Software Defines Tactics: Structuring Military Software Acquisitions for Adaptability and Advantage in a Competitive Era

BY JASON WEISS AND DAN PATT
ABOUT HUDSON INSTITUTE

Hudson Institute is a research organization promoting American leadership for a secure, free, and prosperous future.

Founded in 1961 by strategist Herman Kahn, Hudson Institute challenges conventional thinking and helps manage strategic transitions to the future through interdisciplinary studies in defense, international relations, economics, energy, technology, culture, and law.

Hudson seeks to guide policymakers and global leaders in government and business through a robust program of publications, conferences, policy briefings, and recommendations.

Visit www.hudson.org for more information.

Hudson Institute
1201 Pennsylvania Avenue, N.W.
Fourth Floor
Washington, D.C. 20004

+1.202.974.2400
info@hudson.org
www.hudson.org

Cover: The unique design of the new F-35A Lightning II helmet at Hill Air Force Base, Utah July 18, 2017. (US Air Force photo by Michael McCool)
Software Defines Tactics: Structuring Military Software Acquisitions for Adaptability and Advantage in a Competitive Era

BY JASON WEISS AND DAN PATT
ABOUT THE AUTHORS

Jason Weiss

Jason Weiss brings an exceptional background in software engineering architecture and cryptology dating back to his service in the US Navy as a cryptologist during the first Gulf War. He has lectured internationally on computer science topics, authored multiple books, holds several patents, and is a successful serial entrepreneur. He holds an MA in intelligence and a BS in computer science. Jason is presently the chief software officer at Conquest Cyber, and prior to that he was the first chief software officer for the Department of Defense, responsible for driving software modernization policy and guidance, and was a co-author of the DoD’s Software Modernization Strategy. Previous positions held include vice president, IT transformation, at BAE Systems, Inc., and senior vice president, cloud platform and applications, at SMART TRAC N.V.

Dan Patt

Dan Patt is a senior fellow with Hudson Institute’s Center for Defense Concepts and Technology. His experience is at the intersection of technology, business, and national security strategy. His work at Hudson focuses on the role of information and innovation in national security.

Dr. Patt supports strategy at the artificial intelligence company STR and supports Thomas H. Lee Partners’ automation and technology investment practice. He has more than 15 years of experience operationalizing emerging technology, including artificial intelligence, networked information systems, robotics, supply chain automation, and enterprise information technology. He holds advisory board roles at the University of Michigan College of Engineering, Worcester Polytechnic Institute, and Andrew W. Marshall Foundation.
# TABLE OF CONTENTS

Executive Summary 7

Introduction 9

It’s All about Adaptability 15

Like Digital Logistics 24

The Modern Software Factory 34

Applied Software Acquisition 43

Exercising ACTS: A Vignette 57

Appendix A: Documentation as Code 64

Appendix B: List of Acronyms 65

Endnotes 69
Software Defines Tactics

ACTS: Acquisition Competency Targets for Software
Six principles that enable superior speed and adaptability for delivering modern military capability

ACTS 1
Software Factories
Evaluate existing software factories before you build your own
Corollary
Rationalize staffing ratios between Government civilians and contractors, know that staffing needs will ebb and flow across program life.

ACTS 2
Continuous Authority to Operate (cATO)
Partner with Authorizing Officials (AOs) to realize a Continuous Authority to Operate
Corollary
Establish a plan for software bill-of-materials (SBOM) consumption

ACTS 3
Interfaces
Own the interfaces and APIs, more than the implementation or code
Corollary
Ensure the APIs provide access to the data the program cares about

ACTS 4
Think in Service Levels
If you can define a service level agreement (SLA) around a portion of your charter, contract that portion out.
Corollary
The lollipop (software interface) is more important than the enterprise architecture

ACTS 5
Zero Trust
Define Zero Trust outcomes within software applications, to infuse with resilience
Corollary
Realize digital transformation without explicitly demanding it

ACTS 6
Behave Like a Software Recruiter
Creatively recruiting digital talent will power your program’s success
Corollary
Shrink the distance between the engineer and the warfighter

Figure 1. The Six ACTS
You would not be reading this if you did not realize that it is important for the Department of Defense (DoD) to get software right. There are two sides to the coin of ubiquitous software for military systems. On one side lies untold headaches—new cyber vulnerabilities in our weapons and supporting systems, long development delays and cost overruns, endless upgrade requirements for software libraries and underlying infrastructure, challenges in modernizing legacy systems, and unexpected and undesirable bugs that emerge even after successful operational testing and evaluation. On the other side lies a vast potential for future capability, with surprising new military capabilities deployed to aircraft during and between sorties, seamless collaboration between military systems from different services and domains, and rich data exchange between allies and partners in pursuit of military goals. This report offers advice to help maximize the benefits and minimize the liabilities of the software-based aspects of acquisition, largely through structuring acquisition to enable rapid changes across diverse software forms.

This report features a narrower focus and more technical depth than typical policy analysis. We believe this detail is necessary to achieve our objectives and reach our target audience. We intend this to be a useful handbook for the DoD acquisition community and, in particular, the program executive officers (PEOs) and program managers as they navigate a complex landscape under great pressure to deliver capability in an environment of strategic competition. All of the 83 major defense acquisition programs and the many smaller acquisition category II and III efforts that make up the other 65 percent of defense investment scattered across the 3,112 program, project, and activity (PPA) line items found in the president’s budget request now include some software activity by our accounting. We would be thrilled if a larger community—contracting officers, industry executives, academics, engineers and programmers, policy analysts, legislators, staff, and operational military service members—also gleaned insight from this document. But we know that some terms may come across as jargon and that not everyone is familiar with the names of common software development tools or methods. We encourage them to read this nonetheless and are confident that the core principles and insights we present are still accessible to a broader audience.

While other recent analyses have focused on the imperative for software and offered high-level visions for a better future, we believe most of the acquisition community already recognizes the potential of a digital future and is engaged in a more tactical set of battles and decisions: how to structure their organizations, how to manage expectations, how to structure their deliverables, and how to write solicitations and let contracts meet requirements and strive for useful outcomes. We attempt to present background and principles that can assist them in navigating this complex landscape.

The real motivation for getting military software right is not to make the DoD more like commercial industry through digital modernization. Instead, it is to create a set of competitive advantages for the United States that stem from the embrace of distributed decision-making and mission command. The strategic context of this work stems from the observation that advantage in future military contests is less likely to come from the mere presence of robotics or artificial intelligence in military systems and is more likely to come from using these (and other) components effectively as part of a force design, and specifically from maintaining sufficient adaptability in the creation and implementation of tactics. Software, software systems, and information processing systems (including artificial intelligence and machine learning, or AI/ML) that leverage software-generated data are the critical ingredients in future force design, force employment, tactics generation, and execution monitoring. Software will bind together human decision-makers and automation support in future conflict.

This report aims to accomplish two goals:

- Elucidate a set of principles that can help the acquisition community navigate the software world—a turbulent sea of
buzzwords, technological change, bureaucratic processes, and external pressures.

- Recruit this community to apply energy to a handful of areas that will enable better decisions and faster actions largely built around an alternative set of processes for evolutionary development.

The report is structured in chapters, each built around a core insight. Chapter 1 notes that the speed and ubiquity of digital information systems is driving military operations ever closer to capability development, and that this trend brings promise for capability and peril for bureaucratic practices.

Chapter 2 offers the military framing for software development, pointing out that it can enable greater adaptability in force employment and that the DoD cannot derive future concepts of more distributed forces and greater disaggregation of capability from requirements and specifications alone. Instead, software should enable employment tactics to evolve, especially as the elements of future combat are outside the control of any one program manager or office.

Chapter 3 introduces a framing analogy between software delivery and logistics. As logistics practitioners recognize, there is no one-size-fits-all solution to the problem of moving physical goods, but a set of principles that helps navigate many gradients of logistics systems. Similarly, the world of software is full of diversity—different use cases, computing environments, and security levels—and this heterogeneity is an intrinsic feature that the DoD needs to embrace.

Chapter 4 introduces the essential tool of the modern program office—the software factory—the tooling and processes via which operational software is created and delivered. All existing operational software was made and delivered somehow and, whether we like it or not, we will have to update it in the future. Thus, the production process matters. In many ways, the most important thing a PEO can do is identify, establish, or source an effective software factory suited to the particular needs of their unique deliverables.

Chapter 5 seeks to break down the ideas introduced earlier into actionable atomic principles that the DoD can use to navigate the tough decisions that the acquisition community faces; we refer to these ideas as ACTS. To aid the reader in understanding these, we also provide a fictitious vignette.

We recognize the inherent tension between making a report compact and accessible and covering the myriad facets of a broad and fast-changing scene. To balance this, we believe that we have focused this report on the most impactful aspects of software acquisition.
Embracing and orchestrating an ever-changing, diverse mixture of software is the key to competitive technical advantage in an era of ubiquitous computing and data.

The software-defined world is no longer speculative; it is here. Software controls critical systems in our homes, from internet-connected smoke detectors to refrigerators and stoves. It controls core systems in our cars, helping us parallel park and even braking for us. It controls the critical infrastructure of cities, from traffic lights to drinking water. And in the case of the DoD, software controls everything from command-and-control systems to tactical sensors deployed at the edge, from business systems that issue paychecks to the voltage controller on a battery for a man-portable radio.

From the onset, readers should understand that the title Software Defines Tactics is a nod to what is conceptually a closed-loop system between military operations themselves (whether in peacetime or wartime) and the development and fielding of military capability. As more military systems are software-enabled and artificial intelligence becomes more proliferated, the coupling in this loop becomes tighter and the linkages become explicitly defined by digital data (see figure 2). The design, implementation, and deployment of software produce vast amounts of digital data whether the target environment is the cloud, a sensor at the tactical edge, or an embedded system in a warfighting platform.

That data is labeled, curated, and made accessible to data scientists looking to design and build next-generation AI or ML.

models. Scientists can extract the value of any of these AI/ML models only once they integrate a model back into a software system, completing the closed loop: software begets data, data begets AI/ML models, and AI/ML models need to be integrated back into software.

The Department of Defense, in its acquisition and development, needs to focus first and foremost on software, and only then consider the supporting hardware. The digital triad of software, data, and AI/ML models are so intertwined that to advance any one without advancing the other two is the equivalent of digital suicide. There is marginal value in expanding the scope and quality of digital data generated by a software solution if that data cannot be stored and retrieved for access by data scientists. By extension, there is little value in generating advanced AI/ML models if it is impossible to integrate those models into software that the US military can deploy in an operational environment at the speed of relevance. In fact, we hypothesize that the only leg of the digital triad that the DoD could advance in isolation is software, and this has been the de facto situation for several decades.

Focusing on the last two decades, the birth and corresponding hyper-growth of cloud service providers dramatically altered the
way companies architect, build, test, and deploy software. Simultaneously, the growth of compute, network, and storage in these ecosystems reached a level of scale and affordability that spawned the current growth of data science to include narrowly focused AI/ML models. Our society now sees literally every commercial sector—from aerospace and defense to education, energy, finance, healthcare, and utilities—as software defined. The advent of cloud computing, cloud-native architectures, and what we now call the software factory has affected business strategies as much as it has affected warfighting strategies and tactics.

The DoD’s epiphany that whichever force exemplifies a mastery of cloud computing and software will have the upper hand in future conflicts materialized on September 17, 2017. On this date, the secretary of defense signed out a memo entitled “Accelerating Enterprise Cloud Adoption.” The memo recognized that commercial industry was outpacing the DoD, cloud infrastructure was enabling machine learning, and cybersecurity was becoming far more critical as a result. The memo concluded, “Speed and security are of the essence.”

Following this epiphany, the FY18 National Defense Authorization Act (NDAA), section 872, directed the secretary of defense to task the Defense Innovation Board “to undertake a study on streamlining software development and acquisition regulations.” On May 3, 2019, the Defense Innovation Board published Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage and a corresponding implementation plan. The report rationalized three fundamental themes:

1. Speed and cycle time are the most important metrics for managing software.
2. Software is made by people and for people, so digital talent matters.
3. Software is different from hardware (and not all software is the same).

It went on to outline four main lines of effort:

1. Congress and the DoD should refactor statutes, regulations, and processes for software.
2. The Office of the Secretary of Defense (OSD) and the services should create and maintain cross-program/cross-service digital infrastructure.
3. The services and OSD will need to create new paths for digital talent (especially internal talent).
4. The DoD and industry need to change the way they procure and develop software.

The DoD has made tremendous progress in some areas, like introducing DoD Instruction (DoDI) 5000.87, Software Acquisition Pathway. It has also been able to benefit from the Office of Personnel Management’s (OPM’s) recognition that the Computer Science Series of occupational requirements for government employees inadequately represented the growth of data as a specialty. OPM’s introduction of the Data Science Series allows the DoD to recruit from a pool of resources that it previously excluded because of the 1550 series requirement of studying differential and integral calculus. Calculus has not played as significant a role in the growth of AI/ML as statistics.

Yet in other critical areas, the DoD continues to struggle to find its footing. The department has yet to demonstrate it can consistently adopt and deploy cloud computing at a pace that matches commercial industry. On July 6, 2021, nearly four years after the Shanahan memo, the DoD terminated the $10 billion Joint Enterprise Defense Infrastructure (JEDI) cloud project. As of late 2022, the successor Joint Warfighting Cloud Capability (JWCC) program publicly stated that it has no plans to issue any awards until at least December 2022. Award is never a synonym in the DoD for operationally fielded, which means well over five years will have passed since the original memo declared that speed is of the essence.

Meanwhile in Topeka, Kansas, the co-founder of an agritech start-up entered her credit card data on Monday morning and
by the afternoon was spun up, online, and running an entire business through the cloud, including sales, human resources, and finances functions. The company is using state-of-the-art software tooling to build a next-generation harvest optimization solution. This digital tooling exists for every innovator launching a new business venture, and it exists for peer and near-peer competitors that oppose democracy. This growing disparity should do more than merely cause concern among senior DoD leaders; it should trigger bolder actions.

An industrial agriculture company founded in the early 1900s, perhaps with a vision of making affordable tractors, would have had to think about freezing designs, blueprints, production capacity, spare parts inventory, and maintenance manuals before delivering its first product—rough analogs to factors in the milestones in the DoD acquisition process. However, the new agritech start-up can create operational capability in a day. Over the longer term, the start-up's ability to meet customer needs, effectively sell and distribute its product, and stay relevant in a changing environment will determine whether it succeeds—these are unchanging principles. But information technology has greatly alleviated the burden of up-front planning and enabled the new start-up to evolve its way to success. External feedback helps catch and correct inevitable oversights, errors, and bad judgment; software tools lend insight on progress against the so-called -ilities, like maintainability.

The DoD is also struggling to recognize that data strategies that show great promise for enterprise systems do not translate well to the tactical edge, where denied, degraded, intermittent, or limited (DDIL) communications environments are the norm. The bandwidth available in Link 16—which the DoD designated as the primary tactical data link for all military service and defense agency command, control, and intelligence (C2I) systems in October 1994—has not substantially evolved since.8 Yet over this same period, internet traffic volume has grown exponentially.9 The totality of data that computers generate on a naval vessel underway is measured in terabytes, and most of it must be thrown on the proverbial deck. Why? Due to a ship’s size, weight, and power (SWAP) constraints, adding racks of servers is not a viable option, and backhaul over Link 16 or other SATCOM channels is impractical.

Elsewhere, that same start-up in Topeka, Kansas, entered its credit card data and purchased state-of-the-art, low-latency broadband internet that works in remote and rural locations around the globe. Combining that service with a solar panel, this start-up has established more powerful backhaul for farm field environmental sensors than what the world’s most capable military has ready access to. To add insult to injury, the start-up spent less than the cost of two weeks of groceries for American households in the lowest income quintile.10 In the Ukrainian war, this same broadband has even demonstrated its ability to resist a peer military competitor’s electronic warfare attacks.11 These facts lead the warfighter to an incredulous gasp and open ponderance of how an organization that Congress has authorized a top-line budget of $847 billion in the FY23 NDAA cannot figure out how to ubiquitously escape dial-up speeds at the tactical edge.

Lastly, and perhaps most critically, the DoD is struggling to realize gains from the Software Acquisition Pathway in a repeatable manner and at a scale that can outpace either near-peer or peer military competitors. A closer review of the pathway states:

Programs will require government and contractor software teams to use modern iterative software development methodologies (e.g., agile or lean), modern tools and techniques (e.g., development, security, and operations [DevSecOps]), and human-centered design processes to iteratively deliver software to meet the users’ priority needs.12

Overwhelming evidence shows the world is deep into the turning point of the Age of Software.13 Too often, PEOs are found asking themselves what constitutes modern in an institution
like the DoD that has demonstrated a proclivity for continually applying Industrial Age methods in the Age of Software. Some attempt to educate PEOs that water-Agile-fall is modern, when in reality they would do better to frame it as the “best some organizations can do to progress toward faster development.”

We believe that the PEOs and their cadre of program managers are the critical unit of action for modernizing the department. And to deliver a warfighting advantage to their end-user, the DoD needs to equip these PEOs with knowledge and empower them to identify and look beyond the usual incumbents if water-Agile-fall is “the best” the organization can do. In the software-defined conflict, lives literally hang in the balance.

*Software Defines Tactics* picks up where the *Software Is Never Done* report ended, exploring three new critical concepts that the DoD needs to unpack to realize its ambition of dominating the Age of Software. It postulates that *Software Is Never Done* contained enough material regarding digital talent that we have chosen to omit the subject from this piece. It also accepts, begrudgingly, that congressional action on statutes and regulations will occur across a period measured in years, if not decades. Instead of focusing on those aspects of the discussion, this piece explores several areas that *Software Is Never Done* either implied or omitted.

First, we recognize and explore the value of diversity of form—the heterogeneity of software, if you will—and the fact that a one-size-fits-all approach, whether it is a software architecture or a data standard, is incongruent with the software-defined world the warfighter must fight in. Perhaps the most obvious example of this position is found in the enterprise system (e.g., travel reimbursement, payroll), which the DoD will never forward deploy and force to operate in an austere environment subject to denied, degraded, intermittent, or limited (DDIL) communications. Enterprise systems never have to make difficult design trade-offs because of SWAP constraints that exist on an aircraft, ship, or satellite. The worst self-imposed affliction on DoD enterprise systems is known as a period of non-disruption (POND), when a general or flag officer has determined that even applying software patches to avoid the chance of the system going offline is unacceptable. (The cybersecurity implications of a POND are worthy of separate and extended discussion.)

Second, the DoD needs to act in a way that recognizes software, not legacy warfighting platforms, *controls the speed and efficacy of the modern kill chain* and military dilemma. The duality of software components, libraries, and frameworks is that they all apply to social media or gaming as much as to warfighting or intelligence, surveillance, and reconnaissance (ISR) systems. The acquisition engine should consciously acknowledge its own original sin, predilection for prediction, and accept and embrace the idea that warfighting systems have to adapt at the speed of software and not the speed of acquisition. This is not a hypothetical illustration. Public reporting on the low quality of F-35 software has been occurring for years to the extent that the DoD called in multiple universities to see if they might be able to help resolve the problems.

