

As a tool, AI is a serious systems engineering enabler. AI enhances analytical speed and insight across the lifecycle in multiple areas, examples include performing trade studies, system modeling, risk analytics, and requirements management. AI-assisted trade studies allow engineers to explore expansive design spaces quickly, identifying non-obvious trade-offs and sensitivities (Li 2026). In the past we would be limited in our analysis due to time constraints, but AI can review large quantities of data faster than a human ever could. Optimization algorithms can surface Pareto-optimal architectures far faster than manual methods, strengthening early concept decisions under uncertainty.

In model-based systems engineering (MBSE) environments, AI can detect model inconsistencies, suggest interface refinements, cluster related requirements, and highlight architectural patterns. In one case study completed by Ford Motor Company, when the integrated AI into their MBSE process they saw improvement in requirement clarity and detection of inconsistencies in the models during early design (de Oliveira A. H. 2025). These capabilities augment, not replace, systems engineers by

reducing cognitive load and accelerating analysis. Leveraging AI to identify early risk indicators has shown to improve early decision making. Predictive risk analytics use historical cost, schedule, defect, and performance data to detect patterns that precede integration failures or performance degradation. Early risk signals enable proactive mitigation rather than reactive correction (Diameh 2025). Safety critical systems require bidirectional traceability. In dynamic environments this can be very challenging. AI proven that it can support automated requirement traceability, using natural language processing to link evolving requirements to models, code, and test artifacts. As requirements change, trace relationships can be updated systematically, preserving coherence in dynamic environments. At Meta Reality Labs, we demonstrate how AI-driven traceability enables faster release cycles while maintaining the highest standards of safety and regulatory adherence (Ginsbourg 2026). When AI is aligned with system-level processes, these capabilities compress feedback loops and improve decision quality.

AI as a System Component

AI as a system component, when embedded in delivered systems, AI introduces new engineering challenges such as managing non-deterministic behavior, data drift, continuous model recalibration, and ethical considerations (Hinrichs 2020). AI systems often exhibit non-deterministic behavior. Performance must be assessed statistically rather than through binary pass/fail testing. Engineers must define acceptable error rates and confidence levels rather than fixed deterministic outputs. AI performance is also data dependent. Changes in operational inputs can create data drift, degrading model effectiveness even when code remains unchanged. Consequently, training datasets, feature pipelines, and model weights become configuration-controlled assets. Many AI systems require continuous training or recalibration, extending the lifecycle beyond deployment. Model updates must follow disciplined processes aligned with safety and compliance requirements. Finally, AI raises ethical and assurance implications, including bias, explainability, accountability, and robustness to adversarial manipulation. These considerations expand systems engineering beyond technical integration to include transparency and oversight mechanisms.

Engineering AI-Enabled Systems

Engineering AI-enabled systems requires adapting architecture and governance to probabilistic behavior and continuous evolution. Engineers must define probabilistic performance envelopes that describe acceptable behavior across operational conditions. Runtime monitoring architectures detect performance degradation or drift and trigger alerts or fallback mechanisms. MLOps integrated with systems engineering governance ensures that retraining and model updates follow configuration control, validation, and approval processes consistent with safety and mission assurance. AI cannot operate outside established systems engineering discipline. Digital twins enable validation under uncertainty by simulating rare or adversarial scenarios prior to field updates. Finally, context engineering ensures AI components are designed with awareness of mission intent, human operators, and environmental constraints.

Fictional Vignette

Consider an AI-enabled intelligence, surveillance, and reconnaissance (ISR) platform using computer vision to identify objects of interest. As environmental conditions shift lighting, terrain, adversary camouflage model accuracy begins to degrade. Runtime monitoring detects increased classification error rates. The digital twin environment is used to test retrained models against expanded datasets before deployment. The updated model is processed through an MLOps pipeline integrated with configuration control and safety review. By combining probabilistic performance thresholds, continuous validation, and governed updates, the system adapts without compromising mission assurance.

HARDWARE VS SOFTWARE VS AI-DRIVEN SYSTEMS

Dynamic and uncertain environments affect hardware, software, and AI-driven systems in fundamentally different ways. Effective systems engineering requires recognizing and reconciling these distinct tempos of change.

Hardware Constraints

Hardware engineering is constrained by physical irreversibility. Once components are fabricated, integration paths narrow and design changes become expensive. Tooling, certification testing, and environmental qualification introduce long lead times that cannot be compressed easily. In addition, supply chain volatility including material shortages, vendor dependencies, and geopolitical risk can alter cost and schedule assumptions unexpectedly. Hardware programs must therefore anticipate uncertainty before production decisions are locked in. Hardware development often involves long integration cycles, especially in safety-critical or mission-critical systems. Testing requires physical prototypes, environmental chambers, and system-level integration events. These realities demand early architectural foresight, stable interfaces, and disciplined change control.

Software Agility

Software operates under a different dynamic. Through continuous integration and continuous deployment (CI/CD) pipelines, software can evolve incrementally and frequently. Requirements can be reprioritized, defects corrected, and enhancements fielded with comparatively low latency. The marginal cost of iteration is far lower than in hardware. Code can be refactored or redeployed without physical rework. However, agility without architectural discipline risks instability. Software must still conform to interface definitions and system-level constraints, particularly when hosted on hardware platforms with slower refresh cycles.

AI-Driven Systems

AI-driven systems introduce additional complexity beyond traditional software. They exhibit data lifecycle complexity, where datasets, model weights, feature pipelines, and retraining processes become configuration-controlled assets. Performance is tied to data quality and representativeness. AI systems also require continuous validation. Because behavior is probabilistic, validation must assess statistical performance across evolving operational conditions. Over time, behavioral drift can occur as environmental conditions change. Even without code modification, performance may degrade. This necessitates runtime monitoring and structured retraining processes integrated with governance.

Hybrid System Strategy

Most modern systems combine hardware, software, and AI components. Effective strategy depends on deliberate synchronization points across development cadences. Hardware refresh cycles, software releases, and AI retraining must align at defined integration events. Architectural modularity and stable interfaces allow components to evolve independently without destabilizing the whole. Finally, digital prototyping before hardware commitment using simulation and digital twins reduces risk by validating software and AI behaviors prior to irreversible physical production decisions. Key features of our hybrid approach are illustrated below in Table 1.