In a world where software defines tactics, it is irrational, irresponsible, and perhaps even unethical to contractually require more than a broad set of outcomes because we simply cannot know actual architectural requirements a priori in a software-defined world.

Finally, the DoD needs to formally recognize the digital triad of software, data, and AI/ML as equal peers. The triad symbolizes the principled belief that *AI/ML model research cannot occur without troves of relevant digital data, and the only way to create digital data is through software*. Completing the feedback loop that validates AI/ML models in an operational environment is possible only if the DoD deploys the model—via software. Yet deployment frequencies measured in months, quarters, and years are undeniably not suitable for meeting the speed of relevance in a software-defined world.
Our collective conviction is that these principles drive our premise that the bedrock of success in the modern PEO is nothing less than identification and selection, empowerment, and operation of the software factory. The more effort PEOs put forth to recognize and embrace the digital triad as a closed-loop system that affects their kill chains, the more resilient and relevant capabilities they will field for the warfighter. Why? The side with the more capable software factory that can push out new fields for data collection faster, push out refined AI/ML models faster, and operate in austere environments with greater assurances of fielding cyber-resilient systems will enjoy both strategic and tactical advantages that too many military leaders underappreciate today.

Each member of the digital triad—software, data, and AI—is proverbially a mile deep and a mile wide. Software is the primary focus of this work, and therefore we openly invite those with rich industry expertise in data and AI/ML to build on our body of work in this report to support PEOs in the other two members of the digital triad.

In the pages that follow, we will explore these three critical concepts intentionally from the perspective of the PEO: heterogeneity of software, how software controls the speed and efficacy of every possible modern kill chain, and, specifically, a coherent value proposition of the software factory.
Why software defines tactics for the modern kill chain and how to extract a strategic advantage

The presence of software in current and future military systems is inevitable and undeniable; its existence should be sine dubio today. The newer F-35 has a shorter range and lower cruise speed and is reportedly less stealthy than the similarly priced F-22 but is still considered the more useful weapon. The reasoning for this relates to the less-physical “intangible” features in the software of the F-35, including its electronic warfare capabilities, superior sensor fusion, ability to communicate more broadly, and ability to detect more threats. Even as early as 1960, the F-4 Phantom fighter used software to implement around 8 percent of the performance specifications that the DoD required it to achieve.\textsuperscript{16} By the time the F-22 rolled around, 80 percent of weapons specifications depended on software, while emerging unmanned systems, including the Navy MQ-25 and future Air Force Collaborative Combat Aircraft (CCA) systems, will require software to perform every specified function. And, of course, modern militaries depend on more than simply tangible weapons like aircraft and ships to fight. They need command and control systems, networking stacks, and edge data processing and presentation. These capabilities exist only as software, and more often today they run on commodity commercial hardware because of economics, a theme found elsewhere in our paper.

So if software is already ubiquitous in the fielded capability of DoD, why is it necessary to change apparently competent acquisition practices? Why do many think that the acquisition

Photo Caption: A woman wears a VR headset on July 29, 2022, at a base in Kyiv Region, Ukraine, where Ukrainian military personnel learn to control drones for combat missions. (Kaniuka Ruslan/ Ukrinform/Future Publishing via Getty Images)
corps does not appreciate the value of software in the modern kill chain? Any conflict in the foreseeable future will likely leverage mostly the equipment that we already own, and military personnel will likely have similar educational and career experiences to those of today’s personnel. If the military is largely made up of the three ingredients of personnel, hardware, and software, and we already have all three, then why should the military pay any special attention to the procurement, management, and delivery of software beyond the status quo?

The best answer to this question could give the United States a fundamental and strategic advantage. Put simply, if the DoD acquires and develops software according to the set of principles articulated here, with high-speed delivery mechanisms capable of quick changes, then we believe software can enable superior adaptability for the US military compared to strategic competitors. Make no mistake—adaptability is not an inherent property of software or of a software-defined system. No shortage of software exists in an entirely fixed and unadaptable context. Luddite leaders did not dictate nuclear command centers’ use of eight-inch floppy disks through 2019, well after such disks were no longer manufactured; an inability to easily adapt the underlying software did so. Moreover, foreign militaries also have software in weapons, and they too are rapidly realizing that software defines tactics.

The fundamental advantage of software is that it possesses both fluid and solid forms, where these terms relate to relatively low and high costs of change. When code is in development and in its source form, it is relatively easy to make a tweak here and there to adjust the code’s behavior. However, once the code becomes a compiled binary running on an obsolete IBM/1 system, making a change is quite expensive. Hardware, in an analogy to carbon, does not possess a practical fluid form. Once it has been deployed, changing its design normally becomes extremely expensive. Similarly, once code has been compiled and deployed and disconnected from the build chain that produced it, it generally becomes rigid and resistant to change. Change, if any, happens only by returning to code in its source form, and as we will explore further, modern source code requires dozens of supporting tools to maintain its malleability.

Achieving a strategic advantage from the hybrid legacy/modern force that the US possesses depends on keeping more software in a more fluid form; driving change to counter discovered vulnerabilities; and adapting to emergent opportunities that include urgent feature needs, new operational concepts, and especially the deployment of new or modified data fields for data science and AI/ML activities. This becomes especially true as future force employment is increasingly likely to rely on the cooperative employment of systems-of-systems that any single contractor or a singular program office cannot practically deliver. Modern software is no longer some monolithic thing. It exists in no singular repository with the advent of distributed version control systems like git. It is no longer built from a singular programming language but lives in a diverse and ever-changing environment where infrastructure-as-code and cloud native architectures accept, if not encourage, disparate tool chains. Today’s software is rarely designed for a singular type of hardware and is often capable of elasticity to expand or contract based on the hardware resources accessible to it. Understanding how to achieve software adaptability in the face of the scope and scale of software heterogeneity is therefore a central theme of this report.

The Force of the Future

Military capstone documents and science fiction alike point in the same direction for the force of the future. A new Joint Warfighting Concept and its supporting concepts, like Joint All Domain Command and Control, emphasize bringing together weapons from different military services that different program offices procure, different contractors build, and different personnel from different military specialties operate in different physical domains. This is not merely some change in organizational chart design or a set of enhanced training exercises. The Joint Warfighting Concept, drawing from DARPA’s Mosaic Warfare concept, emphasizes the ability to aggregate and disaggregate the underlying combat functions from military units and
weapons contributed by all of the services. This vision requires the ability to technically integrate and orchestrate the software inside mission systems, operational support systems, communications systems, weapons, and even enterprise systems. As this document will show, the heart of this vision lies in adaptability for delivering software, in pipelines that allow the specific tactics and operational concepts of military capability to shift on demand with the ease of sliding a finger across the capacitive glass screen of a mobile phone.

In the science fiction version of future warfare, human operators engineer and corral vast collectives of automated systems to achieve a combat advantage. We imagine swarms of systems maneuvering around complex terrain, receiving mission updates from satellite constellations, and seamlessly coordinating to realize desired battlefield objectives. These visions tend to focus on the ingredients—AI, drones, robotic vehicles, goggles, and touchscreens that allow these capabilities to team up with their human counterparts. Robots, in particular, feature in leading roles in this vision. But more exciting and more realistic than robotic weapons systems is the prospect of more dynamic military capabilities: combinations of forces, operational concepts, schemes of maneuver, and tactics that have themselves become dependent on software.

Upon closer examination, a richer version of human-machine teaming that goes beyond a crewed fighter and an uncrewed wingman is already upon us. Automated code tests and automated deployment pipelines may seem like trivial robotic agents compared to the R2D2 and C3PO that we watch on television but are very real and very mature parts of the software delivery enterprise. We are on the verge of discovering that software will define not just weapons systems capabilities, but also tactics—how forces are used—and thus military capability writ large.

**Autonomy in the Force**

Consider a simple example as depicted in figure 3. On the left, an acquisition program procures a new advanced fighter...
aircraft. The same program office also procures an uncrewed combat aircraft that acts as a wingman, providing distributed forward electro-optical and passive radio-frequency sensing in the battlespace, leveraging a smaller physical size to reduce its own optical signature, and baiting adversary systems into illuminating it with active radars to give away their own location.

The program office elected to acquire these systems from two different prime contractors, with subcontractors providing key mission systems. After the contractors delivered each aircraft and its subsystems, the DoD pushed them through a rigorous verification and validation process. Each works well, and excellent program management delivered them on time and under budget. The DoD also tested the combined manned-unmanned team against surrogate adversary air assets and showed it is robustly capable of conducting its distributed air-to-air mission.

Unfortunately, after sudden and unexpected foreign aggression, conflict breaks out. Adversary fighter aircraft reveal wartime modes with longer-range missiles that out-stick optical sensors and a new mode in their own radar systems that enables them to detect, track, and prosecute the uncrewed wingman without giving away their own positions. Suddenly the advanced fighter no longer has an engagement range advantage, and the uncrewed wingmen are more of a liability than an asset.

However, all the ingredients for restoring advantage are at hand; the US military simply needs experts and contractors to develop the operational concepts, tactics, and digital integration. Operators and technologists suggest three independent mitigations to the adversary development. First, developers can update an overhead commercial RF-sensing constellation with a new mode to detect adversary fighter emissions; the constellation has the ability to downlink cues to the advanced fighter aircraft. Second, nearby naval surface assets have SM-6 missiles that can prosecute adversary aircraft if they receive accurate targeting information while the ship itself remains in a passive mode. Third, forces can deploy the uncrewed aircraft in a new mode that allows them to act as decoys, baiting adversaries into firing missiles that create flares detected via overhead infrared constellations, then using a new jamming mode to defeat the targeting endgame of the adversary missile.

Implementing any or all of these three mitigations in a relevant timescale is impossible if the DoD follows a conventional acquisition paradigm, even neglecting requirements and authorities considerations. No single program executive office oversees all of the relevant elements of the mitigations—all require cooperation among multiple military services, multiple existing mission systems, and multiple industry vendors. Working conventional operational testing and evaluation across the requisite combinations would be extraordinarily time-consuming, and yet it must result in a high degree of certainty in the solution. Indeed, within the conventional acquisition paradigm, despite technical feasibility, these mitigations appear to be more of a fantasy than a solution.

This example illustrates that modern combat will be largely concerned with the operational concepts and tactics of employing multiple weapons systems in concert. Furthermore, it illustrates the centrality of adaptability, automation, and autonomy in driving combat outcomes. While warfighters have always had to shift tactics in response to changing conditions in conflict, this theme now extends into the digital realm. Thanks to a proliferation of software-enabled mission systems across domains, defining the tactics of a human-machine collective using software becomes possible.

Often, people mistakenly conflate autonomy with automation. They often improperly refer to an aircraft with automated guidance, navigation, and control as “autonomous” when it would be more appropriate to say that the aircraft’s automated control capability permits it autonomy over its flight path. Fundamentally, autonomy is a relational concept: a case of authorities or duties assigned to an agent, whether human or machine. However, as more machines enter the force of the future, it be-
comes necessary to configure them with duties on a time scale consistent with mission planning and execution. We cannot freeze the relative roles of a fighter and a wingman solely during the requirements generation phase. And because autonomy is about the relationship between elements of the force, acquirers cannot fully define roles in the requirements phase except in the most trivial of use cases, like return-to-base behaviors.

With an alternative approach to software acquisition, implementing the imagined mitigations in time to affect an adversary’s success would be possible. And the solution does not depend on centralized management, common standards, or common development environments. Instead, it depends on the organizations responsible for operating the systems involved in the example provided—the Overhead Persistent Infrared (OPIR) program office, the Aegis combat system, the commercial satellite services company, and the program office for counter-air systems—maintaining the ability to continuously deploy updates and configuration changes to operational capability.

This model preserves the software of each of the relevant mission systems in a fluid state. Developers can push out updates to forward aircraft maintainers to load new waveforms, automated flight behaviors, and data processing algorithms onto vehicles. They can test these updates robustly using high-fidelity simulation that can also participate in multi-system live, virtual, constructive testing to vet the latency-critical aspects of satellites and aircraft exchanging data. Program offices can loosely coordinate, allowing engineers to directly cooperate on the details of bit-packing a message for transport without any lengthy coordination of a consensus decision-making body. In short, with the right approach for software acquisition, the US will have superior combat adaptability and the ability to update and define new tactics on the fly.

The essential feature revealed in this section is that advantage in future military contests is less likely to be the presence of robotics or AI in military systems and more likely to be effective employment of these (and other) components as part of a force design, and specifically the sufficient maintenance of adaptability in the creation and implementation of tactics.

The Military Motivation for Adaptability
Scientists and strategists have long studied the role of adaptability in military contexts. In perhaps the defining book studying the topic of adaptation of militaries, Williamson Murray writes:

In Clausewitzian terms, war is a contest, a complex, interactive duel between two opponents. It is a phenomenon of indeterminate length, which presents the opportunity for the contestants to adapt to their enemy’s strategy, operations, and tactical approach. But because it is interactive, both sides have the potential to adapt to the conflict at every level, from the tactical to the strategic. Thus, the problems posed by the battle space do not remain constant; in fact, more often than not, they change with startling rapidity. Moreover, war in the past two centuries has seen an increasing pace of adaptation, as military organizations confront not only the problems posed by their adaptive opponents [but also a changing technological and economic environment].

This excerpt introduces the idea that change is an essential feature of conflict. No a priori prediction or perfect requirement can accommodate all potential futures. Instead, honing an ability to adapt to an uncertain future delivers a military advantage. Theorist Steven Rosen suggests that military innovation in peacetime is concept-driven; it is motivated by watching the external environment—what adversaries are doing and what technology is capable of. This contrasts with wartime adaptation, which is driven by direct exposure to adversary engagement. This suggests that the present moment is the right one to configure software acquisition and development practices to deliver future adaptability.
In his book examining how the war on terror helped China develop concepts to engage the United States in war, David Kilcullen likens wartime adaptation to evolutionary processes, with the selection pressure of combat effectively rendering extinct outdated tactics resulting in casualties and equipment losses.\(^2\) Faced with certain surprise in wartime, a contestant must choose between defeat and adaptation and perhaps hope that their ability to adapt will enable a surprise of their own.

Drawing on DARPA’s Mosaic Warfare concept, Chris Brose popularized discussion of military kill chains,\(^2\) describing a kill chain as a three-step sequential event in which cognition, decision, and finally action yield an objective. This functional decomposition suggests that flexibility and adaptability of the force drive the outcome of future conflict more than strength and quantities of force. The military that fields the more adaptable kill chain enters the fight with a concrete advantage. In a sobering acknowledgment, Brose shares his belief that the US defines its kill chains too rigidly, opening the door for adversaries to devise ways to neutralize our kill chains and, in effect, neutralize our technological advantages.

**A Troubled Relationship with Prediction**

If adaptability is so important, then it is logical to seek to measure it. Indeed, just writing a requirement that software systems be adaptable might even make sense. But here the paradox of adaptability emerges—that one cannot write a verifiable test condition for adaptability, and therefore one cannot write a valid requirement for adaptability. A test for adaptability would presumably measure the ability to overcome some adverse impact, but a verifiable version of this test would require that the specific adverse impact be known in advance. With the foreknowledge of the specific adverse impact, the problem shifts to engineering mitigations for this impact, which could be worthwhile but has nothing to do with adaptability. Properly speaking, adaptability is about overcoming unforeseen events that we have not adequately predicted. Adaptability enables overcoming predictive failure.

Adaptability also enables other subordinate concepts, especially resilience. It can enable the positive effect of inflicting surprise by modifying capability. Resilience focuses specifically on the capacity to overcome the negative effects of setbacks, damage, and loss to maximize military performance and combat effectiveness. This report examines both adaptability and resilience in the context of technology-enabled force design, namely several software-enabled mission systems that collectively enable a successful military mission, likely supported by military personnel and host platforms.

Napoleon’s statement “To be defeated is pardonable; to be surprised—never!” reveals that good military officers often gain great value from planning and attempting prediction, and perhaps that he believed he could somehow escape that intrinsic feature of warfare—surprise. The DoD has a particularly strong affinity for prediction, partly arising from its bureaucratic mechanics. It assumes the requirements process, in particular, can identify and translate future gaps into predictions for weapons systems that perform in a way that satisfies them. The Planning, Programing, Budgeting, and Execution (PPBE) process then transforms this requirement into a contract between Congress and the program office assigned execution responsibility. This contract notably has predicted the specific phase of development and expenditure rate as many as eight years in advance. Then the program manager must defend program execution to an obsolete baseline, an obsolete requirement, and an obsolete prediction of the future that she did not generate. That defense acquisition works at all under such circumstances is underappreciated. Andrew Krepinevich elucidated the incentives for this bureaucratic behavior, suggesting that an employee in the DoD has an incentive not to think about the future because doing so “implies things might change and they might have to change along with it, [and to] the extent they ‘tolerate’ such thinking, they attempt to ensure that such thinking results in a world that looks very much like the one for which they have planned.”\(^2\)

In short, the problem of being ill-prepared for surprise is not the attempt at prediction but the hubris of believing that one’s pre-
dictions are correct. This belief interferes with the constructive feedback cycle that drives learning: making trials, observing results, correcting predictions, and trying again. Unfortunately, the DoD has developed a number of incentives and processes that fall into this dangerous trap, for historical reasons that former Navy Secretary Richard Danzig summarized:

A half century ago, Robert McNamara and his “whiz kids” intensified the predictive tendency, but for different reasons than their predecessors. For McNamara and his colleagues, the challenge was to take an internally competitive, substantially disorganized and significantly dysfunctional DOD and make it more manageable and rational. A key step to this end was to adopt the then-modern concepts of strategic planning with which McNamara had been closely associated at Ford Motor Company. A related initiative was to establish for DOD a single scenario—a Soviet invasion of Western Europe—against which most investments could be measured. This mechanism of resource allocation became a mechanism of program planning in accord with the proposition that “what you measure is what you motivate.” This result was rationalized with the observation that the Soviet scenario was so stressful that all other contingencies would be lesser included cases; they could be readily handled with the equipment, training and doctrine designed for the most demanding Soviet scenario. Of course, this scenario was never as dominant in practice as it was in theory.

This history also highlights the offensive power of adaptability. In the case of the Cold War, at least part of the military advantage came from pressing strategically chosen investments (e.g., stealth strike capability) that the Soviet system reacted to in a predictable, non-adaptive way (building out expensive and decreasingly effective air defense systems). The key is that effective use of adaptability binds an adversary to their predictions and pre-established plan.

Fortunately for the United States, the Cold War featured an adversary that was almost uniquely bureaucratic, ponderous, and predictable, which enabled a relative advantage. This is by no means an endorsement of the predictive power of the DoD, which failed to anticipate the breakup of the Soviet Union or the rapid rise of China as a threat.