Table 1. Hybrid strategies reconcile hardware stability with software and AI adaptability through synchronization, modularity, and digitally enabled foresight

	Feature	Benefit
1.	Defined Synchronization Points	Aligns different development tempos; prevents software or AI updates from outpacing hardware readiness; reduces integration surprises.
2.	Architectural Modularity	Enables independent evolution of hardware, software, and AI components without destabilizing the overall system. Reduces rework and integration risk.
3.	Stable Interface Control Documents (ICDs)	Preserves system coherence while allowing internal component changes; protects downstream teams from volatility.
4.	Digital Prototyping Before Hardware Commitment	Validates software and AI functionality in simulation before irreversible manufacturing decisions; lowers cost of change.
5.	Digital Twin Environments	Enables early testing of edge cases, rare events, and AI performance envelopes under uncertainty. Improves confidence prior to deployment.
6.	Integrated Configuration Management Across Domains	Maintains traceability for hardware versions, software builds, datasets, and model weights; prevents drift between system elements.
7.	Coordinated Planning Horizons	Balances immediate iteration with long-term architectural intent; reduces short-term optimization at the expense of system integrity.
8.	Governed MLOps Integration	Ensures AI retraining and updates follow disciplined SE processes; supports safety, compliance, and mission assurance.

INFRASTRUCTURE TO SUPPORT ADAPTIVE SYSTEMS ENGINEERING

Engineering in dynamic and uncertain environments is not simply a matter of adopting agile practices or integrating AI tools. It requires a supporting infrastructure technical, organizational, and governance-based that enables anticipation of change and disciplined response to it. Without this infrastructure, adaptability becomes improvisation rather than engineered resilience.

Digital Engineering Ecosystems

At the core of adaptive systems engineering is a digital engineering ecosystem. Model-based systems engineering (MBSE), integrated data environments, and digital threads create continuity across requirements, architecture, implementation, testing, and sustainment. In dynamic environments, traceability is not merely documentation, it is a mechanism for controlled change. Digital threads link evolving requirements to models, code, test artifacts, datasets, and deployed configurations. When change occurs whether driven by mission evolution, technological advancement, or regulatory shift the impact can be assessed across the system rapidly. This reduces the risk of unintended consequences and accelerates informed decision-making. Digital twins further enhance anticipation by allowing engineers to simulate new conditions before committing to deployment. Edge cases, adversarial scenarios, and performance boundaries can be explored in a controlled environment, reducing uncertainty before field exposure.

Continuous Validation Architecture

Traditional validation approaches assume stable baselines and milestone-based certification. Dynamic systems demand continuous validation architectures. Integration pipelines must embed automated testing, simulation, and performance monitoring to generate ongoing evidence of compliance and mission suitability. For AI-enabled systems, runtime monitoring becomes part of the system architecture itself. Telemetry feeds back into analytics environments that detect drift, degradation, or anomalous behavior. This closes the loop between operations and engineering, enabling proactive adaptation rather than reactive correction. Continuous validation also supports incremental governance. Rather than waiting for major review events, decision-makers can evaluate rolling evidence packages aligned with iterative capability releases.

Adaptive Governance

Infrastructure for anticipation is incomplete without governance reform. Many organizations attempt technical agility within rigid milestone-based funding and approval frameworks. This creates friction between iterative delivery and static oversight. Adaptive governance introduces risk-informed, evidence-based decision cycles aligned with iterative development. Contracts and funding mechanisms must allow modular delivery rather than single milestone-bound releases. Oversight bodies must evaluate continuous evidence rather than static documentation packages. This does not imply reduced rigor. On the contrary, adaptive governance can increase assurance by relying on real performance data and traceable digital artifacts rather than paper compliance alone.

Human and Organizational Alignment

This system includes people and requires cross-domain literacy systems engineering, software engineering, hardware engineering, AI/ML, cybersecurity, contracting is essential. The “frozen middle” problem often emerges when mid-level managers are incentivized to protect stability rather than enable adaptation. Organizations must redefine roles to support synchronization across hardware, software, and AI cadences. Systems engineers increasingly serve as integrators of change balancing architectural intent, operational feedback, and governance constraints. In dynamic environments, anticipation is enabled by digital continuity, continuous validation, adaptive governance, and organizational alignment. Response is enabled by modular architecture, synchronized cadences, and disciplined configuration control. Together, these elements transform systems engineering from a stability-centric discipline into one capable of sustaining mission relevance under persistent uncertainty.