Implementing and Measuring Adaptability

While changing the nature of a bureaucracy is hard, enabling better adaptability in systems during their development and acquisition is eminently feasible. As this report will go on to show, a core enabler will implement modern DevSecOps practices. The fundamental principle behind this measure, however, is keeping some aspects of a system relatively easy to change. Indeed, the concept of an “architecture” can be thought of as the relationship between things that are easy to change and those that are hard to change. As development of any system progresses, developers and engineers make choices that freeze aspects of the design.

Table 1 illustrates some aspects of military systems that are relatively easier or harder to change. While any software project becomes harder to change over time, it is now possible to maintain more aspects in a fluid state. Choosing these aspects and allowing for their continual modification will be the essence of an adaptability advantage.

As joint warfighting and multi-element kill chains, possibly enabled by uncrewed systems and artificial intelligence, become part of the future of warfare, it implies an architectural shift from relatively monolithic software implementations that were tightly coupled to extract performance in SWAP-constrained systems toward less stateful and more service-oriented or API-enabled approaches that enable recomposition of the underlying mission system capabilities in weapons. As military

SOFTWARE DEFINES TACTICS: STRUCTURING MILITARY SOFTWARE ACQUISITIONS FOR ADAPTABILITY AND ADVANTAGE IN A COMPETITIVE ERA
Adaptability is in the class of -ilities that we cannot directly measure, and cannot require, but can characterize with the help of fitness functions. DARPA has previously explored schemes for measuring adaptability of systems. One promising approach that emerged was to use the time required to respond to a previously unanticipated requirement—the time to react to surprise. The Air Force’s Kessel Run project demonstrated the ability to deploy aircraft operations planning tools overnight to accommodate a previously unanticipated need during the Afghanistan evacuation, essentially overcoming predictive failure. Netflix’s approach to engineering resilience in its systems by using a “chaos monkey” to inject surprise via severing processes and links is a related fitness function that holds engineers accountable for architecting a resilient system. It’s not surprising that time (or time to respond) emerges as the metric for adaptability. This aligns with the finding of the Defense Innovation Board that speed and cycle time are the most important metrics for managing software.

Table 1: Gaining an Adaptability Advantage across a Force Design Involves Pushing Some Elements of Capability to Regions of the Design Space Where Change Is Relatively Easy

<table>
<thead>
<tr>
<th>WHAT IS HARD TO CHANGE?</th>
<th>WHAT IS EASY TO CHANGE?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The outer-mold line of a stealth fighter</td>
<td>Which human operators are assigned to a specific mission</td>
</tr>
<tr>
<td>The gauge of rail lines used for logistics in a given country</td>
<td>The configurations file called when a system initializes</td>
</tr>
<tr>
<td>The composition of steel used in the hull of a submarine underway</td>
<td>The source code for a signal processing algorithm on an intel analytics project with a continuous delivery pipeline</td>
</tr>
<tr>
<td>The computing capacity of a processor deployed to a satellite</td>
<td></td>
</tr>
</tbody>
</table>

Source: Authors

capability becomes disaggregated (a satellite sensor passes data to a fusion engine hosted onboard a command-and-control aircraft, which passes a target coordinate to a fire control computer on a mobile missile launcher), the computing aspects are also likely to become distributed. This further implies that not all of the elements are under the control or authority of one program executive officer, or will be designed to the same “global” standards. Pragmatically, this creates a mandate to adjust the software architecture and development environments to allow for more decoupled development between distinct, heterogenous systems.

Fielding capability is not just about development, of course. It needs to accommodate a whole range of -ilities. Here again, we find technological developments that begin to make this practical. Automated tests that are part of a continuous delivery pipeline can approximate many of these. Development must also include the creation of environments that allow for these automated tests. We can consider them as miniature “fitness functions” that help shape the evolution of a body of code by providing feedback on the impact of changes. These will span in scope from code unit tests all the way up to live, virtual, and constructive (LVC) environments that allow testing the performance of multiple mission systems from different program offices together.

Change is a certain feature for the future, stemming from commercial innovations, adversary developments, and brilliant new ideas. No data schema, interface standard, programming language, or tool set will remain timeless in the face of these changes, but some will change faster than others. The United States can extract military advantage by un-
derstanding how to ensure adaptability in software-enabled systems.

There is a longstanding and fundamentally flawed notion that the DoD should standardize acquisition processes and technologies across all its activities. One of the principal conclusions of the Defense Science Board (DSB) in its 1987 review of software development and acquisition for the DoD perhaps best represents this idea:

It is very important for DoD to have a standard programming language; Ada is by far the strongest candidate in sight.

This quote is the epitome of flawed thinking about homogeneity. The document went on to lament the period when the DoD could effectively set standards for the commercial software development ecosystem, noting Navy Rear Admiral Grace Hopper’s role in creating the COBOL language that was widely adopted in commercial practice. The DSB sought to reassert a military role in defining standards that could shape developments in the commercial sector.

With the benefit of 35 years of hindsight, most readers will intuitively recognize the folly of trying to constrain all of computer programming to a single language controlled by the DoD. Moreover, if the DoD attempt at asserting centralized control over development had succeeded, the world might have never known many of the innovations in programming languages that have occurred in the three decades since—from C’s approximation of a portable assembly language with high efficiency to the high-level languages (like Python) that speed development and massive libraries (like react.js) that power rapid development of the modern web.

Of course, modern software development combines multiple languages in single projects; for example, one web application might use JavaScript for front-end development, Python for data processing, and PHP for server-side scripting. The debate about a single standard turns out to be frivolous, and the DoD would do much better by spending time understanding which tools fit which jobs. This extends well beyond programming languages and into the massive decomposition of software tooling that has occurred over the years. In the 1990s, the client-server architectural style saw singular tools capture programming teams. Tools like FoxPro, PowerBuilder, and Delphi were homogenous programming environments—front end, business rules, database queries, compiling, testing, source control—in which everything was built into the tool, and the idea of third-party tool chains was nascent at best.

Today, every part of the development process relies on different tooling. One supply chain source provides the source code repository tooling, while other sources provide the integrated development environment (IDE), infrastructure automation, cloud tooling, user experience (UX) design features, and user interface (UI) testing. Where the architectural quanta used to be an ordinal value of it, today’s modern and adaptable development environments pull together dozens of tools. Modern software development activities are more logistically focused than at any time in the past. Today’s software development environments are so heterogenous and adaptable that they are more akin to digital logistics.
Software delivery is about getting the right bits to the right places at the right time

An old principle of warfare suggests that outcomes are less about the forces you have than about your ability to keep them in motion and employ them effectively. The pithy quote “Infantry wins battles, logistics wins wars,” attributed to Army General John J. Pershing, commander of the American Expeditionary Forces on the Western Front during World War I, summarizes this concept.

But what of modern military capability? It still needs infantry, but it also needs to pull data from a satellite imaging system, connect a radar track to a missile battery, and deliver a moving map with adversary locations to an infantryman’s tablet.

As the previous chapters have shown, software is not only transforming the economy but becoming deeply intertwined with military capability. Much like logistics is about moving materiel to deliver capability to the time and place of greatest need and impact, software delivery is about delivering software artifacts to the time and place (host environment) of greatest need and impact. And much like in logistics, the mastery of software delivery can be a war-winning (or -losing) capability. Also similar to logistics, the DoD needs to lay the groundwork for software delivery in peacetime to enable the strategic advantages of superior adaptability described in the previous chapter.

Reflecting on Logistics

The logistics system has evolved over centuries as depicted in figure 4. Inventions gave rise to networks, which led to systems and then systems-of-systems. As system designers coupled

Photo Caption: A-10 software engineers work together in teams to verify functionality of organically developed operational flight program software on November 5, 2020, at Hill Air Force Base, Utah, before it is released to the warfighter. (US Air Force photo by Alex R. Lloyd)
more elements, complexity increased, and design spaces became more open.

Notably, the logistics system is much more than a collection of ships, trucks, and airplanes. It couples together the highway network, rail network, air transport network, and more—a rich network-of-networks for different transportation modalities. Further, it is a complex system-of-systems run by software connecting many systems and networks to maximize flexibility. For example, the military combines its dedicated transportation capabilities with long-term commercial contracts and on-demand indigenous support from host countries.

Layers of information systems stitch together this enterprise: A military software system automatically issues orders, and a commercial transportation management system receives them. Third parties collect worldwide location information, fuse it to make delivery time predictions, and sell it back to enable superior inventory management. Notably, diversity and complexity are an essential feature of the logistics system.

Breaking this down to a simpler use case as shown in figure 5, readers can see intuitively that embracing this inherent diversity of transportation modes and coordinating across them is a feature. Key performance parameters like distance and payload help drive choice, but so do availability and personnel familiarity.

There are hidden dimensions to the logistics problem as well, as depicted in figure 6. An organization may acquire any given transportation element in many different ways—choosing to buy a service, or operate the vehicle itself, choosing to buy a vehicle off-the-shelf, custom-build, or rent. It would be irrational to imagine designing a logistics system of any reasonable scale that is homogenous in nature—using only one transportation
Figure 5 Logistics Coordinates Activity across Transport Modes to Deliver Goods

<table>
<thead>
<tr>
<th>What are you trying to do?</th>
<th>How could you do it? What are the implementation options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get across town with a letter</td>
<td>Skateboard, Scooter, Bike</td>
</tr>
<tr>
<td>Get across town with two pieces of luggage</td>
<td>Cargo Bike, Bus, Car, Pickup</td>
</tr>
<tr>
<td>Get across town with two pallets of supplies</td>
<td>Pickup, Box Truck</td>
</tr>
<tr>
<td>Get across the country with 20 pallets</td>
<td>Semi Truck, Train, Airplane</td>
</tr>
<tr>
<td>Get across the globe with 100 pallets</td>
<td>Airplane, Containership</td>
</tr>
</tbody>
</table>

Source: Authors.

Figure 6. Logistics Deals with Links, Coordination, and Flow between Elements

Logistics concerns the flows of physical goods to meet the needs of a diversity of end-users

Acquisition of any element has many options to consider

- Own & Operate?
- Own & Contract Operation?
- Rent & Operate?
- Custom or Off-the-shelf?
- Procure as a Service?
- Shared Service?
- Dedicated Service?

Source: Authors.
mode. Instead, the ideal is the right modes for the right parts of the problem.

**Similarity to Software Systems**

As illustrated in figure 7, software delivery has more than a passing similarity to delivery of goods. Many forms of application and target architecture are available. Just as it would be irrational to have the military pick one mode of transportation, it is irrational to ask the military to standardize one solution architecture from the set (mobile phone, tablet, laptop, desktop, browser, augmented or virtual reality device, middleware, embedded, firmware). Instead, effective mastery of software systems involves coordination across multiple modalities.

For example, not all military software will be delivered and consumed as cloud-hosted web applications. In figure 8, a programmer might use a desktop environment to commit changes to a cloud environment, which also draws in libraries and middleware from other communities. The same cloud environment might host a web application that provides real-time status updates for a deployed squadron. Someone forward-deployed on the squadron might use a laptop computer to pull a feature release from the cloud when connectivity is available, then configure an embedded system on an aircraft prior to its mission without connectivity.

The choices of transportation modes are not only about distance and payload; they can be about changing circumstances as well. Generically, we can describe these changes of circumstance or need as gradients. A Silicon Valley resident might rent a scooter to get to the bus stop, take the bus to the train station, take the train into San Francisco, then order an Uber because it is now raining and the destination is uphill. Changes in environment drive each decision to transition.

As the military recognizes that it operates using systems-of-systems, which implies a large number of gradients, it also needs to accommodate that. A future aviator might use a mobile phone

![Figure 7. Software Delivery Coordinates Activity across Modes](source: Authors)

<table>
<thead>
<tr>
<th>What are you trying to do?</th>
<th>How could you do it? What are the implementation options?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I need a software solution to get user input on the move</td>
<td>Native Mobile App</td>
</tr>
<tr>
<td>I need a software solution that is available regardless of compute platform</td>
<td>Progressive Web App</td>
</tr>
<tr>
<td>I need a software solution and it must process massive datasets</td>
<td>Browser App</td>
</tr>
<tr>
<td>I need a software solution and it must be able to do image recognition in a disconnected state</td>
<td>Middleware</td>
</tr>
<tr>
<td>I need a software solution with 4K video editing</td>
<td>Embedded</td>
</tr>
<tr>
<td>I need a software solution and it must augment the physical environment</td>
<td>Laptop App</td>
</tr>
<tr>
<td>I need a software solution highly optimized for a custom radiation-hardened chip headed to the moon</td>
<td>Desktop App with GPU support</td>
</tr>
<tr>
<td>I need a software solution that is available regardless of compute platform</td>
<td>AR/VR App</td>
</tr>
<tr>
<td>I need a software solution and it must process massive datasets</td>
<td>Firmware</td>
</tr>
</tbody>
</table>
models, and other supporting technologies. In the face of this complexity, it simply is not realistic to assume there is one best software delivery solution for the DoD. Instead, acquisition personnel need to embrace heterogeneity to successfully navigate the complexity.

Software Solutions as Combinatorics Problems

Now consider the development of a complete software solution architecture in this ecosystem as a combinatorics problem that every PEO faces. The following is notional and illustrative, incomplete and certainly not normative. But the point is clear to the reader in table 2.

Just performing the combinatoric math on this vast array of choices and considerations suggests that there are more than 6 billion combinations for a PEO to navigate. This is without considering effects of characteristics of data, AI/ML

Navigating the Complexity

Navigating this complexity is possible, but the conversation should be framed around economics, not technology or innovation. At the end of 2021, GSMA reported that 5.3 billion people subscribed to mobile services, representing 67 percent of the global population. When accounting for machine-to-machine mobile subscriptions as well, there will be more than 8.5 billion mobile connections globally by 2023. This astounding statistic means there are more cellular system on a chip (SoC) microprocessors connected to the internet than there are people living on the planet today. In 2021, the industry manufactured over 1 trillion chips, according to ASML, representing 130 chips for every person on earth, and the cost to build a fab topped $14.6 billion. Organizations can realize the massive investments required to advance the state of digital technologies only because of the scale of the consumer market; no DoD acquisition program can possibly compete with this scale or pace.
Table 2: Several Illustrative Dimensions of the Problem Facing Software Program Managers

<table>
<thead>
<tr>
<th>PEOPLE</th>
<th>SDLC/METHODOLOGY</th>
<th>TARGET CPU INSTRUCTION SET FOR THE SOFTWARE SOLUTION</th>
<th>ARCHITECTURAL COUPLING</th>
<th>PRIMARY AND SECONDARY ARCHITECTURAL CONCERNS (THE -ILITIES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 100% military and government civilian coders</td>
<td>1 Waterfall</td>
<td>1 ARM</td>
<td>1 Libraries: “A developer’s code calls library”</td>
<td>1 Affordability</td>
</tr>
<tr>
<td>2 50/50 split of military/government and contractor</td>
<td>2 Scrum/Kanban/Agile</td>
<td>2 x86</td>
<td>2 Frameworks: “The framework calls a developer’s code”</td>
<td>2 Compatibility</td>
</tr>
<tr>
<td>3 100% contractor coders</td>
<td>3 SAFe</td>
<td>3 GPU (CUDA)</td>
<td>3 Extreme Decoupling: NASA WorldWind Java relied on zero libraries and zero frameworks and was entirely self-contained for an extended period</td>
<td>3 Portability</td>
</tr>
<tr>
<td>CI/CD PIPELINE</td>
<td>4 proprietary/Custom</td>
<td>4 FPGA</td>
<td>4 Interoperability</td>
<td>4 Provenance</td>
</tr>
<tr>
<td>1 100% government-owned and -operated CI/CD pipeline</td>
<td></td>
<td>5 ASICS</td>
<td>5 Securability</td>
<td>5 Provenance</td>
</tr>
<tr>
<td>2 50/50 split, with CI owned and operated by contractor, and CD owned and operated by government via support contractors</td>
<td></td>
<td></td>
<td>6 Portability</td>
<td>6 Provenance</td>
</tr>
<tr>
<td>3 100% contractor-owned and -operated CI/CD pipeline</td>
<td></td>
<td></td>
<td>7 Composability</td>
<td>7 Provenance</td>
</tr>
<tr>
<td>ACQUISITION FRAMEWORK</td>
<td></td>
<td></td>
<td>8 Distributability</td>
<td>8 Provenance</td>
</tr>
<tr>
<td>1 Mid-tier Acquisition</td>
<td></td>
<td></td>
<td>9 Exensibility</td>
<td>9 Provenance</td>
</tr>
<tr>
<td>2 5000.87 Software Acquisition Pathway</td>
<td></td>
<td></td>
<td>10 Maintainability</td>
<td>10 Provenance</td>
</tr>
<tr>
<td>3 OTAs</td>
<td></td>
<td></td>
<td>11 Availability</td>
<td>11 Provenance</td>
</tr>
<tr>
<td>CONTRACTING APPROACH</td>
<td></td>
<td></td>
<td>12 Autonomy</td>
<td>12 Provenance</td>
</tr>
<tr>
<td>1 Procured as unlimited rights source code from contractors</td>
<td></td>
<td></td>
<td>13 Correctness</td>
<td>13 Provenance</td>
</tr>
<tr>
<td>2 Procured as a service delivered by contractor</td>
<td></td>
<td></td>
<td>14 Reusability</td>
<td>14 Provenance</td>
</tr>
<tr>
<td>3 Mixed procurement approach</td>
<td></td>
<td></td>
<td>15 Standards Compliance</td>
<td>15 Provenance</td>
</tr>
<tr>
<td>IMPACT LEVEL</td>
<td></td>
<td></td>
<td>16 Administability</td>
<td>16 Provenance</td>
</tr>
<tr>
<td>1 Impact Level 2—Commercial</td>
<td></td>
<td></td>
<td>17 Administration</td>
<td>17 Provenance</td>
</tr>
<tr>
<td>2 Impact Level 4/5—PII/HIPAA/CUI</td>
<td></td>
<td></td>
<td>18 Availability</td>
<td>18 Provenance</td>
</tr>
<tr>
<td>3 Impact Level 6—Classified</td>
<td></td>
<td></td>
<td>19 Responsiveness</td>
<td>19 Provenance</td>
</tr>
<tr>
<td>NETWORK CONNECTIVITY</td>
<td></td>
<td></td>
<td>20 Testability</td>
<td>20 Provenance</td>
</tr>
<tr>
<td>1 Always Connected (Cloud)</td>
<td></td>
<td></td>
<td>21 Autonomy</td>
<td>21 Provenance</td>
</tr>
<tr>
<td>2 Denied, Degraded, Intermittent, or Limited Bandwidth (DDIL)</td>
<td></td>
<td></td>
<td>22 Correctness</td>
<td>22 Provenance</td>
</tr>
<tr>
<td>3 Air Gapped</td>
<td></td>
<td></td>
<td>23 Distributability</td>
<td>23 Provenance</td>
</tr>
<tr>
<td>SOFTWARE ARCHITECTURE</td>
<td></td>
<td></td>
<td>24 Exensibility</td>
<td>24 Provenance</td>
</tr>
<tr>
<td>1 Monolith</td>
<td></td>
<td></td>
<td>25 Maintainability</td>
<td>25 Provenance</td>
</tr>
<tr>
<td>2 Microkernel</td>
<td></td>
<td></td>
<td>26 Reliability</td>
<td>26 Provenance</td>
</tr>
<tr>
<td>3 Event-Driven</td>
<td></td>
<td></td>
<td>27 Reusability</td>
<td>27 Provenance</td>
</tr>
<tr>
<td>4 Service-Oriented Architecture (SOA)</td>
<td></td>
<td></td>
<td>28 Standards Compliance</td>
<td>28 Provenance</td>
</tr>
<tr>
<td>5 Microservices</td>
<td></td>
<td></td>
<td>29 Administability</td>
<td>29 Provenance</td>
</tr>
<tr>
<td>6 Service-Based</td>
<td></td>
<td></td>
<td>30 Availability</td>
<td>30 Provenance</td>
</tr>
<tr>
<td>7 Serverless</td>
<td></td>
<td></td>
<td>31 Demonstrability</td>
<td>31 Provenance</td>
</tr>
<tr>
<td>8 Predictability</td>
<td></td>
<td></td>
<td>32 Durability</td>
<td>32 Provenance</td>
</tr>
<tr>
<td>SYSTEM CATEGORIZATION OF THE SOFTWARE SOLUTION</td>
<td></td>
<td></td>
<td>33 Orthogonality</td>
<td>33 Provenance</td>
</tr>
<tr>
<td>1 Enterprise System</td>
<td></td>
<td></td>
<td>34 Serviceability</td>
<td>34 Provenance</td>
</tr>
<tr>
<td>2 Weapon System</td>
<td></td>
<td></td>
<td>35 Traceability</td>
<td>35 Provenance</td>
</tr>
<tr>
<td>3 Life-Critical System (human in the loop, human must return home to family)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Authors.
Consumers realize the true value of all of these chips only through the software applications running on them. By 2025, these chips and the software will produce 175 zettabytes of data per year.\textsuperscript{32}

These statistics portray the economic landscape that the DoD and its collection of PEOs find themselves operating in today. The DoD was once the proverbial 800-pound gorilla; landing a DoD contract justified ramping up production. This just is not the case anymore. Today, the economic landscape has changed so dramatically that the global chip foundries and massive global software organizations like the Cloud Native Computing Foundation (CNCF) could more accurately describe the DoD’s consumption as a rounding error. The most suitable word for the effect of this economic landscape is forage; the DoD has to forage for access to chips and software solutions these days because landing the DoD as a customer is not nearly as economically exciting as targeting an enormous consumer market.