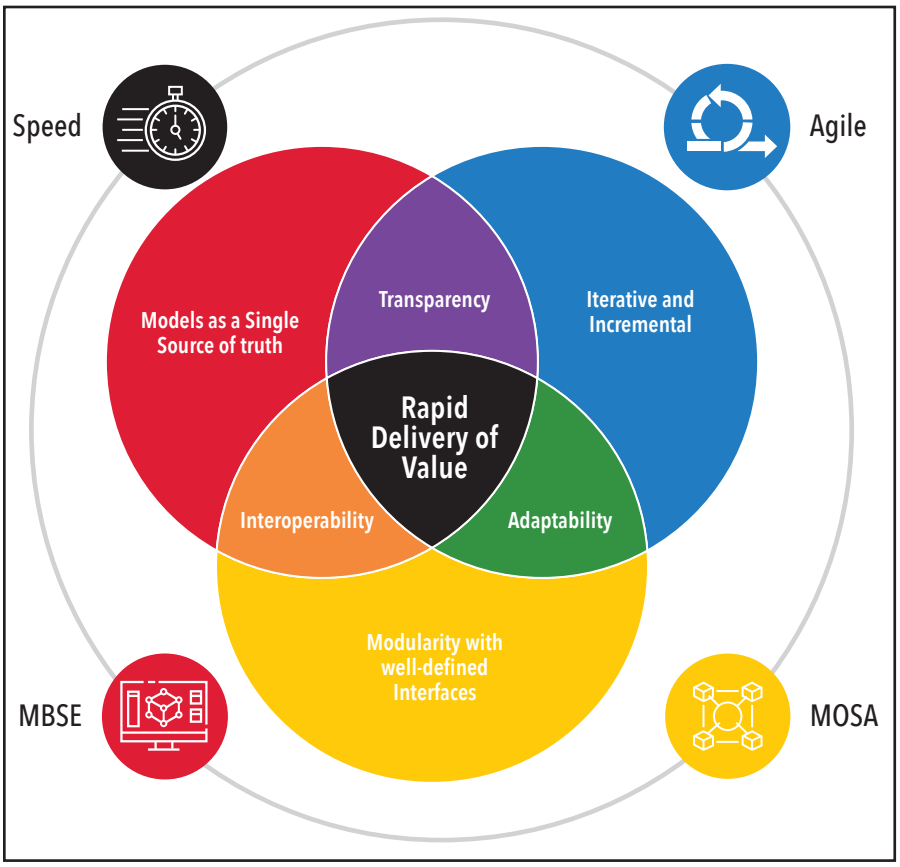


Figure 2. Proposed adaptive systems engineering model

Adaptive Systems Engineering Model

The adaptive systems engineering operating model illustrated in Figure 2, integrates agile delivery, model-based systems engineering (MBSE), and modular open systems architecture (MOSA) within an open, digital ecosystem to enable rapid and resilient delivery of value. Agile provides iterative and incremental execution mechanisms that accelerate learning and responsiveness. MBSE establishes models as the single source of truth, creating transparency and traceability across requirements, architecture, and validation artifacts. MOSA enforces modularity with well-defined interfaces, preserving interoperability and enabling components to evolve independently without destabilizing the overall system. At their intersections, transparency, adaptability, and interoperability emerge as system-level properties rather than isolated practices. Artificial intelligence acts as a cross-cutting accelerator enhancing trade studies, automating traceability, detecting risk patterns, and enabling adaptive decision-making across the lifecycle. The outer emphasis on speed reflects the model's purpose: sustaining mission relevance in dynamic and uncertain environments by synchronizing planning horizons, digital engineering infrastructure, and incremental delivery within a disciplined architectural framework.

AGILE + AI IN A DYNAMIC MISSION CONTEXT

Case Study: GEC-Marconi Aerospace

The aerospace sector represents one of the most demanding engineering environments: high system complexity, long development cycles, strict regulatory oversight, and evolving operational requirements. Traditional aerospace development has historically relied on sequential validation models, emphasizing extensive upfront design and rigid phase gates. However, increasing global competition and technological volatility have pressured organizations to reduce cycle time without sacrificing reliability (Patel 2025).

The Shift

One of the most significant changes was the adoption of concurrent engineering, integrating design, validation, and manufacturing preparation in parallel rather than sequentially (Patel 2025). This reduced product development cycles while preserving traceable quality assurance processes. From an adaptive systems engineering perspective, this represents a move from milestone-bound stability toward disciplined adaptability. Instead of waiting for late-stage validation, subsystems were iteratively refined through overlapping development activities, reducing rework, and integration risk.

Digital Twins and Continuous Monitoring

GEC-Marconi also leveraged digital twin technologies to enable virtual simulations and real-time performance monitoring (Patel 2025). Digital representations of systems allowed predictive maintenance, operational optimization, and earlier detection of potential failures. This aligns directly with the continuous validation architecture described earlier in this paper. Digital twins reduced dependence on purely physical prototyping and enabled earlier risk discovery particularly important in aerospace, where hardware irreversibility and certification constraints are significant. By shifting validation left through simulation and digital modeling, the organization effectively engineered feedback loops into the lifecycle.

Modular Manufacturing and Interface Stability

Modular manufacturing cells were introduced to create reconfigurable production lines (Patel 2025). Hardware systems were segmented into independently manageable subsystems with defined interfaces. This modularity allowed adaptation to changing customer demands without destabilizing entire platforms. In adaptive systems engineering terms, MOSA principles were operationalized at the manufacturing level. Stable interfaces preserved system coherence while enabling incremental upgrades precisely the synchronization challenge faced in hybrid hardware/software/AI systems.

Results and observed Benefits

The implementation of agile manufacturing principles yielded:

- Shorter development cycles
- Improved supplier coordination
- Enhanced customization capability
- Increased responsiveness to market and technology shifts.

The following challenges were documented:

- Investment in digital infrastructure
- Workforce upskilling requirements
- Cybersecurity concerns associated with cloud-enabled analytics.

These findings reinforce a central argument of this paper: Agile in hardware-intensive domains succeeds only when supported by digital infrastructure, modular architecture, and governance adaptation.

Relevance to Adaptive Systems Engineering

While the original case focuses on hardware manufacturing, its structural lessons extend directly to AI-enabled systems. Digital twins enable probabilistic validation. Modular design supports safe insertion of adaptive capabilities. Concurrent engineering reduces the risk of late-stage integration failure. The aerospace example demonstrates that adaptability in safety-critical domains is achievable not by abandoning rigor, but by restructuring it around iterative validation, modularity, and digitally enabled feedback. In dynamic mission contexts, adaptability must be engineered at the system level. The GEC-Marconi case illustrates those agile principles, when integrated with digital engineering and modular architecture, can coexist with and even enhance high-assurance hardware development.

CONCLUSION

Engineering in dynamic and uncertain environments requires more than incremental improvement to existing processes. It requires a shift in how systems engineering defines its purpose. Traditional systems engineering has often been control-centric focused on stabilizing requirements, freezing baselines, and reducing variance through structured decomposition. In environments characterized by rapid technological evolution, adversarial adaptation, regulatory flux, and AI-driven behavioral variability, stability alone is no longer sufficient. Relevance depends on disciplined adaptability. This paper has argued that agile principles, AI-enabled capabilities, and systems engineering discipline are not competing paradigms but complementary forces. Agile provides responsiveness and structured learning loops. MBSE and digital engineering establish transparency and traceability across evolving artifacts. MOSA enforces modularity and interface stability, enabling components to evolve independently. AI accelerates trade studies, risk detection, and traceability while simultaneously demanding new approaches to validation, configuration control, and governance. When integrated within an adaptive operating model, these elements produce transparency, interoperability, and adaptability system-level properties that enable rapid and resilient delivery of value.

The case study demonstrates that speed alone does not create impact. Local agile success without system-level synchronization results in friction. AI experimentation without architectural discipline increases risk. Hardware constraints ignored in planning create integration delays. Adaptation must be engineered deliberately through synchronization points, modular architectures, digital threads, continuous validation, and governed MLOps pipelines. The role of the systems engineer therefore expands. No longer solely a steward of requirements and baselines, the systems engineer becomes an integrator of change balancing mission intent, evolving technology, operational feedback, and governance constraints. Anticipation becomes as important as verification. Resilience becomes as important as optimization. Continuous validation becomes as important as initial certification.

For practitioners, the call to action is clear:

- Leverage multiple horizons of planning
- Design architectures for change
- Embed feedback loops
- Treat AI as both an accelerator and responsibility
- Move to continuous governance.