The Government Accountability Office stated in a 2022 report that the DoD plans to acquire 2,500 F-35 aircraft.\textsuperscript{33} The precise number of chips in the F-35 design is not publicly available. However, even in a ridiculously concocted hypothetical scenario in which the DoD seeks to purchase 10,000 chips per aircraft, that’s only 25 million chips, or roughly 0.000025 percent of the trillion chips manufactured in 2020. The model year 2020 Mercedes S-Class is reported to include over 30 million software lines of code.\textsuperscript{34} In contrast, the F-35 is reported to have only 8 million software lines of code.\textsuperscript{35}

Viewed through an economic lens, the consumption of chips and software by the most technologically advanced platforms under development by the DoD pale in comparison to the total addressable market in the commercial space.

Only a few decades back, this was not the case—DoD products were at the forefront of new-scale fabrication processes and facilities. We cannot lament this change or attribute it to DoD dys-function; instead we need to recognize the virtuous economic cycles spun up by information technologies that create a massive commercial market. And we should also accept the changes it implies. Economically speaking, the DoD has to forage for dual-use technologies, and this is where both the greatest opportunity and the gravest risk to PEO forecasting exist.

**Finding Economies of Scale**

Traditionally, the DoD approach to software delivery was to distribute funding and requirements to a program office, which issued contracts with discrete deliverables that could meet said requirements.

The downside of this approach is that it treats weapon systems as independent, as snowflakes derived from requirements. Only the rare program manager would seek out commonality and opportunities to reuse other building blocks. This is the model that yields a littoral combat system with a control console that comes fully bundled—screens, CPU, throttles, operating system, and software with user interface to ship machinery that is integrated and tested by the prime contractor. As time progresses, though, the fleet and the program office must struggle to patch the operating system, replace a failed memory chip, or interface with an updated valve servo controller that uses a different message protocol. Each of these requires contract modifications to the original prime and an updated snowflake that fixes the problem.

An attractive alternative seeks an analogy between a commercial company and the DoD, depicted as the centralized model in figure 9. No modern nimble start-up would dream of development in the snowflake model. Instead, the founders would choose a cloud provider and build out a development platform, and as employees come onboard and products diversify, the company would build a whole set of applications on top of this cloud infrastructure and platform services. This has the advantage of low incremental costs for every new application—there is no need to reinvent the wheel and build out a new platform.
for each new application. Presumably, the DoD would contract for centralized secure cloud services, build out its own joint common foundation for platform services, and either centrally fund or mandate use of these services. This could enable rapid collaboration and data consolidation and is especially conducive to centralized applications, like many commercial ML approaches.

Unfortunately, the centralized model is unworkable. The needs of the DoD are simply too diverse. There are legitimate needs

Note: Any software application depends on supporting platform and infrastructure. Neither fully federated nor fully centralized approaches make sense. Instead, similar efforts are more likely to share platform or infrastructure.
to operate at a variety of security levels, and it is impractical to simply elevate the security privileges of all users and applications. Some DoD software needs will be humble, like embedded software for servo controllers to point antenna arrays, which will need to operate in an air-gapped environment yet still be capable of future updates to handle patching. This cannot be forced through a monolithic development pipeline for web applications. Additionally, the centralized model fails to fully leverage the benefits of commercial development. In some instances, commercial applications will be well suited for use in a disconnected secure network. In other cases, a commercial data extract-transform-load or AI model-building tool will be perfectly suited to serve a secure application. In other cases, the DoD could be developing its own applications on a quasi-public cloud for use by allies and partners. Put another way, the DoD is more like an economy than a company.36

Thus, this report proposes a third model—an opportunistic model—in which, like the snowflake model, the burden of choosing a path still falls to the PEO with the requirement. But in this third model, the PEO asserts their role as the final integrator of the delivered capability, and shoulders the responsibility for seeking out available and similar infrastructure and platforms from partners. This is the most workable model, providing both the developmental efficiency and fiscal responsibility that taxpayers demand. For most large acquisition efforts, this will likely involve establishing a software factory or finding a suitable existing one. The DoD will almost certainly make mistakes, as when the Air Force’s Kessel Run initially chose Pivotal Software, which Dell has since acquired, to provide its platform before eventually realizing that this compromised the government’s needs to continue to integrate, maintain, and deliver the overall capability. While it lost some time due to the mistake, this was part of the learning process, and ultimately Kessel Run emerged in a stronger position. Silicon Valley start-ups often make similar foibles; for example, Snap reversed its decision to go all-in on Amazon Web Services and recognized its need to diversify its infrastructure providers.38 The external environment and available infrastructure and platforms will continue to shift through the course of any development effort.

**Evolutionary Architectures**

A common fallacy is seeking to fully define a software architecture before starting software development. This draws from flawed but conventional wisdom on the importance of getting the requirement right before starting a hardware acquisition. Of course, it is possible to make colossal architectural errors at the outset of an effort, but these errors have severe consequences only if architects doggedly adhere to flawed choices in the face of evidence. Experienced software architects will readily admit that the best architectures are evolutionary, rarely rigid over the life of a software application.

It is easy to get caught in prediction traps and to ignore the fact that future-proofing in software development makes it more difficult for developers to use the code for any single purpose. Future-proofing can make it harder to deliver an initial working product and collect feedback. In many cases, it is better to start solving an important problem and to accept a marginal amount of technical debt. Those same software architects from above will also say that premature optimization is evil.

Architectural patterns show up in logistics too, of course. The major commercial logistics company FedEx does have to plan its investment in air transport capacity, contracted surge capacity, ground transport capacity, sortation capacity, and so forth. But it does not try to predict the right path for each package years ahead of time. FedEx has made some choices that are frozen—hub airports, air fleets, major sortation facilities—but has kept open the ability to flow any individual package over many paths depending on the superior knowledge of the present moment. It handles uncertainty by flying empty aircraft from its hub that can divert to satellite airports in case of mechanical failure or excess demand. While leaders made some choices early in the history of the company—including selecting Memphis as a hub and enabling overnight delivery—much of the physical architecture
of FedEx emerged in the decades that followed as the company evolved and discovered new markets in international delivery, home delivery, and palletized load delivery. Even as a company heavily invested in capital infrastructure, FedEx did not start by defining its architecture; it focused on solving customer problems. Instead of defining and reviewing software architectures in a waterfall fashion, a PEO can better spend their time deciding which of their desired outcomes and user needs is most important, which future development paths are most uncertain, and which aspects of their program will most likely need to change. Time spent on these questions, coupled with initial development, is how architectures will emerge.

The focus on where the initial development occurs is a complicated matter. Abe Karem, the ex-Israeli aircraft designer who created the Predator unmanned aerial vehicle (UAV), famously stated that “the defining characteristic of a good engineer is their laziness; mediocre engineers try to do all of the work themselves.” By extension, the defining characteristic of a good PEO is a desire to extend what exists instead of doing all of the work themselves. This amplifies the sentiments of senior leaders such as Renata Spinks, cyber technology officer to the US Marine Corps Forces Cyberspace Command:

The Marines do not want to do anything all by themselves. We are a team-of-teams institution. Every platform is different, and none of them are one-size-fits-all. If there’s a capability or a gap, the first thing we do is go see who else has this and what does “right” look like.

Over the last few years, the number of cleverly named software factories across the DoD has rapidly grown to at least 30 today. Others across the Air Force, then across each of the other services, and finally across the DoD’s fourth estate quickly followed the trail that Kessel Run blazed. How does a PEO explore, quantify, and then rationalize the logistics required for a software factory that can complete the initial software development tasks?

Over a decade ago, John D. Cook postulated that software reuse is better described as performing an organ transplant than snapping together Lego blocks. By extension of his analogy, the modern software factory likely has more in common with a Level 1 trauma center than a Legoland Discovery Center. As software continues to permeate the fabric of every program, each PEO has to confront complicated software factory logistics and choose a path: build or reuse.
The term software factory has an industrial-age ring to it. Perhaps it inspires visions of an iron boiler providing steam to power a hot, sweaty building where programmer-workers toil monotonously in round-the-clock shifts, producing near-identical chunks of software to meet the insatiable market demands. More pragmatically, one might imagine government-badged coders wearing hoodies and sitting in an open-plan office setting, banging away on keyboards to produce non-proprietary, government-owned software in order to prevent vendor lock from greedy contractors. This, however, would be a fundamentally mistaken vision of the software factory. Instead, a software factory is a process and supporting tool that provides software development teams with a repeatable, well-defined path to create and update production software and deliver it in a robust and resilient manner.

We intuitively recognize what happens when a production line closes down, whether it’s due to a temporary government shutdown or a decision to curtail production of an F-22 fighter. Rehiring the workers, rebuilding the tacit knowledge of assembly, or making the facility productive again is very difficult. A full shutdown of an aircraft assembly line is no different than a full shutdown of a DevSecOps factory. The libraries and frameworks they used become stale, unsupported, brittle, and vulnerable. Someone cannot just “turn it on” again and start spitting out software updates. Even while the source code for a program has not changed, the build tools and dependent libraries have done so continuously. Those changes have now created broken, incompatible application programming interfaces and other factors.

The digital world is relentless in its ceaseless change as updates and dependencies cascade around the globe and round the clock. So older software development paradigms—with the right compiler and source code—have had to adapt to new ways of designing, building, and managing software. Bri-

Photo Caption: Judges representing the active-duty Air Force, Air National Guard, and the Air Force Reserve deliberate on presentations given during Hack the Ranch, a hackathon hosted by Corsair Ranch, the Air Force Reserve Component’s software factory. (US Air National Guard photo by Senior Master Sgt. Charles Givens)
Figure 10. False Impressions of a Software Factory, As Imagined by Midjourney AI Using the Authors’ Misleading Prompts.

an Kernighan—the K in the foundational K&R, the C programming language book that yielded the so-called “one true brace” style of programming—recently tried to contribute code to the text-parsing tool AWK that he created decades ago. AWK now represents a foundational element of Linux scripting, supporting cloud computing management among countless other projects. It is a foundational dependency, but it still has to be updated occasionally. Kernighan said, “Once I figure out how . . . I will try to submit a pull request. I wish I understood git better, but in spite of your help, I still don’t have a proper understanding, so this may take a while.” Even the true legends of the software community reach a point where their skills become stale when measured against modern development practices.

Ukraine’s use of Stinger and Javelin missiles perfectly illustrates that the challenge for digital components of military systems is far greater than simply turning the lights on and dusting off the equipment. Raytheon has openly stated that companies have stopped manufacturing key components, forcing the company to “redesign electronics in the missile’s seeker head.” The Iowa Army Ammunition Plant has been undergoing “wholesale renovation” for several years now, and the industry has lamented the fierce challenges of modernizing a plant while ramping up production to meet demand with inexperienced workers and securing the massive 19,011-acre facility against attack.

Analogously, even if a PEO takes possession of all source code, build scripts, libraries, frameworks, etc., that act alone does not guarantee that the software’s CI/CD pipelines will whirl into operation several years later or that a workforce with up-to-date skills will be around. APIs change, and security patches stop being issued. For each legend like Kernighan who struggles with git, a foundational software development tool by modern standards, there are thousands if not tens of thousands of software developers who lack either exposure to or understanding of stale software environments that were hugely popular only 10 to 20 years ago (see CORBA and Enterprise Java Beans, for example).

Pause and reflect on this fact: The DoD is proudly home to software like Mechanization of Contract Administration Services (MOCAS), which developers were writing around the time the Soviet Union launched Sputnik in 1957 and which demonstrates the staying power of COBOL within the DoD. This is not just any software, either. It is mission-critical software that manages around $1.3 trillion in obligations spread across 340,000 contracts.

How the Ebb and Flow of Feature Development Affects Staffing

Like manufacturing factories, software factories require significant financial investment and the application of active management techniques to absorb the natural ebb and flow of major feature developmental periods as well as periods of patching and operational sustainment. This aspect of the software factory is so critical that leaders must chant it as a mantra regularly: if software is never done, then we can never mothball or defund its software factory for extended periods. In an era when software defines tactics, opining that it might be “a year or two” before a DevSecOps factory can be turned back on is irrational. PEOs should carefully study and consider how the traditional manufacturing workforce migrates from prime to prime as a program is re-bid over the years. They should also recognize that they will likely need to support this behavior with DevSecOps factories, too.

The program needs some people all of the time (e.g., System Reliability Engineers, or SREs) but does not need all people all of the time (e.g., software architects, data scientists, AI researchers, etc.)

The uneven distribution of work is especially apparent in software because the program needs some people all of the time but does not need all people all of the time. The simple recognition of this fact leads to an intriguing debate about the actual role of the DoD in software development. There is widespread
recognition that industry is adept at corraling the requisite cutting-edge talent to contribute the big, shiny, new capabilities and features, the corollary of which is that the modern DoD is not known for introducing cutting-edge software innovations to the commercial world. There is also a consensus that the DoD’s military and civilian leaders evaluate and define its risk and cyber-defense posture. For example, the DoD may turn the dial based on world events and declare a POND. This same group of professionals is largely capable of operating the day-to-day routine patching and sustainment of an application ecosystem, specifically the necessary work that cutting-edge talent tends to avoid because it does not allow them to exercise critical thinking and design skills above the waterline separating innovation from routine (see figure 11).

Recognition of the high variability of work within a software factory is a concept so critical that we capture it as Factor 1 in what we call the Two Factors Principle. Factor 2 is the implementation model of the software factory itself. In the early days of Kessel Run, the Air Force’s leading software factory, the government was its predominant owner and operator. This arrangement contrasted with many programs in which the software ecosystem is primarily contractor-owned and contractor-operated (COCO) model. The characteristics of the COCO model can be detrimental to the government’s needs and taxpayer interests due to six "empirically demonstrated" vulnerabilities in government-funded COCOs. Each of the six vulnerabilities relates to the modern software factory, but one in particular stands out as a significant concern: a contractor’s ability to unilaterally decide to leave the market. Using MOCAS as an example, a COCO software factory asset focused on COBOL development in 2022 would draw significant shareholder scorn and risk simply being shut down for the sake of profitability.

Figure 11: Staffing and Program Costs for Program’s Lifecycle

![Figure 11: Staffing and Program Costs for Program’s Lifecycle](Image)

Source: Authors

Note: Staffing needs and program costs will go up and down across the lifecycle of a program; design a contracting and support structure that will accommodate these shifts.
Our premise in this report is that the DoD has to operate as integrator for all critical combat software capabilities to mitigate the six vulnerabilities of relying on contractor-delivered binary software or even on COCO software factories. Behind closed doors, the defense industry voices concerns about this model. They usually frame their argument in self-serving terms, and some see it as a threat to both their operational profit models and their ability to generate value from their own intellectual property and capital assets. On the other hand, the DoD has well established its angst over sharing software designs only to have them come back stamped as proprietary information. Implementation model selection is a critical aspect that PEOs cannot afford to marginalize. A PEO who opts for a pure government model risks alienating a statistically significant portion of the defense industrial base (DIB). A PEO who opts for the COCO model could be “giving away taxpayer money to a private corporation to build a capital asset that the government does not own or control necessarily.”

There is a third model that has not been aggressively explored in the era of intangible data and software: the government-owned, contractor-operated (GOCO) model. GOCO facilities are not some radical idea and in fact can trace their roots all the way back to the Manhattan Project. The GOCO model may provide
the win-win scenario for both the DoD and the DIB. Consider that beyond the Iowa plant cited earlier; the government also owns plants in Scranton, Radford, Kansas, and elsewhere—all of which contractors operate. This model extends beyond the DoD to the Department of Energy (DoE). The DoE owns the Nevada Test Site, Los Alamos National Lab, and several other major national security assets, but has contractor assistance in operating them under the GOCO model.

The DoD should own its software factories, and the DIB should not irrationally fear this construct. Industry can and should play a vital role in contributing innovations during periods of feature flow; they can bring their cutting-edge software to the factory, and even retain their intellectual property. Adoption of a GOCO model also mitigates the risk of losing access to a software factory ecosystem when DIB executives grapple with their fiduciary obligations to cut costs, raise profits, and issue dividends to institutional shareholders. In short, a commercial entity cannot decide to sell a government-owned asset. This barrier allows the DoD and its PEOs to ensure that the software factory remains in operation during both periods of feature flow and periods of sustainment, readily available to support the warfighter in their mission.

The Two Factors Principle is a nod that each factor cannot be described independently of the other. The DoD's modern software factories should not and cannot be exposed to capitalist behaviors because the DoD is not a capitalist organization—it is a monopoly with statutory authorities to conduct war when authorized by civilian leadership. Active kinetic warfare is not a perpetual activity, and neither is active software feature development. All software, from mobile apps to operating systems, cyclically operates across periods of feature development and sustainment, and the realization of a series of GOCO software factories would accelerate software development best practices and perhaps even the DoD's digital modernization aspirations.

The outcome that the Two Factors Principle embodies is that DevSecOps factory model selection has to offer assurance that the software factory can never be fully shut down at any point during the life of the program and that the staffing ratio of the program will naturally ebb and flow in response to periods of feature innovation and periods of routine sustainment.

**Breaking the Caste System and Acknowledging That Development Is Production**

Too many visionary leaders have fallen into the trap of perpetuating the belief that development environments are inferior to production environments. Senior executives do not get notifications when a development environment suffers an outage, but they get calls in the middle of the night when a production outage occurs. The DoD treats development environments as so-called permissive environments for doing all types of so-called risky work, like using a software compiler or testing websites using a nonstandard port like 8888. Accepting a fragile, brittle, or unreliable development environment represents a category of risk on par with that of production environments. A software factory that cannot instantly respond with a push to production in the minutes after the enemy exploits a zero-day cyber threat during wartime is the digital equivalent of a forward-deployed aircraft carrier with an unreliable catapult system. We do not separate the catapult from the carrier, and we should not separate software development from production.

Despite the highly publicized impact of software supply chain attacks and the need to invest more in development of ecosystems, all areas of the DoD still lag in advancing the posture and status of the venerable software development environment. In fact, over a decade has passed since the DoD substantially updated rules that prohibit entire categories of software from connecting to production networks like NIPRNet. The unwillingness and inability to allow a tool like git onto secure networks is absurd—it is not a compiler and cannot be used in isolation to field new unauthorized executable applications onto a network like the NIPRNet, because it is a development tool. It firmly demonstrates either a complete lack of understanding of
modern digital logistics or a profound stubbornness to admit that the policies of the past conflict with the logistical need to fight and win tomorrow’s conflict.

As far as tooling goes, git is the bedrock of all modern software development environments. Service Organization Controls (SOC) audits require significant evidence that the organization controls who has access to source code or can perform a deployment into production, that the code cannot be blindly merged into a main branch without a review by multiple eyes, and that older versions are easy to access. Acceptance of the reality that software defines tactics is a transitive acceptance of the idea that development is production and that git is the critical component. Stated in more unequivocal terms, git is the Microsoft Word of the modern software-defined world. It should not be a tool that only software engineers can use, either, in an era when the speed of the observe–orient–decide–act (OODA) loop matters more than ever. Let’s explore this concept further to understand how the tooling of the modern software factory can help organizations operate at a higher velocity.

**Documentation as Code**

Using git, it is possible to easily illustrate how modern software factory tooling could change the way the DoD goes about consensus building across its vast organization. To craft this illustration, we first have to introduce Washington Headquarters Service (WHS), the entity that is the custodian of the Correspondence and Task Management System (CATMS).

The WHS describes the CATMS as “a flexible and scalable system that allows component users to create, delegate, assign, respond, and research a knowledge repository linked with task and correspondence accurately and efficiently.” In simpler language, the CATMS “coordinates” memos or documents across the military departments in a very specific manner. Coordination starts by circulating the memo to the affected GS15/O-6 crowd, who must adjudicate all comments and feedback by a given suspense date that is typically either 14 or 30 days out. Rarely do all entities redline the document and respond by the suspense date, so extensions are common.

There are substantive and critical comments, the latter of which are strenuous objections. Per the CATMS FAQs, “All comments submitted in coordination must be adjudicated,” and that occurs with a DD Form 818, the Consolidated DoD Issuance Comment Matrix—an Excel spreadsheet. For example, suppose the author chose the word resilient and a component feels that uncompromising is a more compelling word. The system captures the rationale to accept or reject the feedback. Critical non-concurs usually require formal meetings and a lot of compromise to resolve—politics enter here. Too many substantive comments can also justify a non-concur.

After the relevant departments complete this round of edits, the memo is republished for general officer / flag officer (GOFO) review. These one- and two-star military and civilian equivalents now have their opportunity to redline the document. The parties issue a suspense date, grant extensions, and adjudicate comments.

Next up is principal staff coordination, involving the under- and assistant secretary levels of the bureaucracy. Again, the parties give the document a suspense date, extension, and adjudication as necessary. Finally, they prepare a signature package for the deputy secretary of defense. They long ago socialized the intent of the memo, and now they review and may ask why previous reviewers did not consult Component X or selected a particular word or phrase—and they may ask for more coordination. Once they sign, everyone already knows what they have signed because it represents the consensus of the organization and it was fully coordinated.

Now, let us reimagine this process using git. Instead of relying on a word processor and spreadsheet that perpetuate duplicative data entry problems during adjudication, the process pivots to adopt markdown. Markdown is easy to learn and ubiquitous
across the internet, and an abundance of tooling exists to accept it as input and apply style sheets to render it in visually compelling ways.

In the git repository, developers create a new project representing the memo, create a branch called first_draft, and then draft the memo. They issue a merge request and invite in all of the reviews, requiring everyone to openly share feedback with everyone else. This eliminates the unintentional secrecy that too often occurs during coordination because components cannot instantly and immediately see feedback from others in real time. Components that have a substantially different vision can branch off first_draft, make their edits, and issue their own merge request. Anyone can advance future iterations of the policy or memo merely by creating a branch, making the changes, and issuing a merge request. Modern git tooling supports what is known as signed commits, adopting digital signatures to ensure non-repudiation—confirming the author really is who they say they are, in essence. The collaboration is transparent and instant, and there is no need to reenter modified sentences or formally find times for meetings because the git history contains everything.

It is unrealistic to believe that WHS will abandon CATMS anytime soon, but PEOs should explore this technique of documentation as code because it inculcates the software factory's importance throughout the program and for the life of the program. Envision a scenario in which it becomes impossible to drive the necessary paperwork of the program without a functioning software factory. Documentation as code is one of many illustrative techniques that could help a PEO focus on an outcome in which the lifelines of the program and software factory become indistinguishable.

**Seeking Good and Amplifying It**

As an institution, the DoD is not known for its marketing prowess. Highly valuable communities of practice, like the DevSecOps Community of Practice that meets virtually every month, represent some of the best forms of grassroots collaboration across the military departments and the DoD's fourth estate. Unlike commercial enterprises, the DoD allocates no marketing budget to raise awareness of these valuable events. By and large, word of mouth is as good as it gets, and for an organization the size of the DoD, voices only carry so far. Discerning what works and, perhaps more importantly, sharing and learning from what did not work are too often ad hoc activities. The modern software factory is not a static rendezvous point identified by precise Global Positioning System coordinates. Identifying what good looks like remains an open problem for the DoD.

The DOD CIO Library offers a wealth of consensus-driven reference designs that capture what good looks like. The DevSecOps Guides and Playbooks offer strategic-level insights, while multiple approved reference designs provide tactical implementation guidance. The existence of multiple consensus-driven, peer-reviewed reference designs is a sign that leadership is beginning to recognize and explore the value of diversity of form, the heterogeneity of software.

**More Work Remains**

Embedded systems and hardware-in-the-loop (HIL) capabilities lag behind their cloud-centric reference designs. The Cloud Native Access Point (CNAP) is one of the DoD's best-kept secrets born from the Air Force's early leadership in embracing modern software development. Both government and industry should analyze this reference design through the lens of software heterogeneity. It is vendor agnostic and demonstrates a level of elasticity and adaptability that should make it a default choice for every new program the DoD launches.

Undoubtedly the Pentagon will remain a consensus-driven organization. Suggestions that the organization somehow behave like an autocratic software start-up will not come to fruition and are arguably counterproductive. However, PEOs cannot and should not invoke consensus as an excuse for lethargy in
a world where software moves at a frenetic pace. Every program will have the proverbial aha! moment in realizing its own modern software factory. Every program is equally at risk of wasting time, money, and energy reinventing the wheel. Modern software factories have to avoid falling under the immense weight of a snowflake: “Our program is different, so we will start from scratch.” No more disastrous view exists in a world where open-source software and application programming interfaces drive progress. There is no defensible reason for not recognizing what good looks like and then extending beyond it to capture the unique needs of a program.

This is where DODI 5000.87, the Software Acquisition Pathway, should rise to confront the challenges of establishing modern software factories. The DoD should incentivize PEOs to attain membership in the 5000.87 club. DODI 5000.87 offers a stable platform for institutionalizing the Two Factors Principle, requiring PEOs to seek out a model and invest in the roughly three dozen software factories that exist across the DoD. It can force PEOs to pause and reflect on their program’s distinct strategy for the natural ebb and flow of feature work and sustainment. DODI 5000.87 is unequivocally the foundation for applied software acquisition in a world where software defines tactics.
Acquisition Competency Targets for Software (ACTS)

The soundbite “software is eating the world” entered the lexicon with Marc Andreessen’s 2011 Wall Street Journal op-ed. But the most salient point from the article is often lost in routine conversation. Over 10 years later, the world’s largest book-seller (Amazon), the world’s largest video service (Netflix), the world’s largest taxi company (Uber), and the world’s largest hotel chain (Airbnb) are all still software companies, continuing to invest in their software product development although their businesses manifest offerings in traditional business sectors. Andreessen’s statement that “national defense is increasingly software-based” was not a strong endorsement of software’s impact. Fast-forward 10 years, and this report posits that reframing the discussion around national defense is reasonable: Adaptability and resilience define strategy, and software defines tactics of national security more than bent metal.

Even though physical sciences and bent metal keep the F-35 in the air, software and digital systems give it a combat edge over adversary systems.

PEOs need to learn how to skilfully think about and guide the development of their programs as though they are a software company seeking to deliver a weapons system, similar to how executives at Amazon, Netflix, Airbnb, and Uber have harnessed software to deliver market advantage in what were considered traditional industries. Even Congress, not always regarded as the most forward-leaning body, reinforced this shift in the draft NDAA for fiscal year 2023, which sought to create an environment that “enhances incentives for acquisition professionals [to] learn more about the business models of software-first commercial, start-up, and nontraditional companies that may be able to offer solutions to the Department.”

What follows is a concrete set of acquisition competency targets for software, or ACTS, that PEOs and program managers can use to manage development like a software company, even in the heavily regulated environment of defense procurement.

Photo Caption: An employee of the company EnterVR uses the training simulator VR Shield by the company on the first day of the ITEC fair for military and weapons technology in Stuttgart, Germany, on May 15, 2018.
It is important to note that this report does not suggest that the DoD run the military as a commercial software company. The needs of the DoD are far too diverse to roll up to simple metrics like profit, and its equities are too varied to house in a single monolithic code repository. Beyond this, the tremendous responsibility of preparing for conflict does not become lighter with the introduction of beanbag chairs and hoodies. An acquisition professional bears moral responsibility as well as substantial statutory obligations.

But just as the relative combat advantage is moving to sources in software, the relative emphasis in acquisition has to shift to software delivery.

We have designed these ACTS around the key ideas that PEOs and program managers should realize, and should institutionalize, to guide their acquisitions to embrace the three goals of this report:

- **Embrace the heterogeneity of software.**
- **Recognize that software, not legacy warfighting platforms, controls the speed and efficacy of modern kill chains.**
- **Preserve the balance across the digital triad of software, data, and AI/ML as equal peers.**

Our premise is that when PEOs actively implement these ACTS, they better position their programs to produce weapon systems capable of not just operating downrange, but adapting at the speed of relevance when software defines tactics.

Each of these ACTS could stand on their own, but the sum is greater than the parts. We intentionally discuss and present them as principles to avoid making them overly prescriptive. PEOs should view, interpret, and define the ACTS collectively for each individual weapons program because of the inherent heterogeneity of software that we have emphasized in this report. They can realize the ACTS only by recognizing the value of cross-functional teams at every level and every stage; a program cannot extract full value from its ACTS if its supply chains, industry partners, authorizing officials, and warfighters are not part of the journey. Finally, PEOs have to periodically review a program’s implementation of the ACTS through the lens of continuous improvement.

**ACTS 1: Evaluate Existing Software Factories before Building Your Own**

Launching a brand-new software factory within the DoD is an exhausting and expensive endeavor. Frustratingly, effort invested in such a launch does not directly contribute to progress in the acquisition outcomes or warfighting capability that most acquisition professionals are accountable to provide. Stated differently, just because the software factory achieves full operational capacity does not mean it releases production software that provides value to the warfighter. Thus, both to accelerate delivery of capability and to stretch program funding, the first step in a software acquisition development should be to explore whether existing software factories can meet program needs. The good news is that substantial growth has occurred in every military department, and the likelihood of finding a suitable fit is high.

With understanding and knowledge of the continuing nature of software development and sustainment, PEOs need to grow to recognize and anticipate a natural ebb and flow of feature development, sustainment, and more feature development. The PEO–software factory relationship has to be designed to ensure that factory lights stay on during periods when a program requires only routine patching and minor upgrades, and yet the factory should be capable of rapidly scaling up when programs require new features urgently.

The scalable staff and specialized skill sets that exist across the breadth of the industrial base are better suited to deliver complex features, algorithms, and functionality than are government or military personnel. Yet routine patching and sustainment operations within an established software factory ecosystem do not require cutting-edge software engineering talent. Existing
software factories can support PEOs during periods of sustain-
ment while providing access to industrial base talent when they
need to pivot toward innovative feature development.

The premise of this ACTS is that those PEOs who lack an ac-
tionable plan for realizing a long-term software factory relation-
ship will operate at a program at a much higher risk of not realiz-
ing software-defined tactics compared to other PEOs that have
achieved this objective.

Corollary: Rationalize the Staffing Ratios
between Government Civilians and
Contractors

The defense prime contractors have an immense talent pool at
their disposal: the top five employ 618,000 people, including
tens of thousands of software engineers, many of whom are
highly talented and experienced. However, there are a number
of reasons these top primes are at a substantial disadvantage
in an era when software defines tactics. First, these organiza-
tions have significant investments in highly specialized physical
science research laboratories, and they fund their physical in-
fraction through either internal research and development (IRAD)
or direct capital expense offsets. They must amortize
these investments across multiple fiscal years, which affects
their cost models.

Second, the large primes more often groom future leaders from
within their own ranks, which makes them less likely to have ex-
cutives familiar with how software development has changed
over the past decade. Indeed, it is often difficult to find leaders in
a defense prime who recognize that systems engineering is not
synonymous with the MIL-STD-499 V model and that Agile, for
example, is another disciplined systems engineering approach
more suited to software systems. Habits and prior successful
experience can make it difficult to see the forest for the trees.

These challenges, combined with the firms’ inability to carry staff
on a bench, with employees waiting for clearances or a new pro-
gram to work on, make it extremely difficult for contractors to en-
tice software development talent that has lived and breathed an
alternative set of development practices—for example, former de-
developers from Google, Apple, Amazon, Microsoft, or Meta. Work
habits are also different: prime contractors have compensation
and promotion strategies that reward loyalty, which often clash-
es with the two- or three-year employment rotation that is typical
in modern Silicon Valley. The few who are willing to occasionally
cross the tool and methodology chasm between the prime con-
tractors of the defense industrial base and Silicon Valley do so
knowing that they will have to fight a software development equiv-
alaney of septic shock upon arrival. As a result, modern software
practices may take a long time to percolate into prime culture.

In contrast, smaller and nontraditional businesses across the
defense industrial base have an early lead in providing software
staff to operate government-led software factories because
they neither carry the overhead nor have the lethargic pro-
cesses that come with major primes. Smaller businesses also
typically avoid the institutional investors that demand dividends
from the senior leadership team, giving the smaller competitor
more freedom to navigate job titles and compensation pack-
ages. Smaller businesses were often born digital and can give
their software practitioners modern computing tools without the
lethargic bureaucracy typical of large legacy prime contractors.

However, those same small businesses also have more diffi-
culty forming a direct relationship with a program office, as any
source selection must weigh the risks of placing a smaller com-
pany in a critical role against expected savings in cost and time.
Small businesses are more nimble and able to align their oper-
tional models with the so-called software body shops than are
most major primes. Both roles are necessary parts of a weap-
ons system program, and establishing transparency around this
fact mitigates alienating one in favor of the other.

It is often best for the program manager to take a compositional
approach to their acquisition strategy: dividing development
into chunks appropriate for different classes of industrial base performers and using the appropriate acquisition pathway for each chunk. These components of an acquisition may surge or recede as program needs change. This approach leaves some tasks—like establishing, running, or sourcing a software factory—to the program manager and appropriate partners.

**ACTS 2: Partner with an Authorizing Official to Realize a Continuous Authorization to Operate**

The previous chapter discussed the development environment as a production environment. Operating a software factory is the precursor to modern software development, and the factory cannot begin operating until the system where it operates has received an authorization to operate (ATO). New programs need to recognize this dependency early and consciously build a partnership with their authorizing official (AO). Why? There are too many AOs who are simply unsure what “shifting cybersecurity left into the CI/CD pipeline” means, what it accomplishes, or why they should support these activities because they (the AOs) have not been involved in modern software development techniques or tooling (see MOCAS conversation in the previous chapter).

Program managers can benefit from establishing direct knowledge of the most recent DoD guidance on the realization of continuous ATOs (cATOs). In 2022, the DoD’s senior information security officer (SISO), David McKeown, released a highly relevant memo that outlines the minimum requirements for obtaining a cATO. This memo establishes continuous monitoring, active cyber defenses, and a secure software supply chain as so critical that they represent the minimum requirements to even consider operating a system under a cATO.

In particular, and building on the salient points defined in the previous chapter, the SISO memo establishes that “a system must embrace the DoD Enterprise DevSecOps Strategy, aligning to an approved DevSecOps Reference Design.” The memo essentially codifies the relationship between ACTS 1 and ACTS 2; a program that builds out a modern software factory is taking the requisite steps to pursue a cATO, or, if it intends to adopt an existing software factory, the program could benefit from its existing cATO.

Why the emphasis on a cATO over an ATO? The value to the PEO of operating under a cATO directly relates to the ebb and flow of software factories outlined in ACTS 1. Even during periods when the program is not aggressively pursuing new feature developments, operating under the cATO provides cybersecurity resiliency beyond the traditional ATO process. It accomplishes this through the combination of three mandatory elements outlined in the memo. Given the dynamism of advanced persistent threats, zero days, and the need to patch at the speed of relevance, the cATO provides the PEO with the greatest possible assurance. First, the software factory will be capable of performing critical software updates in a time of need without waiting on paperwork or paper processes that offer marginal defenses when the Cybersecurity and Infrastructure Security Agency (CISA) identifies a new key exploited vulnerability (KEV) that requires risk mitigation through concrete actions.