Dynamic environments will not slow. Uncertainty will not diminish. The systems engineering community must respond not by abandoning discipline, but by evolving it. Adaptive systems engineering offers a path forward preserving rigor while enabling responsiveness so that engineered systems remain mission-relevant in a world defined by persistent change. ■

REFERENCES

- Adeyeye, O. J. 2024. "Artificial intelligence for systems engineering complexity: a review on the use of AI and machine learning algorithms." *Computer Science & IT Research Journal*, 787–808.
- Akerele, J. I. 2024. "Increasing software deployment speed in agile environments through automated configuration management." *International Journal of Engineering Research Updates*, 7: 028–035.
- Angelelli, L. A.-D. 2022. *Improving Decision Making by Reducing Uncertainty in Complex Systems of Systems*. Retrieved from NATO Publications: STO. nato. int. https://www.sto.nato.int/publications/STOMeeting_Proceedings/STO_Meeting_Proceedings-MSG-143/MP-MSG-143-15.pdf.
- Berry, M. A. 2025. "Exploring the Challenges in Hardware Development Projects using Agile: a Literature Review of Challenges and Research Opportunities." *Australian Journal of Multi-Disciplinary Engineering*, 1–13.
- Borg, M. 2022. "Agility in Software 2.0 – Notebook Interfaces and MLOps with Buttresses and Rebars." *Lean and Agile Software Development - 6th International Conference, LASD 2022, Proceedings* (pp. 3–16). Lecture Notes in Business Information Processing.
- Claps, G. G. 2015. "On the Journey to Continuous Deployment: Technical and Social Challenges Along the Way." *Elsevier*, 21–31.
- Dahlke, P. A. 2023. "Towards Agile Systems Engineering in Early Stage Design for Complex Naval Vessels." *Proceedings of the 25th International Dependency and Structure Modelling Conference (DSM 2023)* (pp. 48–57). Gothenburg, SE: The Design Society.
- de Oliveira, A. H. 2025. "AI-Driven Agile Systems Engineering Approach for Managing Cross-System Interactions." *Procedia Computer Science, Elsevier*, 402–410.
- de Oliveira, A. H. 2025. *Model-Driven Systems Engineering Automation with Artificial Intelligence for Robustly Writing Automotive System Requirements*. SAE Technical Paper.
- Diameh, J. T. 2025. "Integrating AI-Driven Predictive Analytics in Project Risk Management to Optimize Decision-Making and Performance Efficiency." *Int. J. Eng. Technol. Res. Manag*, 373--389.
- Dove, R. A. 2023. "Agile Systems Engineering–Eight Core Aspects." *INCOSE International Symposium* (pp. 823-837). Wiley / INCOSE.
- Dubey, S. 2026. "From Test Case Design to Test Data Generation: How AI is Transforming End-to-End Quality Assurance in Agile and DevOps Environments." *Artificial Intelligence*.
- Ginsbourg, S. 2026. *From Bottleneck to Benchmark: Using AI to Automate the Requirements Traceability Matrix (RTM) for FDA Submissions*.
- Hinrichs, T. A. 2020. "Can AI-based Components be Part of Dependable Systems?" *2020 IEEE Intelligent Vehicles Symposium (IV)*.
- Hubbart, J. 2025. "Lean and Agile Strategic Planning: Reframing Organizational Strategy for Competitive Advantage." *International Journal of Innovation, Management and Technology*, 1-4.
- Huss, M. A. 2023. "An Agile Model-Based Software Engineering Approach Illustrated through the Development of a Health Technology System." *Software*, 234–257.

- Johnson, S. A. 2023. *Industrial DevOps: Build better systems faster*. IT Revolution.
- Li, J.-P. A. 2026. "A Review of AI-Driven Engineering Modelling and Optimization: Methodologies, Applications and Future Directions." *Algorithms*.
- Mohamed, M. G. 2024. "Engineering's Next Leap: How Fourth Industrial Revolution is Shaping the Future of the Industry." *ERU Research Journal*, 970–992.
- Office, U. G. 2025, June 11. *U.S. Government Accountability Office*. (U. G. Office, Producer) Retrieved October 2025, from Weapon Systems Annual Assessment: DOD Leaders Should Ensure That Newer Programs Are Structured for Speed and Innovation: https://www.gao.gov/products/gao-25-107569.
- Office, U. G. (2025, October 5). *Weapon Systems Annual Assessment: DOD Leaders Should Ensure That Newer Programs Are Structured for Speed and Innovation*. Retrieved October 2025, from https://www.gao.gov: https://www.gao.gov/products/gao-25-107569.
- Patel, K. A. 2025. "Agile Hardware Development: A Cross-Industry Exploration for Faster Prototyping and Reduced Time-to-Market." *4th World Conference on Mechanical Engineering*. Indian Institute of Information Technology Design and Manufacturing Kancheepuram.
- Pendzik, M. A. 2023. "Identification and Classification of Uncertainties as the Foundation of Agile Methods." *Proceedings of the Design Society, 3*: 2165–2174.
- Rosenberg, D. A. 2017. "Rapid, Evolutionary, Reliable, Scalable System and Software Development: The Resilient Agile Process." *Proceedings of the 2017 International Conference on Software and System Process*, (pp. 60–69).
- Shandilya, S. K. 2024. "Navigating the Regulatory Landscape." In S. K. Shandilya, *Digital Resilience: Navigating Disruption and Safeguarding Data Privacy* (pp. 127–240). Springer.
- Tao, A. a. 2024. "Dynamics-Aware Adversarial Attack of Adaptive Neural Networks." *IEEE Transactions on Circuits and Systems for Video Technology*, 5505--5518.
- Yeman, R. 2025. The Application of Agile to Large-Scale, Safety-Critical, Cyber-Physical Systems. Colorado State University.
- Yeman, R. 2025a. "Advancing Industrial DevOps: Harnessing Digital Twins and AI for Future-Ready Engineering." *AIAA SCITECH 2025 Forum* (p. 0284). Orlando, US-FL: AIAA.
- Zarei, E. 2024. "Expert Judgment and Uncertainty in Sociotechnical Systems Analysis." In E. A. Zarei, *Safety causation analysis in sociotechnical systems: advanced models and techniques* (pp. 487–530). Springer.
- Zimmerman, P. A. 2019. "Digital Engineering Transformation Across the Department of Defense." *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 325–338.

ABOUT THE AUTHOR

Robin Yeman is a results-focused leader and award-winning author with over twenty-eight years of expertise in systems and software engineering supporting the Department of Defense (DoD), building everything from submarines to satellites. Robin currently, works at Leidos as a senior director with a focus on generative and agentic AI. She earned her PhD at Colorado State University, with research in delivering large-scale, safety-critical cyber-physical solutions using agile and DevSecOps.