Second, active cyber defense goes beyond merely defending the software’s binary runtime with, including the software supply, the totality of intelligence available across the operational parameters of the application itself.

**Corollary: Establish an SBOM Consumption Plan**

In early September 2022, the Pentagon temporarily halted deliveries of F-35 fighters when it discovered that manufacturers had sourced raw materials from China to produce a magnet. The modern software supply chain is even more complex, and the impact and role of open-source software across the supply chain are virtually inescapable. It is unreasonable to expect the DoD to develop all software systems from scratch without accepting either direct or transitive dependencies on open-source software. The ubiquitous Linux operating system depends fully
on open-source software libraries, and even firmware found in chips can include open-source software—all of which foreign nationals subject to their countries’ draconian laws might have influenced, designed, or perhaps implemented. Instead of fearing open source, treat it like any other program risk and make risk-informed decisions by using a proper risk frame.

For software supply chains, the focal point is the software bill of materials (SBOM). Asking for and taking possession of an SBOM are insufficient measures in and of themselves to mitigate any risk. First, program managers need to be educated and understand the criticality of direct and indirect (transitive) dependencies. Different development languages use different files, notations, and mechanisms to capture all dependencies. In JavaScript, there may be a package.json file in a root directory, while in Go a go.mod and a go.sum file are present in the root directory. Other languages have their own unique ways of managing dependencies.

Knowing how to interpret SBOM data is vital for proper program risk management. For example, in our experience, it is common to build a software application that may define two dozen direct dependencies only to have the SBOM explode to literally contain hundreds of indirect dependencies that show up in the SBOM. First, not every indirect (transitive) dependency can affect the runtime behavior of the application. Developers use some of those dependencies to establish and track minimum versions of libraries in use. They may also use some of those dependencies as code generators during the prebuild phase of the application.

In the popular Go programming language, for example, a developer may issue a go generate command that reads in database table definitions and a set of database queries and rely on a library that dynamically creates source code to map the database and those queries. The library itself is a direct dependency (you cannot prebuild the project without it), but it is not present and cannot directly affect the runtime environment beyond the code that it generated. Unfortunately, PEOs cannot dismiss these nuances and must have access to technical experts who can decipher and discern this type of information from an SBOM.

In larger applications, it is neither unreasonable nor unexpected for a software team to hand over an SBOM that contains over 1,000 libraries. Yes, one person may build and support some of those libraries in their spare time.

Yes, some of those libraries will have open common vulnerabilities and exposures glossaries (CVEs) across the full spectrum (low, medium, high, and critical). And here’s the challenging part: not even the software team that provided the SBOM will be able to fully explain how every one of those indirect (transitive) dependencies affects the software application without painstaking and time-consuming research.

A practical conclusion from this corollary is that it is useful to access DoD software supply chain threat intelligence in order to cross-reference and risk-mitigate the well-known threats. Statistical analysis can compare a codebase to that of other programs to explore, understand, and risk-mitigate one program in the PEO that uses a software library that no other program in the PEO—or, for that matter, in the military service—is using. Analysis should include a careful evaluation and risk-mitigation of the SBOM’s outliers using the data in the SBOM to generate an open-source software nutrition label.60

We intend for ACTS 2 and this corollary to ensure that a PEO is making risk-based decisions and avoiding application of meaningless heuristics like equating the number of active contributors to an open-source library to a level of supply chain security. With one contributor or a thousand, the risks are the same, and popularity of software libraries, statistical analysis, and an effective CI/CD pipeline that reduces the mean time to recovery (MTTR) metric to minutes instead of hours or weeks will provide a high degree of cyber resiliency that both the PEO and the AO can rely upon.
ACTS 3: Own Your APIs, Even More So Than the Implementation

The previous chapters outlined the overwhelming need for adaptability in complex, heterogeneous software systems. Interfaces are an essential element of managing the creation, destruction, and modification that are parts of this adaptability. They are the connections between the elements of capability, and especially between executable code often running in distinct environments. In the software-first model, application programming interfaces (APIs) become the foundation on which everything else is built.

Because interfaces feature so prominently, it is important to understand the tools available to an acquisition professional. The term software appears in the Federal Acquisition Regulation (FAR) 570 times. Upon closer inspection, virtually every occurrence of the term is phrased like this: “computer software and computer software documentation.” Further exploration reveals that the acronym API occurs across the FAR and Defense Federal Acquisition Regulation Supplement (DFARS) a grand total of four times. Strictly speaking, neither the FAR nor the DFARS defines regulations that preclude PEOs from establishing APIs as the nucleus of every weapon system acquisition. APIs live in the whitespaces of the regulatory landscape.

However, Section 804 of the FY21 NDAA recognized the central role of interfaces in development, giving PEOs statutory authority to both focus on APIs and ensure that they are discoverable via machine-readable definitions. Indeed section 804(a)(2)(B) calls for the collection of interfaces, namely the following:

1. software-defined interface syntax and properties, specifically governing how values are validly passed and received between major subsystems and components, in machine-readable format;
2. a machine-readable definition of the relationship between the delivered interface and existing common standards or interfaces available in the interface repositories established pursuant to subsection (c); and
3. documentation with functional descriptions of software-defined interfaces, conveying semantic meaning of interface elements, such as the function of a given interface field.\(^{61}\)

The law further mandates that these requirements “shall apply to any program office responsible for the prototyping, acquisition, or sustainment of a new or existing weapon system.” This is a powerful tool to assert control over acquisition destiny.

The DoD cannot afford to underinvest in API design. The intellectual property rights to the left or right of an API are negotiable, but the DoD has to control, inventory, and publish the APIs, which has to therefore come with no less than government purpose rights (GPR). If a program owns the API and has GPR to invoke the API as often as they like, then the program’s data is accessible without requiring the prime to deliver data via CD-ROMs using first-class postal service delivery with additional cost and unacceptable latency.

Let’s explore this point further. Design patterns have existed for over three decades now, and schools train every software architect to program to an interface, not an implementation. If the program wants record-level data access, define an API that supports query access to a single record. If the program wants set-level data access, define an API that returns a record set. If the program wants access to the entire data set, define an API that supports downloading a compressed file of all data in a canonized format like JavaScript Object Notation (JSON). Proper API design and negotiation of invocation rights at the API level mitigate the risk involved when the DoD does not own an implementation.

When the DoD owns the API, a program office can rapidly swap out a physical implementation for a virtual or game simulation implementation. When the program office can do so properly, it’s as easy as updating an environment variable, configuration
flag, or feature flag. This also means the DoD cannot accept anything less than GPR on the public interfaces that matter, but what is on either side of that interface could be proprietary or locked down with more restricted rights. Thankfully, the FY21 NDAA reinforced this rights position statutorily and allowed the ability to retroactively assert interface rights in updated 10 CFR 2320, which now states that “the United States may release technical data . . . pertaining to an interface” for purposes of integration.

The focus on the API means the DoD needs to force the PEO to evaluate and identify system boundaries and modularity at the start of a new program’s journey. Interface design cannot be an afterthought; it is the forethought that determines the program’s ability to act as a force multiplier for programs that leaders have not even yet envisioned. However, the government does not need to use every internal API (and thus does not require GPR for source code), and therein lies the importance of system boundaries and a comprehensive understanding of separation of concerns between disparate software components. As an example, the need to control the API for managing the process of sending a document to a printer is unlikely to be as relevant as controlling the API that provides advanced searching, filtering, and querying across a highly relevant program data set.

On teams responding to the RFP, software engineers are likely to reply with a head shake and an eye roll when an RFP demands that “all software APIs must be delivered with GPR.” Instead, invest the time to identify the system boundaries, understand where the program benefits from establishing a separation of concerns via APIs, and demand unlimited rights or GPR on those things only. This is an exercise in intelligence that only the program, not the contractor, can perform. In our vignette that illustrates all of these ACTS in a more concrete fashion, we will explicitly demonstrate this concept for additional clarity.

An API is not a Word or PDF document describing a data exchange. Begin to view an API as an interactive boundary that establishes a separation of concern between critical system modules. The API manifests its power only when it is physically exercised and made available in a machine-readable format. However, this a dangerous area because PEOs and program managers have to discern the nuances of different types of APIs and understand how to define, document, and publish them. The highly popular OpenAPI is wonderful for representational state transfer (REST) software architectures. However, this standard is at best incapable and at worst wholly inapplicable for describing service-oriented architectures, interfaces for working field programmable gate arrays (FPGAs), or graphics processing units (GPUs, e.g., CUDA®, a parallel computing platform and programming model developed by NVIDIA), among others. RFPs cannot adequately capture the APIs of a system-of-systems merely by mandating documentation of all interfaces using OpenAPI.

Progressive PEOs have the ability to reuse and expand on existing guidance to create their own code inventory across all of their programs. Consider leveraging the existing DOD consensus standard metadata schema as your starting point. Our recommendation is to add a single new stanza that contains a uniform resource name (URN) identifying a set of APIs, an enumeration that captures the type of API (e.g., REST, SOAP, CORBA, GRAPHQL, etc.), and two uniform resource locators (URLs) that point to the human-readable and machine-readable documentation respectively.

Additionally, ensure you use cloud-native principles to build the system in which these APIs operate. Contrary to popular belief, a cloud-native architecture actually has little to do with the cloud; more accurately, it captures how an application relies on horizontal scaling to elastically respond to spikes in demand. If vertical scaling software is about adding more memory and more CPUs to a single machine, horizontal scaling and cloud native architectures implement elasticity by adding more low-
cost machines horizontally. The DoD has already established through a very public demonstration in 2020 that it can deploy cloud-native in embedded systems just as easily it can in a hyperscaler cloud service provider ecosystem.64

Corollary: Ensure the APIs Provide Access to All the Data the Program Cares About

Carefully consider future program data aspirations and design APIs that account for not only the primary data sets but the tributaries that flow underneath the surface of the application. Many DoD programs failed to recognize the inherent value of an application’s data streams and, under the guise of a vendor’s “intellectual property” stamp, lost access to their own data.

We intentionally use the plural streams in this corollary, and it’s worth further explanation using a water analogy. The Mississippi River comprises flows from the Arkansas, Illinois, Missouri, Ohio, and Red Rivers. The Tennessee, Allegheny, Scioto, and other rivers feed the Ohio River. When you visit the Mississippi River Delta, it’s easy to inadvertently forget about the headwaters and tributaries. If the US suddenly sold the water rights for each of the tributaries, what value would there be in owning the rights to the waters in the Mississippi itself?

By extension, it is easy to focus on the major application data sets, analogous to the mighty Mississippi. There are other headwaters and tributaries that PEOs cannot overlook, including software supply chain data in the form of an SBOM and highly relevant cyber resiliency data sets produced within the CI/CD pipeline, especially the pipelines measurement for mean time to recovery (MTTR), runtime telemetry data that represents how users navigate and interact with the software, and other examples.

PEOs can find greater value and resiliency by pivoting away from defining specific data sets that must be delivered via CD-ROM media and instead focus on an adaptable set of APIs that contracts obligate implementers to implement. Rigid data set definitions could preclude innovation and growth, whereas a well-defined API accessible with GPR not only ensures data accessibility for data science activities but also serves as a proven mitigation strategy against the dreaded proprietary data claims of a defense contractor.

If PEOs can employ APIs to intrinsically assure the government data rights it needs, that does not mean the data will be easily accessible. The final step is to ensure a robust set of APIs that have the capability to fetch a singular record set, sorting and filtering capabilities to reduce noise when seeking a reduced data set across the complete data catalog, robust pagination capabilities to avoid asking for a 2-GB data set in a single request, and perhaps the ability to make long-running data requests where the system assembles data over minutes or hours and makes a programmatic callback to inform the system making the request that the data set is now available for download at this location.

ACTS 4: Behave Like a Software Recruiter

Software Is Never Done established that digital talent matters. Digital talent matters so much that PEOs need to recognize that they are fighting for the same software talent as Google, Apple, Amazon, and the small software start-up down the street. One of the overlooked successes of Kessel Run was the way in which the program behaved like a software recruiter. The way in which it engaged the community, both in and around Boston and virtually at sites like LinkedIn, showed a program engaging in what we could describe as asymmetric software recruiting.

Leadership at Kessel Run recognized the futility of recruiting staff using government websites and intentionally turned to social media to create demand. They advertised in niche locations, like Black Girls Who Code. They successfully portrayed Kessel Run as an exclusive club, something that people should seek to be a part of even if it meant abandoning an existing software start-up to take a government position with a reduced compensation package.
Their approach was a resounding success that is likely far beyond what the Defense Innovation Board even imagined. After their first six-week campaign, the program had amassed over 1,400 applications for its new roles, seven times more than their three previous traditional hiring campaigns.65

The DoD finds itself in a place where the military services are failing to meet their recruiting goals.66 People generally do not view civilian jobs in the government as exciting positions, and grassroots viral campaigns like Fix Our Computers further advance the notion that working for the government is slow, dull, and painful.67

Every PEO needs to actively seek out ways to counter these challenges, studying how Kessel Run and other successful programs attract digital talent. They need to generate ideas, use professional networks to share both what is working and what is not working, and explore other recruiting playbooks.68

Corollary: Shrink the Distance between the Engineer and the Warfighter

Programmatic vision is powerful, but society recognizes and rewards execution more often than vision. Execution is not possible without the right set of digital talent. A PEO can differentiate their programs from competing commercial opportunities drawing candidates from the same digital talent pool by shrinking the distance between the engineer and the warfighter. How many software engineers have been within feet of a military aircraft, tank, or ship? How many software engineers can proudly raise their head and declare, “The code I wrote yesterday saved the life of an American service member today and ensured their safe return to their kids tomorrow”?

Devise plans and offer to rotate software engineers to remote military bases to ensure that they have access to the user community—the pilots, tankers, sailors, field medics, special forces, and other roles unique to the military service. Consciously embark on a program that helps engineers realize the importance of their work in DoD programs is unlike anything else they will find in competing private sector positions. Shrinking the distance between the engineer and the warfighter is a concrete mechanism for amplifying passions that already exists in the digital workforce.

ACTS 5: Think in Service Levels

Accountants think in terms of spreadsheets, graphic artists in terms of visual images, and project managers in terms of the venerable Gantt chart. The Gantt chart is excellent for assembling a list of tasks, dependencies, and milestones and visualizing progress to date. This project management tool has proven indispensable for coordinating industrial activities across a production floor at a given time—conduct physical activities to ensure a resource is available at this location not later than this milestone. In the industrial setting, the hard part is not connecting the pieces, it is getting all the pieces to arrive on the factory floor at just the right moment.

The very essence of software, and in particular DevOps, is the opposite. Generally speaking, highly advanced commercial software is available on demand for access, from state-of-the-art enterprise resource planning (ERP) to product life cycle management (PLM) software and from learning management systems (LMS) to human resource management systems (HRMS). The hard part is not getting all the pieces to arrive by a milestone with software, it is connecting them to interoperate reliably and at speed. The data integration market is growing at a compound annual growth rate (CAGR) of nearly 12 percent, and experts expect it to achieve a market share in excess of $19 billion within the next four years.69

Software development tools, programming languages, codified design patterns, and the advent of programming with Google (in which a developer uses Google to find a snippet of code that meets their immediate needs) have made software development easier than it historically has been. It is the integration between different architectures and API styles and dozens of
identical programming standards (as depicted in an infamous XKCD cartoon70) that creates delivery delays and affects the reliability and operational speed of a system. Software integrations are not analogous to twentieth-century industrial integrations, and the Gantt chart has proven to be an unreliable management technique for software integration activities.

We can simplify software integration to the concept that data is in motion. Data in motion is transiting across distinct and easily identifiable system boundaries. It should be obvious that modern standards judge any system that is unreliable or unavailable as a failure. As reliability and speed demands on the data in motion increase, so too does the operational cost.

Organizations measure and capture reliability and speed using service levels. For example, Google defines precise numerical targets for system availability using a service level objective (SLO), enabling developers to frame design or architectural discussions around quantitative targets that they must meet or exceed.71 Where the SLO is developer-centric, the service level agreement (SLA) is typically a legal instrument that establishes thresholds and penalties, typically financial penalties, when systems do not meet performance targets. Logically, the SLA might establish a set of metrics that must be at least 99.9 percent, while internally the development team might be operating against a SLO that defines the same set of metrics at 99.95 percent. Finally, the service level indicator (SLI) is the discrete measurement of successful probes against the set of metrics the team uses to calculate the actual service availability percentage.

If it is possible to cleanly express both reliability and speed and if established vendors can demonstrate SLIs that meet or exceed those metrics, the PEO’s first instinct should be to outsource that capability to a commercial or dual-use provider. The PEO’s first instinct should be to outsource that capability to a commercial or dual-use provider. The PEO’s first instinct should be to outsource that capability to a commercial or dual-use provider. The PEO should focus on building novel solutions for the undefined requirements that cannot readily define reliability and speed or that a provider cannot demonstrably meet. When the act of production is difficult, PEOs should want diamonds, and when the speed and reliability of data in motion are difficult, they should want service levels.

**Corollary: The Lollipop Is More Important Than the Enterprise Architecture**

The DoD operates an exceptional logistics network capable of moving people and materials around the planet at extraordinary speeds. It is possible to easily define an SLA for routine envelope, package, and pallet delivery to any US military base around the world. Outsource that. Is it possible to define an SLA for the insertion of special forces in an austere environment while under hostile fire? No. There are too many variables, yet this is precisely the type of activity that the DoD must accomplish with precision and reliability. The government, and in particular the DoD, explicitly owns force generation.

Purists will argue the only thing that matters is the enterprise architecture (EA): if the program gets the EA right, everything else will just work. We argue that this is a naive position because the modern world is built from systems-of-systems, and it is the integration between disparate software systems that drives economies and creates disproportionate battlefield advantage against peer competitors.

There is value in adopting enterprise architecture activities within a program, but PEOs and program managers need to accept the inherent brittleness when hundreds, if not thousands, of discrete systems come together to achieve force projection downrange that realizes a kill chain. Who can identify a singular set of DoD Architecture Framework (DoDAF) models that capture the unified EA for the system-of-systems required to insert that special forces team?

The “lollipop” is a colloquialism that software developers use within the unified modeling language (UML) to represent a software interface.72 Measuring the reliability and speed of data across an interface is common practice. How do you measure reliability and speed of data using a DoDAF model on a printed sheet of paper?
When PEOs actively think in service levels, it marginalizes the need to have difficult conversations about the right intellectual property posture or contract structure. It avoids believing in paper models that seemingly project 100 percent reliability. The EA of Amazon’s, Microsoft’s, and Google’s cloud is likely very impressive, but let’s not forget recent headlines:

- “Google Cloud Outage Takes Major Websites and Apps Down”
- “Extended AWS Outage Disrupts Services across the Globe”
- “Azure’s DNS Problem Highlights Fallibility in Cloud Services”

When commercial tech giants like Google, Amazon, and Microsoft cannot achieve 100 percent reliability and speed of data in motion, PEOs cannot accept the position that if they get the EA right, everything else will just work. Never accept the idea that a model is more powerful than a functioning interface and avoid the acquisition original sin of predilection for prediction. Focusing on the lollipop inherently drives outcome-based thinking. Software integration outcomes occur only through interfaces, and the PEO can measure, monitor, and, in extreme cases, rapidly pivot interfaces to another service provider that implements the exact same interface.

**ACTS 6: Define Zero-Trust Outcomes within Software Applications**

Over the last decade, a great shift in cybersecurity strategy took place as organizations recognized that the so-called castle-and-moat approach was no longer viable. Today, there is widespread recognition that every network is porous no matter how wide or deep the proverbial moat. Even air-gapped networks, seemingly impervious to attack because they are not connected to the internet, have proven vulnerable. In 2014, Wired ran a story entitled “An Unprecedented Look at Stuxnet, the World’s First Digital Weapon” detailing how an air-gapped network includes hardware and software that engineers at some point designed, developed, compiled, and deployed from artifacts that must transit the air gap moat. The modern view is no one should ever trust hardware, user, network traffic, data, etc. implicitly. In what is arguably an unfortunate misnomer, the term zero trust (ZT) captures the modern cybersecurity strategy but is more about avoiding implicit trust than advocating for zero trust as an absolute. After all, at some point organizations must trust the computer, user, and software enough, or no digital work would be possible.

What has been missing from the ZT conversation is widespread education and conversation about how PEOs have to recognize that ZT is a fundamental component of every software activity. From CISA to the DoD, publications cover maturity models to reference architectures, all of which are beyond the scope of this report. These documents are valuable, but none of them explicitly elucidates the role of the PEO in realizing software that natively exhibits ZT principles. This needs to change.

In its essence, ZT introduces a policy decision point (PDP) that defines operational policies and captures them outside of the compiled source code. Chapter 2 established that once developers have compiled, deployed, and disconnected code from its build chain, it becomes rigid and resistant to change. PDPs are valuable because the cybersecurity policies are detached from the compiled binary and stored as external files that are explicitly intended to be updated weekly, daily, or even faster when defined inside of an OODA loop where software defines tactics. Defining policies like this inherently creates cyber resiliency when paired with an enforcement engine, a policy enforcement point (PEP) that makes an access determination. It is here that the notion of zero trust turns from a linguistic absolute into a practical assertion that avoids implicit trust.

ZT and its PDP/PEP combination offer PEOs an unprecedented opportunity to build cyber resiliency in ways that previous military platforms (ships, tanks, planes, etc.) have not capitalized on. Instead of operating against a binary decision format, per-
mit or deny, ZT introduces a third option known as deny with countermeasures (see figure 13). Forward-looking PEOs have started to realize that ZT’s pivot away from a binary and toward a ternary decision format represents a cybersecurity revolution in military affairs. Let’s explore this in more depth.

As an example, consider sensitive acquisition data that the DoD manages through a typical web application user interface. Historically, either the user had access to the data, or they did not. The website might require a government-issued laptop, engagement of a VPN, multi-factor authentication for the user, etc. But the decision was always binary: the user either could or could not access the data.

The incorporation of a ZT ternary decision format conceptually creates an unlimited set of outcomes based on combinations of metadata: person/non-person entity, device state, data sensitivity levels, network perimeter, multi-factor authentication, etc. If a user is on a government-owned laptop that is up to date with all software patches and directly attached via wired connection to NIPRNet in a Pentagon office, and logged in via MFA, then the totality of the acquisition data set is available within the application. If all things are equal except that the network is now a Wi-Fi connection from an internet cafe with VPN enabled, bid/pricing data is no longer accessible. If the same user is on a personal mobile device, only the RFP is accessible—no bid data.

As a final example, consider a second and concurrent login from an OCONUS location that occurs while the user is actively logged in from their Pentagon office. Instead of a blanket deny, a deny with countermeasures occurs. The system grants the attacker access, or, more accurately, routes them to a honey net that was populated with inaccurate bid data, including inaccurate DoDAF models and shuffled bid/pricing data. Instead
of giving a deny, deny with countermeasures creates a scenario in which cyber defenders actively mine the attacker's data and their tactics, techniques, and procedures (TTPs). Within minutes of the attack's conclusion, the greatest example of software defining tactics occurs: cyber professionals act on what they have learned and immediately update the PDP. Take this example and extrapolate: project it into the design of software systems for next-generation military platforms. ZT represents a cyber revolution in military affairs because it epitomizes how software defines defensive cybersecurity tactics, pivots from a binary outcome to one predicated upon combinatorics, andopaquely masks the characteristics of both strategy and tactics even when the corresponding doctrine is openly published. Knowing that engineers designed a system with ZT does not offer an attacker any insight if they breached defenses under "permit" or a "deny with countermeasures" conditions. In short, properly integrated ZT throughout a program's software artifacts has the potential to sow fear, uncertainty, and doubt about the data an attacker is accessing. The DoD needs to thoroughly think through ZT and meticulously include it through the software's architecture and throughout its novel algorithms. It has to consciously design it at every system boundary, at every API endpoint, and throughout the user interface itself so that screens do not offer a trace that there is, was, or could be more data than what the user has access to at this moment. In fact, these are the types of novel characteristics that a PEO should concentrate on because an SLA cannot capture them. The PEO who embraces ZT throughout their novel software implementation intrinsically realizes a greater degree of cyber resiliency then merely "shifting security left" into the modern software factory.

Corollary: Realize Digital Transformation without Explicitly Demanding It
Stepping back for a moment, shift the frame of reference away from planes, tanks, ships, vehicles, or whatever platform a PEO was established to advance. The realization of ZT is only truly possible when a PEO realizes all of the ACTS described in this report. When this happens, the silhouette that begins to form on the horizon is one of digital transformation. A simple search will reveal that we have not used this phrase, digital transformation, until this point—near the end of this report.

The DoD has been actively discussing its desire to digitally transform, articulating that it needs to do this to prepare for a great power competition. But what does it mean to digitally transform? In July 2019, the DoD published a Digital Modernization Strategy that defined four strategic initiatives, including one explicitly focused on resilient cybersecurity. In describing what this means, it uses many adjectives including seamless, agile, resilient, transparent, and secure. It established a Digital Modernization Initiative Executive Committee (DMI EXCOM) as a governance body for realizing digital modernization across the DoD.

Corollary: Realize Digital Transformation without Explicitly Demanding It
Stepping back for a moment, shift the frame of reference away from planes, tanks, ships, vehicles, or whatever platform a PEO was established to advance. The realization of ZT is only truly possible when a PEO realizes all of the ACTS described in this report. When this happens, the silhouette that begins to form on the horizon is one of digital transformation. A simple search will reveal that we have not used this phrase, digital transformation, until this point—near the end of this report.

The DoD has been actively discussing its desire to digitally transform, articulating that it needs to do this to prepare for a great power competition. But what does it mean to digitally transform? In July 2019, the DoD published a Digital Modernization Strategy that defined four strategic initiatives, including one explicitly focused on resilient cybersecurity. In describing what this means, it uses many adjectives including seamless, agile, resilient, transparent, and secure. It established a Digital Modernization Initiative Executive Committee (DMI EXCOM) as a governance body for realizing digital modernization across the DoD.

Through the eyes of a PEO, is digital modernization a perpetual activity analogous to the idea that software is never done? Or is it a set of projects to execute?

All too often, organizations demand digital modernization through ineffective mandates and misaligned metrics. A key metric that DoD CIO relies on is the financial expenditure rates that the owning service discloses during the annual budget certification. Not only do these rates not reflect progress, they often do not accurately reflect expenditures within the PEO's programs. Organizations have also proposed nonfinancial metrics to measure their digital transformation journey, from cloud adoption rates to usage of modern programming languages (not Ada or COBOL). But these metrics ultimately have marginal value in a quest to decisively declare that an organization has successfully completed its digital transformation.

Perhaps the most compelling way to realize digital transformation is to avoid explicitly demanding it. Digital transformation is
complex because it is both strategic and tactical, opaque and transparent, a journey and a destination. The DoD can reach the destination through the journey of advancing each pillar of the digital triad within every PEO, in equilibrium. Transformation materializes where digital logistics meets highly adaptable software-defined build chains in a modern software factory, where software defines policy outside the realm of brittle compiled executables to create high levels of cyber resiliency, where SLIs cleanly measure delivery of SLAs that offer certainty of execution, and where software refines a US kill chain faster than an enemy’s. In practice, the most compelling sign of a successful digital transformation of the DoD is that each of the 83 major defense acquisition programs and thousands of smaller efforts can showcase how software defines tactics.
EXERCISING ACTS: A VIGNETTE

Under pressure to deliver

This is a fictional exercise with fictional contractors. Any representational link to actual entities is purely coincidental and unintended.

Situation: It’s a Tuesday in July. You are the Navy program executive officer (PEO) for unmanned systems and are getting an update brief on an emerging Navy requirement for a new anti-submarine warfare system provisionally referred to as the Future Autonomous Underwater Submarine Tracker (FAUST) system. Extensive studies by the Office of Naval Research, N81, and DARPA have suggested that architectures like this are the only affordable, near-term option to track a large number of diesel submarines over a wide area. Budgetary constraints have forced the Navy to dive into an unconventional and high-risk acquisition—a Faustian bargain, if you will.

Key components of the FAUST reference design include:

- A containerized information processing system that can be hosted on multiple Navy vessels, including the littoral combat system, that supports tactical employment and possible target engagement
- Small unmanned underwater vehicles with an acoustic payload and a ship-based launch system
- Ship-launched unmanned aircraft capable of deploying small buoys with acoustic payloads
- Specialized algorithms for estimating submarine types and locations, coupled to acoustic payloads
- Terminals and connections to commercial satellite telecommunications providers, over which the system establishes a secure link for remote low-bandwidth data exchange

Photo Caption: Aerographer’s Mates from US Navy Fleet Survey Team utilize a IVER3-580 Autonomous Underwater Vehicle to scan the ocean floor for hazards on July 14, 2022, in Honolulu, Hawaii. (Royal New Zealand Air Force photo by Cpl. Dillon Anderson)
A support information system that can track logistics status and requests for the system and provide tactical tracking data to other military users via a secure web application deployed to SIPRNet.

The requirements are frustratingly fuzzy, and it seems your challenge will be as much expectation management as program management. Navy seniors seem to expect an operational capability to be delivered tomorrow despite an unproven architecture and brand-new acquisition approach. Your team has been tracking this requirement as the staffing process develops and refines it. Staff have begun performing market research in support of planning an acquisition strategy. They are also evaluating technical approaches that differ from the reference design. The Navy’s experience with similar modular system architectures—including counter-mine modules for the littoral combat system—inform thinking about the risks and challenges of a distributed system-of-systems acquisition.

As part of market research, staff meet with industry business development teams. We have summarized the most memorable meetings below.

PrimeCo, the third-ranking defense contractor last year, visits with an impressive briefing outlining a possible bid team, an engineering and development plan, and an overview of relevant prior work and promises to support milestone development. It claims that we can simply adapt or reuse most of the requisite software from prior programs and that it has the only track record including every one of the many dimensions of programmatic risk. PrimeCo strongly advocates a one-time competition for an integrated contract for the entire activity, suggesting that the PEO simply does not have the staffing to support any other approach. Although it did not describe the process in the initial brief, when questioned PrimeCo says that it will stand up an internal software factory, leveraging independent R&D investments that created a custom DevOps environment for its PrimeWorx division.

Snazz-Al, a budding venture-backed defense contractor, argues that the core technologies for the system-of-systems revolve around artificial intelligence and autonomy and suggests that PrimeCo lacks the software expertise or staff to deliver on a software-centric acquisition program. Snazz-Al has derived its early products from commercial technologies, and it has a reputation for rapid delivery. Snazz-Al has a highly unconventional request: it wants the government to bid out the entire capability under a lease model, to acquire the capability as a service. It argues that this aligns incentives and allows Snazz-Al to build in continual updates and inform software development dynamically. It would bring on subcontractor team members where needed and build out its entire plan around a DevOps approach, investing in advance and claiming it would start with delivering a minimally viable deployable capability only six months after contract award.

SETAsaurus, now the leading defense support services contractor, argues that the government should offer it a contract modification to be the system design agent for the program. It would build out the requisite infrastructure and establish reference designs and interface standards for the key components. It would even build out a custom cloud computing capability using commodity hardware and the OpenStack software set, which it claims is the only way the government can really know the environment is secure and maintainable. SETAsaurus built out a custom AI extract-transform-load capability and DevOps pipeline for another government customer using open-source tools, highlighting its technical prowess. It provides history showing its ability to staff up rapidly in preparation for a quick program start. It claims that a rapid sole-source start as system-design agent would let the government maintain control of a complex program, own the interfaces, and avoid vendor lock.

A handful of smaller companies that tend to specialize in individual elements of the requisite systems-of-systems also brief. Commercial UUV providers that primarily service the oil industry show off some impressive vehicles at low cost but do not seem prepared to answer questions about security or sustainment. An
analytics house presents some mind-blowing algorithmic results with much greater precision than expected. Another small firm specializes in developing web applications for secure government networks and shows off its experience using other deployment pipelines. A commercial firm produces acoustic transducers for other applications, but despite impressive frequency response, its application does not support the power output projected.

The new service acquisition executive, frustrated by a billion-dollar contract modification needed to modernize a key Naval radar system to counter an emergent electronic warfare technique, issues a memorandum mandating that all future systems acquisitions take steps to avoid vendor lock. Most of the staff interpret this guidance to mean that the government should claim unlimited rights or government purpose rights to source code and that they should be sure to ask for disks with source code in contract deliverables. Combining this with mandatory compliance checks for the Navy’s open architecture initiatives, they seem convinced that this can deliver a flexible acquisition approach that accommodates technical insertion without being beholden to a legacy contractor.

The PEO staff also meet with congressional staff tasked with oversight of Navy acquisition. They are concerned about the mixed track record of prior systems-of-systems and unmanned system programs, including LCS modules. They’ve also heard from companies that complain the Navy has a tendency to reinvent commercial tools, paying too much for technology that is two generations behind. One line of questioning revolves around the Navy’s failure to demonstrate sufficient technical maturity for the capability, and another focuses on the Navy’s reluctance or apparent inability to adopt state-of-the-art commercial tools, including cloud computing, modern communications, and AI tool sets.

Taken together, all of this input seems overwhelming and contradictory, the perfect complement to unrealistic customer expectations.

Breaking Up the Problem

You, as the PEO, recognize that FAUST will be a systems-of-systems effort in which the connection between systems is the essential ingredient. Realistically, this means software integration is the core technical risk to the effort even though it will also rely on unproven and uncrewed vehicles, communications links, and detection algorithms.

You select one of your junior program managers to lead this effort, who—despite having less experience than her peers—is clearly a digital native and had an algorithms specialization at the Naval Postgraduate School. You know that building out the right team is your first job. You crack open the quick reference guide in Software Defines Tactics and scan the list of ACTS until your finger settles on ACTS 4: Behave like a Software Recruiter. You remember the need to generate such a level of excitement that top talent is eager to accept a government position. With a quick Google search, you identify several techniques that successful programs like Platform One, Kessel Run, and Army Software Factory used to amass a thousand or more applicants in just a few weeks.

At a highly unconventional meeting, you get your team to collaborate on launching a LinkedIn campaign that announces the opportunity to help shape a new era of digital-first acquisitions. In the meantime, you work with human resources to identify an open billet that you can fill with a highly qualified expert (HQE) term position. Thanks to your persistence, you end up sourcing a retired commander, a former surface warfare officer who ended up as the chief operating officer of a Maryland-based healthcare software company. With his company recently acquired, he has great interest in applying his experience back to his home service. He joins as the architectural lead for the FAUST effort and dives right into helping structure this unconventional acquisition.

Recalling ACTS 1: Evaluate Existing Software Factories before Building Your Own, you decide to use your team to engage in a thorough survey. Working with your program manager, your
new architectural lead, and other key staff, you embark on a tour of Navy and Air Force software factories, building a picture of existing capabilities. Some of the most impressive software factories create applications that are only cloud-hosted and delivered as web applications. After touring seven software factories, your team is impressed by a new operation at the Naval Information Warfare Command (NIWC), supporting another Navy effort that also has a mixture of embedded and web development and similar classification needs.

Having selected a target software factory, your team starts to design the acquisition strategy and build out the core capability integration team. They compile this team from a combination of digital-savvy personnel recruited to internal billets, support from a federally funded research and development corporation (FFRDC), and a new task on SETAsaurus’s support contract. Your architectural lead works on establishing a “badgeless” culture in which the whole group works as one without implied hierarchy of government or contractor badge tags. Collectively, your team starts to sketch an acquisition structure that looks like figure 14. There are several broad horizontal bands of continuing activity—with a nexus at the integration and delivery of a software-centric capability for FAUST.

Recalling ACTS 5: Think in Service Levels, your team starts working through the logic of what it can express as an SLA. You are hoping to find as many such pieces as you can in order to drive down the oversight burden and smooth program cost profiles. The program manager ends up deciding that she cannot buy Snazz-Al’s argument; the FAUST capability is simply too complex to lease the entire capability as a service. The governing metrics for the capability are not clear yet, and the team has not yet adequately engaged the user community—even a broad measure like track quality over area is simply too undefined. However, you can cleanly hive off pieces of the capability, including satellite communications and computing infrastructure. These are mature commercial offerings that you can procure as a service. Reflecting on this further, since FAUST is a critical combat capability, the foundational point in this document that the DoD has to “operate as integrator” rings true. You will also be able to continuously complete these key horizontal components (those defined by SLAs) of the overall capability.

As the team keeps working through the acquisition strategy, it becomes increasingly clear that the entire value of the capability depends on the ability to detect, track, and classify—it is algorithmically oriented. Your program manager decides it is essential to establish a direct relationship (prime contract) with the algorithm developer, bringing them into the shared development environment of the software factory over at NIWC, and works to establish a simulation environment early. While the mission systems and core algorithm development are typically a small portion of the overall system cost, these capabilities are often relegated to subcontractor roles. In the case of FAUST, because of the rate of threat evolution, and the importance of iterating around operational feedback, it is essential to move these capa-
abilities into a central role, with a shared work environment with the core integration team.

Within just a few months, NIWC’s simulation testbed evolves into a first “dev” environment for integrated software release. A set of virtualized hardware represents the operational system, even representing the target classification environments and preliminary target hardware architecture. Using this, the mission systems algorithm team leveraged the software factory’s CI/CD automation and has already demonstrated its ability to deploy early versions of the key algorithms into the development environment within 10 months. Of course, it’s a long way from making it into the water or delivering operational results. But the risk profile is now entirely different from that of a conventional acquisition program.

There is no big-bang testing event when all of the capability is supposed to come together for the first time. Instead, your team integrates the entire capability from the beginning, even before selecting hardware.