Hause continued from page 28

- Hause, M., and A. Korff. 2015. "Using Model-based Product Line Engineering for Decision Making and to Guide Product Development." In Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), NDIA, Novi, MI, 4-5 Aug.
- OMG. 2026. Unified Architecture Framework Specification Version 1.3. (2026). Retrieved 31 March. https://www.omg.org/spec/UAF/1.3/About-UAF/.
- Object Management Group (OMG). 2019. OMG2012-06-01. OMG Systems Modeling Language (OMG SysML™) V1.6. http://www.omg.org/spec/SysML/1.6/PDF/.
- INCOSE PLE Working Group. 2019. Feature-based Systems and Software Product Line Engineering: A Primer. https://connect.incose.org/Pages/Product-Details.aspx?ProductCode=PLE_Primer_2019.
- ISO/IEC 26580:2021. Software and Systems Engineering. Methods and Tools for the Feature-Based Approach to Software and Systems Product Line Engineering. https://www.iso.org/standard/43139.html.
- US DoW. 2026. Modular Open Systems Approach (MOSA), https://www.dsp.dla.mil/Programs/MOSA/.

ABOUT THE AUTHOR

Matthew Hause, SSI distinguished engineer, is a principal consultant at SSI, a chair of the UAF group, and a member of the OMG SysML specification team. He has been developing multi-national complex systems for over 45 years as a systems and software engineer. He worked in the power systems industry, command and control systems, process control, oil and gas, SCADA, military systems, and many other areas. His role at SSI includes consulting, mentoring, standards development, specification of the UAF profile, and training.

Implementing Functional-Outcomes-Driven Tailoring (FODT): A Strategic Roadmap for Multi-Organizational Product Development

Barend Botha, *Chief Engineer*, Halcon, bwbotha@tadhole.com

ABSTRACT
As the complexity of engineered systems continues to grow, traditional systems engineering methodologies often struggle to reconcile development agility with lifecycle discipline. Functional-outcomes-driven tailoring (FODT) is proposed as a structured, evidence-based framework that aligns methodology selection with the functional objectives and outcome imperatives of each lifecycle phase. Building on two prior articles that established the rationale and validation for FODT, this paper presents a practical roadmap for large-scale implementation across single and multi-organizational product development environments. The proposed phased strategy addresses the cultural, procedural, infrastructural, and leadership dimensions essential to successful adoption. While designed to complement earlier foundational work, this paper stands alone in its detailed guidance on how to institutionalize FODT at scale—transforming tailoring from an exception to a best-practice pillar.

1. INTRODUCTION: FROM CONCEPTUAL FRAMEWORK TO SCALABLE EXECUTION – BACKGROUND AND CONTEXT

In previous contributions to this series, the functional-outcome-driven tailoring (FODT) framework was introduced as a response to the systemic disconnect between rigid engineering methodologies and the dynamic needs of modern engineered programs. The first article introduced FODT as a framework that integrates function-driven and outcome-focused engineering principles. The second paper examined its alignment with established systems engineering standards and guidance, including ISO/IEC/IEEE 15288, NATO lifecycle management guidance, NASA systems engineering practices, INCOSE guidance, and Department of Defense (DoD) engineering policy. It further explored how tailoring principles are applied across multiple engineering domains, including defense, aerospace, civil infrastructure, healthcare, and energy systems [1][2][3][6][11].

This third and final instalment focuses on execution. It recognizes that while the principles underlying FODT are reflected in established systems engineering tailoring practices and standards-based guidance, their value depends on effective organizational adoption. Designed as a standalone guide, this article provides a high-level yet practical roadmap for implementing FODT across complex corporate ecosystems—particularly those involving multiple business units, subcontractors, and program stakeholders. It addresses not only process changes, but also the mindset, governance, and infrastructural shifts required to enable meaningful lifecycle tailoring.

2. STRATEGIC IMPLEMENTATION PHASES

Successfully transitioning from FODT's conceptual logic to repeatable practice requires more than agreement—it demands structured execution. The phases that follow are designed to guide organizations through a stepwise, scalable implementation process. Each phase addresses a specific strategic lever—from

leadership commitment to infrastructure readiness—and collectively they form a roadmap to institutionalizing FODT across distributed teams and evolving program environments.

FODT deployment is divided into six implementation phases, each with defined objectives and organizational enablers:

Phase 1: Strategic Alignment and Leadership Buy-In

Objective: Establish enterprise-wide commitment and executive sponsorship.

No transformation initiative can succeed without clear top-down support and alignment with organizational objectives. FODT challenges deeply rooted engineering practices and culture, and thus requires not only conceptual endorsement but also visible, authoritative sponsorship to overcome inertia. The objective of this phase is to elevate FODT beyond a technical proposal and position it as a strategic enabler of corporate performance, operational agility, and lifecycle assurance.

The strategic alignment begins by translating the core FODT logic—tailoring based on function and outcome—into language that resonates with executive concerns. These include program cost containment, speed-to-field, assurance of mission success, and stakeholder trust. Framing FODT as a framework aligned with globally recognized systems engineering guidance, including ISO/IEC/IEEE 15288:2023 [3], helps position it as a structured approach to lifecycle tailoring rather than a departure from accepted engineering practice. Instead, FODT is positioned as a modernization step that operationalizes long-accepted principles in a structured and auditable manner.

Key steps include:

- **Executive briefings to build awareness and ownership**: These briefings should present empirical evidence (from pilot programs, industry case studies, and internal assessments) showing the risks of current rigid methodologies and the potential of FODT to enable traceable, risk-based flexibility [4]. Scenarios can be used to contrast outcomes under traditional vs. tailored approaches.
- **Business case articulation with traceable benefits**: Develop a defensible return on investment (ROI) model tailored to your organization. This should quantify expected improvements in risk management, cost predictability, late-stage rework avoidance, review cycle efficiency, and stakeholder confidence. Where possible, link projected savings or value gains to past project challenges [5].
- **Designation of a senior-level sponsor**: Identify a program director, chief engineer, or vice president (VP) of engineering operations to champion FODT implementation. This sponsor should have both organizational clout and technical credibility. Their responsibilities should include oversight of Phase 2 governance structures, conflict resolution, and external messaging to customers and regulators.
- **Organizational positioning**: Communicate that FODT adoption is not about abandoning proven methods such as V-model, agile, or model-based systems engineering (MBSE). Instead, it is about applying them with increased precision, logic, and traceability—mapped to the purpose and maturity of each lifecycle phase [6].

Together, these actions lay the foundation for trust, clarity, and institutional readiness. They also help pre-empt resistance by aligning tailoring with mission assurance and policy compliance rather than flexibility for its own sake.

Phase 2: Governance and Structural Enablement

Objective: Institutionalize governance mechanisms that support structured flexibility.

For FODT to become a viable, repeatable practice within a corporate systems engineering environment, it must be supported by well-defined governance structures that enable flexibility without compromising control. This phase focuses on building the procedural and institutional mechanisms that formalize tailoring decisions, enforce accountability, and provide assurance to internal and external stakeholders. The shift in governance is not just about allowing choice—it is about enabling informed, auditable, risk-aware decisions that reflect engineering purpose rather than methodological habit.

FODT redefines what governance should accomplish. Rather than enforcing compliance with a single methodology, it encourages oversight of method-to-function alignment. Lifecycle tailoring must be seen not as an exception to be justified, but as a norm to be guided. This requires mechanisms that balance autonomy and traceability—a challenge that is increasingly recognized in ISO 15288's tailoring provisions, the NATO Architecture Framework, and the DoD's shift toward outcome-based engineering oversight [7][8][9].

Similar tailoring-based governance approaches have been discussed within the defense systems engineering community, where tailoring is viewed as a disciplined mechanism for aligning lifecycle activities, reviews, artefacts, and engineering effort with program objectives and constraints [2].

This perspective is consistent with industry guidance that advocates documented tailoring approaches driven by program objectives, mission needs, and lifecycle context rather than uniform application of predefined processes [12].

Implementation actions include:

- **Establish a tailoring governance board (TGB)**: This cross-functional body, ideally chaired by the chief systems engineer or quality lead, should review lifecycle tailoring decisions at milestone gates, validate rationale in the systems engineering management plan (SEMP), and ensure functional and outcome alignment. The board also serves as the escalation point for method disputes and risk-based tailoring deviations.
- **Develop templates, checklists, and tailoring logs**: Standardization tools help teams document tailoring decisions consistently. Templates should include: lifecycle-phase objectives, selected methods, justification based on functional needs, associated risks, and intended outcomes. These tools not only support audits but also facilitate inter-team knowledge transfer.
- **Update policy documentation**: Lifecycle governance documents (for example, SEMP templates, technical planning guides, programmatic review protocols) should explicitly recognize tailoring as a valid strategy. References to fixed method enforcement should be replaced with structured tailoring language aligned with outcome-driven practices [10].

By shifting governance from static control to dynamic enablement, this phase ensures that flexibility does not result in fragmentation. Instead, it becomes a discipline in itself—guided, reviewed, and continuously improved.

Phase 3: Capability Building and Workforce Enablement

Objective: Cultivate cross-phase engineering judgment and systems literacy.

The successful implementation of FODT relies not only on structural governance but on the capability of the workforce to make informed, risk-aware tailoring decisions. Traditional training often emphasizes procedural compliance—knowing how to execute predefined steps in a given methodology. In contrast, FODT requires a shift toward lifecycle fluency: understanding why certain methods serve particular functional objectives at various phases, and when they are appropriate to deploy or combine.

This phase focuses on building organiza-

tional maturity through structured learning and continuous skills development. It requires teams to internalize core FODT principles—function-driven alignment, outcome relevance, and adaptive rigor [3][4]—and apply them through judgment rather than rote.

Enablers include:

- **Tailoring-focused training**: Provide modular training programs targeted at different roles (systems engineers, program managers (PMs), quality assurance (QA), integrated product team (IPT) leads). These sessions should incorporate real-world tailoring case studies, comparisons between traditional and FODT logic, and walk-throughs of tailoring templates and SEMP updates.
- **Micro-certifications and role-based credentials**: Establish internal certifications that demonstrate competency in tailoring logic, lifecycle phase analysis, and method selection. These can be issued upon completion of targeted courses or simulation scenarios.
- **Simulation workshops and scenario-based exercises [10]**: Create team-based workshops where participants analyze hypothetical or historical programs and propose tailoring strategies based on FODT. Facilitate group review and discussion to build critical thinking and peer learning.
- **Learning communities and mentoring**: Establish communities of practice and mentorship programs where experienced engineers coach less experienced team members on tailoring application [5]. This supports cultural normalization and breaks down hierarchical resistance to new practices.

By equipping teams with the ability to reason through complexity rather than simply follow static procedures, this phase drives a fundamental cultural shift—from one of method identity to one of functional purpose. It supports resilience, accelerates adaptation, and empowers engineering judgment across the lifecycle.

The Role of Integrated Engineering in Enabling FODT

Functional-outcome-driven tailoring (FODT) is predicated on the ability to align engineering methods with the function and outcome of each lifecycle phase. For this alignment to be practical and sustainable, engineering activities must be supported by a digitally connected, cross-disciplinary infrastructure—what is commonly known as integrated engineering.

Integrated engineering refers to the coordinated use of processes, tools, data, and engineering disciplines to improve lifecycle traceability, collaboration, and system coherence [3][7]. In the FODT context, it acts as a crucial enabler by allowing tailoring decisions to be made, implemented, and assessed within a shared and synchronized environment.

Key dimensions of integrated engineering include:

- **Integrated design**: Systems, hardware, software, human factors, and supportability engineering are managed within a unified framework, ensuring that design decisions reflect all domain constraints and opportunities. This reduces misalignment between method selection and actual phase objectives.
- **Toolchain interoperability**: Seamless interaction between MBSE tools, product line management (PLM) systems, simulation platforms, and verification databases enables traceable tailoring decisions and phase transitions. For example, tailoring logs can be linked directly to architecture models or validation plans.
- **Cross-disciplinary collaboration**: Integrated engineering supports concurrent, real-time inputs from diverse teams, making it easier to align tailored methods across stakeholders and avoid stovepipe planning.
- **Digital thread continuity**: Tailoring decisions are not isolated artifacts—they are embedded in the digital lifecycle. Integrated engineering ensures traceability from early design intent through verification, production, and sustainment.

Without integrated engineering, tailoring risks becoming inconsistent, undocumented, or non-repeatable. With it, FODT evolves from a governance model into an operational capability—anchored in shared data, common platforms, and aligned processes.

Phase 4: Toolchain and Infrastructure Integration

Objective: Ensure tailoring decisions and traceability artifacts are embedded within engineering toolchains.