Using an existing software factory gives you a massive head start on achieving authority to operate. Another effort using that software factory already has instrumentation for collecting performance data on distributed operational secret-level information systems, which can give the developers information on reliability. Reminding yourself of ACTS 2: Partner with an Authorizing Official to Realize a Continuous Authorization to Operate, you can locate a forward-leaning AO to achieve exactly this aim.

After a competition, PrimeCo wins a contract for integration with the Navy fleet, including equipping the containers with edge computing equipment and integrating the unmanned vehicle launch and recovery system into legacy platforms. PrimeCo’s unique expertise with ship alterations and combatant systems is a key part of its value proposition. Its program management approach revolves around a Gantt chart with diamonds for delivery of the control station and timelines for ship modification and testing.

Looking at the exploding opportunities in commercial unmanned vehicles, you decide to directly procure a handful of off-the-shelf systems and make them available for integration. However, Snazz-AI wins a contract for creating a tasking and guidance management system. It integrates with each of the unmanned systems and adds sidecar containers that have proven effective at handling secure tasking. Snazz-AI’s contract enables you to manage the unmanned vehicle portion of the development around the concept of a service level. You are now insulated from the risk of platform obsolescence.

Shortly after the initial delivery, a routine scan within the CI/CD pipeline reports that it detected a new critical vulnerability in one of the Snazz-AI containers. It reports that one of the open-source frameworks improperly handled session tokens, creating the possibility of a replay attack. The AO and their team begin to review the collective set of SBOMs and realize that across all Navy PEDs, this is the only instance of this framework they can find—a peculiar situation. Cyber threat intelligence seems to indicate that one of the core contributors to the library is affiliated with a known state-sponsored offensive cyberattack organization. The government works closely with the Snazz-AI team to identify a replacement framework. In less than 72 hours, the team has integrated the replacement framework, and the CI/CD pipeline’s suite of static and dynamic application security tests, API fuzzing tests, etc. are all indicating a green status.

After this episode, you and the team come together for a quick postmortem. The team reflects on how the selection of a mature software factory following an approved DoD DevSecOps reference design prevented this library from being deployed into an operationally relevant environment. The team also gained confidence that they can respond to future zero-day attacks at the speed of operational relevance, deploying patches in days or perhaps hours, not weeks or months. Lastly, this was the first time that the AO and their team demonstrated the value of an SBOM consumption plan. They wrote their own after-action report to share with other AOs, illustrating the value of SBOMs in identifying high-
er-risk libraries and frameworks and their use of that knowledge to collaboratively guide cyber threat intelligence analysts.

Recalling ACTS 3: Own Your APIs, you make sure to structure the development so that all of the key performers are working with APIs that are housed in a repository associated with the software factory. Your architecture lead is carefully monitoring the APIs to ensure that key sensor data sets are always accessible to the government via the API calls, not CD-ROM deliverables, further reducing concerns about proprietary implementations that may have been funded by internal R&D and excluded from GPR.

Snazz-AI complains initially about the additional friction of getting their developers onto an appropriate network and slightly different development tools. But eventually everyone gets on board and recognizes the value of this approach. The DoD structured these contracts such that involved parties always deliver APIs into or pull them from the shared development environment, offering Snazz-AI and other small business participants in the program the ability to protect their intellectual property.

The program team can accomplish continuous integration into the development environment with containerized software deliveries that definitively meet the expected APIs and SBOMs that provide the AO and their team with high-fidelity software composition analysis. Further, the software factory’s pipeline follows the principles of minimum viable continuous delivery, establishing the pipeline as the authority that determines whether software artifacts are suitable for release, all artifacts are immutable with no human changes after commit, and a production-like test environment minimizes configuration drift between development, test, and production environments.

Eleven months after you selected a junior program manager to run this effort, you spend the morning in a room at NIWC, watching a system exercise in a live, virtual, constructive (LVC) environment. Only the satellite commutations and the control station terminals are live, but nonetheless it’s an impressive demo. This operationally relevant test met the requirements of the Software Acquisition Pathway, and the software generates immense excitement across the user community and several other tangentially associated programs within your PEO. The system seems to be delivering high track quality and rate in almost every scenario.

During the test, a red team initiates a planned but unannounced simulated offensive cyberattack. The red team is targeting a subsystem that updates configuration files, an attack that could negatively affect the operational behavior of the sensors. As the architecture lead realizes what is happening, he smiles and leans back in his chair to watch the system respond, knowing that the program shifted security left into their CI/CD pipeline, and they went to great lengths to follow ACTS 6: Define Zero Trust Outcomes throughout, including a PDP and about half a dozen separate PEPs. The system detects the session replay attack and automatically initiates a deny with countermeasures response that stops the replay and locks out the account associated with the session token because it was compromised. Calmly, the security team also updates the policies at the PDP, and now that part of the system, an admin console for managing the configuration files, is no longer accessible to network traffic that does not originate from a trusted IP subnet.

You have not delivered the capability to the field yet, and there is a long way to go. But you walk away with the recognition that integration risk is no longer at, or near, the top of the program’s risk register. That is a remarkable accomplishment and a testament to how the application of specific ACTS can deliver programs where software defines tactics.

Fast-Forward Two Years
FAUST has achieved initial operating capacity and is already starting to demonstrate its value. The heady days of robust feature development have transitioned to tuning the integrations between the disparate systems and more routine patching.

You anticipated this scenario, as you recognize that software
development has a natural ebb and flow to it. The program’s partnership with an existing software factory is working exceptionally well during this time. As other programs are also using the same NIWC software factory, the cost to your program is nominal, and the CI/CD pipeline remains active and relevant. The staffing ratio between government and contractors has definitively shifted, with fewer contractors required because the program has wrapped up feature development.

Tensions have been building between the US and a peer adversary, and a routine intelligence sortie revealed a capability, possibly a wartime reserve capability, that the DoD does not recognize. To counter this capability, it assembles a task force. The task force quickly realizes that the data they need to respond to comes from three different programs spanning two services. Each of these programs followed the ACTS, and the government’s AI researchers should have access to all of the relevant data they need through APIs to begin modeling a countermeasure. The government saves an immense amount of time because it does not need to negotiate or buy access to its own data across multiple contractors.

A member of the task force, a career Navy chief, was one of the original users supporting the development of FAUST when the program first launched. She remembers how the lead architect walked around quoting one of the ACTS corollaries on a recurring basis: “The lollipop is more important than the enterprise architecture!” She pulls up the documentation, which is located in the same source code repository as the APIs and documented in markdown format. She spends about an hour reviewing the FAUST interfaces and points and finds what she is looking for.

The chief makes a call over to the Chief Digital and AI Office at the Pentagon. The next day at the Pentagon, the chief explains the evolving situation and the task force’s belief that the DoD can build an AI model to detect and counter this newly discovered threat. After a few handoffs, one of the government’s best AI teams in R&E has the data in hand to start modeling. They know precisely what interface they need to map to in order to engage the model, and the DoD has introduced them to the appropriate NIWC software factory personnel to deploy their model into the CI/CD pipeline that has been running continuously for several years now.

A few weeks pass, and the AI model has proven effective within the same operationally relevant LVC environment where the FAUST team initially demonstrated it a few years earlier. The model works, and after a quick review of the SBOM associated with the container where the model is encapsulated, the DoD continuously delivers it to two in-theater assets. The combatant commander’s cybersecurity team also tweaks the policy at the PDP, adding several brand-new deny with countermeasures rules that create further cyber resiliency.

A few weeks later, the DoD informs the task force that the AI model and the new policies worked flawlessly downrange. After the meeting, you realize that this . . . this is what the digital transformation that Pentagon leaders have been speaking about for over a decade looks like. Mandates, policies, misaligned metrics, or acquisition strategies could not have delivered such an effective countermeasure successfully at this speed. Instead, it was the explicit ACTS of these key programs that created this success.
APPENDIX A: DOCUMENTATION AS CODE

There is a profound recognition that the principles and corollaries captured in the ACTS will evolve and change over time, just like software. In Chapter 1, we openly disclosed that the focus of this report is primarily the software arm of the digital triad, inviting those with rich industry expertise in data and AI/ML to build on this body of work to support PEOs with actionable principles. In Chapter 4, we introduced and discussed the value of documentation as code.

We are taking a novel approach with this report, opting both to publish the static PDF and to establish a GitHub repository that contains the ACTS and their definitions in markdown format along with the report’s graphical images. It is our sincere hope that experts in data and AI/ML will literally build on this body of work. Adopting GitHub provides a lightweight consensus mechanism supported via threaded discussions associated with individual pull requests to ensure highly refined contributions.

We invite and encourage the broader community, including those in the DoD as military members or part of its civilian workforce and those who are part of the broader defense industrial base, to suggest edits and offer new ACTS using git branching and pull requests. Through collaboration, it may be possible to enhance the material and ensure its relevance over time. This approach also provides an opportunity to pivot away from social platform paradigms, like LinkedIn, that offer only a forum for academic dialogue and constructive criticism without an opportunity for meaningful follow-up actions.

The GitHub repository can be found in this location: https://github.com/Hudson-Institute-DC/peo-acts.
## APPENDIX B: LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>A horizontal screen resolution of around 4000 pixels</td>
</tr>
<tr>
<td>ACTS</td>
<td>The set of acquisition principles introduced in this document, Acquisition Competency Target for Software</td>
</tr>
<tr>
<td>AEGIS</td>
<td>The Aegis Combat System is an American integrated naval weapons system, integrated by Lockheed Martin</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AO</td>
<td>Authorizing Official</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>ARM</td>
<td>Formerly an acronym for Advanced RISC Machines - a processor architecture</td>
</tr>
<tr>
<td>ASICS</td>
<td>Application-specific integrated circuit</td>
</tr>
<tr>
<td>ASML</td>
<td>The Dutch company Advanced Semiconductor Materials International</td>
</tr>
<tr>
<td>ATO</td>
<td>Authority to Operate</td>
</tr>
<tr>
<td>AWK</td>
<td>A domain-specific language designed for text processing and typically used as a data extraction and reporting tool, available on Linux</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>C2I</td>
<td>Command, control and intelligence</td>
</tr>
<tr>
<td>C3PO</td>
<td>A robot from the Star Wars franchise</td>
</tr>
<tr>
<td>CAGR</td>
<td>Compound Annual Growth Rate (CAGR)</td>
</tr>
<tr>
<td>CCA</td>
<td>Collaborative combat aircraft - USAF uncrewed vehicle concept</td>
</tr>
<tr>
<td>CD</td>
<td>Continuous Delivery (as in CI/CD pipeline) or Compact Disc (as in CD-ROM)</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration (as in CI/CD pipeline)</td>
</tr>
<tr>
<td>CIO</td>
<td>Chief Information Officer</td>
</tr>
<tr>
<td>CISA</td>
<td>Cybersecurity and Infrastructure Security Agency</td>
</tr>
<tr>
<td>CNAP</td>
<td>Cloud Native Access Point</td>
</tr>
<tr>
<td>COBOL</td>
<td>Common Business Oriented Language - A programming language</td>
</tr>
<tr>
<td>COCO</td>
<td>Contractor Owned and Contractor Operated</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture, <a href="https://www.corba.org/">https://www.corba.org/</a></td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>CSP</td>
<td>Cloud service provider</td>
</tr>
</tbody>
</table>
CUDA  Compute Unified Device Architecture: a parallel computing platform from Nvidia
CUI   Controlled Unclassified Information
CVE   Common Vulnerabilities and Exposures, a database of publicly disclosed information security issues - https://cve.mitre.org/
DAF   The United States Department of the Air Force
DARPA Defense Advanced Research Projects Agency
DDIL  Denied, degraded, intermittent, or limited communications
DFARS Defense Federal Acquisition Regulation Supplement
DIB   Defense Industrial Base
DMI   Digital Modernization Initiative
DNS   Domain Name Service
DOD   Department of Defense
DODI  Department of Defense Instruction
DOE   Department of Energy
DSB   Defense Science Board
EA    Electronic Attack
ERP   Enterprise Resource Planning
EXCOM Executive Committee
FAQ   Frequently Asked Questions
FAR   Federal Acquisition Regulations
FAUST Future Autonomous Underwater Submarine Tracker
FFRDC Federally-Funded Research and Development Corporation
FPGA  Field Programmable Gate Array
FY18, 21, 23 Fiscal Year 2018, 2021, 2023
GAO   Government Accountability Office
GOCO  Government Owned, Contractor Operated
GOFO  General Officer/Flag Officer
GPR   General Purpose Rights
GPU   Graphics Processor Unit
GSMA  Global System for Mobile Communications, originally Groupe Spécial Mobile Association
HIL   Hardware-in-the-loop simulation
HIPAA Health Insurance Portability and Accountability Act of 1996
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRMS</td>
<td>Human Resources Management System</td>
</tr>
<tr>
<td>HQE</td>
<td>Highly Qualified Expert</td>
</tr>
<tr>
<td>IBM</td>
<td>The International Business Machines Corporation</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
</tr>
<tr>
<td>IRAD</td>
<td>Independent Research and Development</td>
</tr>
<tr>
<td>ISR</td>
<td>Intelligence, surveillance, and reconnaissance</td>
</tr>
<tr>
<td>JEDI</td>
<td>Joint Enterprise Defense Infrastructure (JEDI) Cloud solicitation</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation, json.org</td>
</tr>
<tr>
<td>JWCC</td>
<td>Joint Warfighting Cloud Capability</td>
</tr>
<tr>
<td>KEV</td>
<td>Key exploited vulnerability</td>
</tr>
<tr>
<td>LCS</td>
<td>Littoral Combat System</td>
</tr>
<tr>
<td>LMS</td>
<td>Learning management system</td>
</tr>
<tr>
<td>LVC</td>
<td>Live, virtual, and constructive simulation environment</td>
</tr>
<tr>
<td>MFA</td>
<td>Multi-factor authentication</td>
</tr>
<tr>
<td>MIL-STD</td>
<td>Military Standard</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MOCAS</td>
<td>Mechanization of Contract Administration Services</td>
</tr>
<tr>
<td>MQ-9</td>
<td>An uncrewed aircraft known as the Reaper</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean time to recovery</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NDAA</td>
<td>National Defense Authorization Act</td>
</tr>
<tr>
<td>NIPR</td>
<td>Non-classified Internet Protocol (IP) Router Network (NIPRNet)</td>
</tr>
<tr>
<td>NIWC</td>
<td>Naval Information Warfare Command</td>
</tr>
<tr>
<td>OCONUS</td>
<td>Outside the Continental United States</td>
</tr>
<tr>
<td>OMB</td>
<td>Office of Management and Budget</td>
</tr>
<tr>
<td>OODA</td>
<td>Observe–orient–decide–act</td>
</tr>
<tr>
<td>OPIR</td>
<td>Overhead Persistent Infrared satellite</td>
</tr>
<tr>
<td>OPM</td>
<td>Office of Personnel Management</td>
</tr>
<tr>
<td>OSD</td>
<td>Office of the Secretary of Defense</td>
</tr>
<tr>
<td>OTA</td>
<td>Other Transactions Agreement</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PEO</td>
<td>Program Executive Officer</td>
</tr>
<tr>
<td>PEP</td>
<td>Policy Enforcement Point</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor, a scripting language</td>
</tr>
<tr>
<td>PII</td>
<td>Personally identifiable information</td>
</tr>
<tr>
<td>PLM</td>
<td>Product Lifecycle Management</td>
</tr>
<tr>
<td>PM</td>
<td>Program Manager</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>POND</td>
<td>Period of Non-Disruption</td>
</tr>
<tr>
<td>PPA</td>
<td>Program, Project, and Activity</td>
</tr>
<tr>
<td>PPBE</td>
<td>Planning, Programming, Budgeting, and Execution - budgeting process</td>
</tr>
<tr>
<td>R2D2</td>
<td>A robot from the Star Wars franchise</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer - an architectural style</td>
</tr>
<tr>
<td>RF</td>
<td>Radiofrequency</td>
</tr>
<tr>
<td>RFP</td>
<td>Request for Proposals</td>
</tr>
<tr>
<td>ROM</td>
<td>Rough order of Magnitude or Read-Only Memory (as in CD-ROM)</td>
</tr>
<tr>
<td>SAF</td>
<td>Secretary of the Air Force</td>
</tr>
<tr>
<td>SATCOM</td>
<td>Satellite Communications</td>
</tr>
<tr>
<td>SBOM</td>
<td>Software Bill of Materials</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>SISO</td>
<td>Senior information security officer</td>
</tr>
<tr>
<td>SLA</td>
<td>Service-level agreement</td>
</tr>
<tr>
<td>SLI</td>
<td>Service level indicator, that measures progress against an SLO</td>
</tr>
<tr>
<td>SLO</td>
<td>Service level objective, a metric within an SLA</td>
</tr>
<tr>
<td>SM-6</td>
<td>Standard Missile 6</td>
</tr>
<tr>
<td>SOA</td>
<td>A Service-oriented architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOC</td>
<td>System on a chip</td>
</tr>
<tr>
<td>SRE</td>
<td>System Reliability Engineers</td>
</tr>
<tr>
<td>SWAP</td>
<td>Software Acquisition and Practices - a study by the Defense Innovation Board. Also Size, Weight, and Power</td>
</tr>
<tr>
<td>TB</td>
<td>Terabyte, a unite of data size</td>
</tr>
<tr>
<td>TTP</td>
<td>Tactics, techniques, and procedures</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned aerial vehicle</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator - a web address</td>
</tr>
<tr>
<td>URN</td>
<td>Uniform Resource Name</td>
</tr>
<tr>
<td>US</td>
<td>United States of America</td>
</tr>
<tr>
<td>UX</td>
<td>User experience</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual private network</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>WHS</td>
<td>Washington Headquarters Services</td>
</tr>
<tr>
<td>WWI</td>
<td>World War I</td>
</tr>
<tr>
<td>XKCD</td>
<td>A comic, <a href="https://xkcd.com/">https://xkcd.com/</a></td>
</tr>
<tr>
<td>ZT</td>
<td>Zero Trust</td>
</tr>
</tbody>
</table>
ENDNOTES


24 Danzig, Driving in the Dark, 13.
27 Ford, Parsons, and Kua, Building Evolutionary Architectures, 117.
28 Ford, Parsons, and Kua, Building Evolutionary Architectures, 2.
39 Drawn from conversations between one of the authors and the FedEx founder, Fred Smith.
50 Eric Lofgren, “Does Lockheed Own All F-35 Data Created by Military Users? Problems of IP,” Acquisition Talk (blog), November
SOFTWARE DEFINES TACTICS: STRUCTURING MILITARY SOFTWARE ACQUISITIONS FOR ADAPTABILITY AND ADVANTAGE IN A COMPETITIVE ERA


PrEP4All, “Deploying the Government Owned, Contractor Operated Model.”


The library can be referenced online at https://dodcio.defense.gov/library/. It includes the DevSecOps Guides and Playbooks


Visit https://xkcd.com/927/ for the cartoon, which illustrates the futility of achieving truly common standards.