For FODT to deliver its full benefit, its logic and decision pathways must be digitally integrated into the tools that support lifecycle engineering. Offline tailoring discussions or ad hoc tailoring records undermine transparency and governance. This phase focuses on embedding FODT into core systems engineering platforms, ensuring that tailoring decisions are visible, reviewable, and aligned with the broader digital engineering strategy [3][7].

Key investments include:

- **Integration of tailoring traceability into PLM and MBSE platforms**: Tailoring decisions should be stored and accessible through lifecycle management and model-based engineering environments, maintaining traceability between engineering decisions, requirements, architecture, and verification artefacts [3][7].
- **Model-based dashboards for tailoring oversight**: Develop visual dashboards that represent method-function alignment across lifecycle phases. These can use heatmap overlays to show where tailoring aligns well or deviates from typical patterns. Dashboards should provide roll-up views for program managers and drill-down detail for engineering leads.
- **Automation of evaluation matrices**: Encode FODT tailoring matrices (previously defined through manual templates) [10] into rule-based decision engines. These tools can assist teams during planning by recommending method options based on system maturity, lifecycle phase, and identified risk domains.
- **Digital thread integration**: Ensure that tailoring decisions are connected to verification artifacts, risk logs, and milestone outcomes across the digital thread. This helps maintain end-to-end traceability from tailoring rationale to operational performance [7][3].

By embedding tailoring into digital infrastructure, organizations make it auditable, repeatable, and resilient. This ensures that tailoring becomes an institutional capability—available at scale, supported by evidence, and governed with consistency.

Phase 5: Pilot Programs and Change Calibration

Objective: Demonstrate value and learn from early implementation.

FODT cannot be imposed in theory alone; its effectiveness must be demonstrated under operational conditions. Pilot programs serve as testbeds where FODT logic, governance mechanisms, and workforce competencies can be exercised, validated, and refined. By selecting programs with varied complexity, scale, and stakeholder sensitivity, organizations can gather representative insights into how tailoring performs in practice. These pilots allow for controlled experimentation while managing exposure to operational risk.

Actions include:

- **Select representative development programs**: Choose pilot candidates based on criteria such as lifecycle phase

diversity, integration complexity, subcontractor involvement, and exposure to delivery risk. Programs nearing key reviews, for example, system requirements review (SRR), preliminary design review (PDR), are especially valuable, as they allow tailoring logic to be tested in milestone preparation [4].

- **Embed FODT tailoring logic across planning artifacts**: Integrate tailoring rationale into systems engineering management plans (SEMPs), risk matrices, and verification planning. Use tailoring logs to document method-function decisions, review tailoring decisions at technical interchange meetings (TIMs), and assess alignment to functional phase goals.
- **Monitor and document pilot metrics**: Establish baselines and collect qualitative and quantitative data. Key metrics may include:
  - Number and type of methods tailored per phase
  - Change requests attributed to inadequate method-function alignment
  - Review gate rework statistics
  - Stakeholder satisfaction with flexibility and traceability
  - Perceived maturity and audit readiness of tailored plans.
- **Conduct structured retrospectives**: After major milestones, facilitate structured feedback sessions with pilot participants to assess what tailoring worked, what failed, and why. Compare tailoring outcomes against both traditional baselines and expected FODT benefits.
- **Use findings to calibrate tools, training, and governance**: Feedback from pilots should drive revisions to tailoring templates, training content, toolchain dashboards, and policy documentation [5][10].

Pilots not only demonstrate value—they foster cultural adoption. Teams that experience the practical benefits of FODT become internal advocates, accelerating bottom-up change. Equally, documented lessons from real implementations help refine and de-risk broader institutional rollout strategies. This ensures that tailoring becomes an institutional capability—available at scale, supported by evidence, and governed with consistency.

Phase 6: Institutionalization and Continuous Improvement

Objective: Embed FODT as a living framework for lifecycle governance.

True FODT maturity does not come from isolated success—it results from systemic adoption and continual refinement. In this final phase, the aim is to establish FODT not as an innovation project or special permission, but as a standard, embedded feature of systems engineering practice. To achieve this, the organization must adopt mechanisms for monitoring FODT application, improving supporting infrastructure, and anchoring it within long-term strategy [3][7][8].

The importance of tailoring lifecycle implementation to project-specific objectives and constraints has also been highlighted in lifecycle planning research, which emphasizes that lifecycle definition should be driven by project needs rather than adherence to predefined process models [2].

Ongoing steps include:

- **Include FODT indicators in program key performance indicators (KPIs)**: Track indicators such as tailoring decision transparency, functional alignment scores, and time-to-review-cycle completion to assess FODT implementation across programs. Tie these indicators to team-level and program-level performance reviews.
- **Audit tailoring logs and SEMP alignment**: Conduct periodic governance audits to ensure consistency between documented tailoring decisions and approved tailoring plans. This reinforces both policy adherence and best-practice evolution.
- **Capture and publish case studies** and lessons learned from internal programs to support continuous improvement and organizational learning [5][10].
- **Engage with external standards bodies**: Ensure FODT practices remain compatible with evolving international and national guidance. Participation in ISO, NATO, INCOSE, and DoD engineering forums helps position the organization as a thought leader while maintaining compliance and peer benchmarking [5][6][7].

3. ADDRESSING RESISTANCE FROM TRADITIONAL METHODOLOGY ADVOCATES

Resistance to change is a predictable and often rational response—particularly when existing systems have long been associated with program success, risk control, and compliance assurance. Traditionalists may perceive FODT as a threat to rigor or as a step away from proven discipline. Overcoming this resistance requires a structured and empathetic change management strategy built on inclusion, demonstration, and governance assurance. As FODT becomes operationalized, organizations must proactively address the cultural tensions between legacy approaches and the tailored mindset.

The following practices outline a multi-faceted strategy to build confidence, maintain rigor, and promote ownership among those accustomed to traditional methods:

- **Position FODT as evolution, not rejection**: Communicate clearly that FODT does not discard traditional methodologies such as the V-model, waterfall, or MBSE. Instead, it places them in context—deploying each where it is most functionally and operationally effective. This framing shifts the narrative from "change" to "optimization," reducing perceived risk to methodological identity.
- **Use internal champions from traditional backgrounds**: Select respected practitioners with deep experience in traditional approaches as pilot leads, FODT ambassadors, or tailoring governance board members. Their endorsement carries credibility and reassures skeptics that tailoring can coexist with legacy principles.
- **Provide side-by-side comparisons**: Offer scenario-based briefings and documented cases that compare outcomes under traditional vs. tailored execution [8][4]. Emphasize improvements in review readiness, reduced rework, and early risk surfacing. Data from pilot programs should be central to this argument.
- **Codify rigor through governance**: Show that FODT enhances—not erodes—discipline through formalized templates, review structures, and auditable tailoring logs. Reinforce that tailoring is governed, reviewed, and traceable—mitigating concerns about inconsistency or quality loss.
- **Maintain familiar artifacts and language**: Where possible, retain existing terminology, for example, critical design review (CDR), SEMP, SRR, and integrate FODT within known frameworks. This reduces the cognitive load and helps reposition FODT as an enhancement rather than a rewrite.
- **Acknowledge and validate concerns**: Create forums for open dialogue where traditionalists can voice concerns and receive reasoned responses. Use those opportunities to clarify misconceptions and highlight FODT's compatibility with established engineering standards like ISO 15288 [3], NATO STANAG 4728 [6], and the DoD Digital Engineering Strategy [7].
- **Emphasize compliance and audit readiness**: Demonstrate how tailoring logs, evaluation matrices, and digital thread integration improve oversight and review performance. Highlight that FODT meets or exceeds conformance

requirements while enabling adaptation to project-specific realities [3][4][7].

4. CRITICAL ENABLERS OF SUCCESS

FODT is not just a methodological toolkit; it is a transformation of organizational engineering culture. As organizations progress through implementation, the distinction between tailoring as an ad hoc adaptation and tailoring as a strategic norm becomes critical. Achieving maturity with FODT requires consistent investment in culture, competence, and infrastructure. This section outlines the foundational enablers necessary to embed FODT as a sustainable, value-generating practice across programs and disciplines. For it to succeed, organizations must go beyond procedural adjustments and instill systems-level thinking and lifecycle fluency across all functions. This requires a harmonized effort combining leadership vision, integrated tooling, structured governance, and continuous skill development. Key enablers include:

- **Executive mandate and vision communication**: FODT must remain a leadership priority to retain visibility and integration with enterprise goals.
- **Method governance that encourages informed flexibility**: Governance structures should emphasize judgment, traceability, and outcome relevance over static method enforcement.
- **Tailored training and embedded support infrastructure**: Institutional learning systems must evolve in parallel with process updates.
- **Cross-program feedback loops and organizational learning**: Pilot insights, audit data, and engineering feedback must be cycled into continuous process improvement [4][10].

Only by embedding FODT within daily workflows—supported by integrated engineering and sustained by data-driven improvement cycles—can tailoring evolve from a reactive workaround into a strategic advantage. In this light, FODT becomes more than a compliance measure—it becomes a means to build responsiveness, accountability, and long-term engineering resilience.

This implementation strategy balances ambition with discipline—enabling organizations to evolve from rigid lifecycle frameworks to dynamic, mission-aligned engineering governance. By anchoring tailoring decisions in functional purpose and validated outcomes, FODT establishes a foundation for high-velocity, high-integrity system development across even the most complex corporate structures. Teams that experience the practical benefits of FODT become internal advocates, accelerating bottom-up change. Equally, documented lessons from real implementations help refine and de-risk broader institutional rollout strategies. This ensures that tailoring becomes an institutional capability—available at scale, supported by evidence, and governed with consistency.

5. CONCLUSION: FROM TACTICAL FLEXIBILITY TO STRATEGIC ADVANTAGE

The shift toward functional-outcome-driven tailoring is not simply a technical adjustment—it represents a broader redefinition of how engineering excellence is achieved in complex, high-stakes environments. FODT will bring structure to flexibility, clarity to tailoring, and purpose to method selection. It empowers engineering teams to operate with agility without compromising control and compliance.

By institutionalizing FODT through deliberate governance, leadership sponsorship, workforce enablement, and integrated infrastructure, organizations can replace procedural rigidity with responsive, outcome-focused strategies. The result is not only improved engineering performance, but also stronger stakeholder trust, reduced lifecycle risk, and enhanced system relevance.

FODT is more than an idea whose time has come—it is a framework that transforms tailoring from exception to strategy, **aligning how we engineer with why we engineer**. ■

REFERENCES

- 1. Bethmann, J. F. 2009. "Simplifying the Lifecycle Definition Process." NDIA Systems Engineering Conference, November, Concurrent Technologies Corporation.
- 2. Miller, C. 2012. "How Has Effective Systems Engineering Benefited Our Defense Programs?" NDIA Systems Engineering Conference, Harris Corporation.
- 3. ISO/IEC/IEEE 15288:2023. Systems and Software Engineering — System Life Cycle Processes. Geneva, CH: ISO/IEC/IEEE.
- 4. Defense Acquisition University (DAU). 2022. *Engineering of Defense Systems Guidebook*. Washington, US-DC: Department of Defense.
- 5. INCOSE. 2023. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, Version 5.0. San Diego, US-CA: International Council on Systems Engineering and Wiley.
- 6. NATO Standardization Office. STANAG 4728 – System Life Cycle Management. Brussels: NATO.
- 7. Department of Defense. 2018. DoD Digital Engineering Strategy. Washington, US-DC: Office of the Under Secretary of Defense for Research and Engineering.
- 8. Elm, Joseph P., and Dennis Goldenson. 2012. The Business Case for Systems Engineering Study: Results of the Systems Engineering Effectiveness Survey. Software Engineering Institute, CMU/SEI-2012-SR-009.
- 9. European Space Agency. 2020. *ECSS-E-ST-10C – Space Engineering: System Engineering General Requirements*. Noordwijk, NL: ESA Requirements & Standards Division.
- 10. Oehmen, Josef, et al. 2012. "The Guide to Lean Enablers for Managing Engineering Programs." Massachusetts Institute of Technology and Project Management Institute.
- 11. NASA. 2016. *NASA Systems Engineering Handbook*, Rev 2. Washington, US-DC: NASA/SP-2016-6105.
- 12. Hantos, P. 2008. IEEE Life Cycle Standards and the CMMI – Implementation Considerations. Annual CMMI Technology Conference (7th).

MBSE with SysML V2 & CATIA Magic

Expert Webinar Series.
On Demand and Live. With Live Q&A Sessions

November 6, 2025 -
December 31, 2026

DASSAULT SYSTEMES

CHECK OUT
THE INCOSE MEDIA KIT!

Contact **advertise@incose.net** to discuss options!