NPS-AM-23-005

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Charting Progress in the Software Acquisition Pathway

December 2022

Capt Richard S. Wahidi, USAF

Thesis Advisors: Jeffrey R. Dunlap, Lecturer
Dr. Robert F. Mortlock, Professor

Department of Defense Management

Naval Postgraduate School

The research presented in this report was supported by the Acquisition Research Program of the Department of Defense Management at the Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact the Acquisition Research Program (ARP) via email, arp@nps.edu or at 831-656-3793.

ABSTRACT

The Department of the Navy (DON) recently implemented the Department of Defense (DoD) Software Acquisition Pathway (SWP), a software acquisition strategy for custom application and embedded software. The purpose of the SWP is to enable rapid and iterative delivery of high-priority software capability to the intended user. But while the SWP uses an agile software development approach, neither the DoD nor the DON have yet provided comprehensive governance tools and methods for SWP programs to iteratively plan, track, and assess acquisition outcomes in agile environments. To close this gap, the author systematically researched commercial software engineering management and digital product development practices as well as prior DoD software acquisition reform studies. Based on the results, the author showed that Earned Value Management is incompatible with the SWP and recommended alternative techniques to measure cost and schedule performance. Additionally, the author recommended a phased approach to manage DON SWP custom application programs, whereby a minimal, unitless work breakdown structure is used to track progress until demonstrating the minimum viable product to the user in a testing environment; product-based metrics are then tracked until initial release of the custom application software; and then outcome-based goals are iteratively set, tracked, and assessed using the Objectives and Key Results framework for as long as the custom application software is in use.

THIS PAGE INTENTIONALLY LEFT BLANK

ABOUT THE AUTHOR

Capt Richard Wahidi graduated from the University of Indiana East in Richmond, IN in 2017 with a Bachelor of Science in Mathematics and commissioned through the United States Air Force Officer Training School. Capt Wahidi's first assignment was at Robins Air Force Base, GA, where he served as a Contract Manager within the F-15 System Program Office and then as a Contracting Officer within the Electronic Warfare and Avionics Program Office. After graduating from the Naval Postgraduate School, he will be reporting to the Air Force Installation Contracting Center headquarters in Wright-Patterson Air Force Base, OH, where he will serve as the Deputy Director of the Commander's Action Group.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors, Professor Jeffrey Dunlap and Dr. Robert Mortlock, as well as Dr. David Tate, Eric Lofgren, Brian Misuraca, Emily Miller, Sarah Arnold, Kristina Botelho, Matt MacGregor, and Melissa Naroski-Merker for sharing their advice and wisdom. Each provided me critical, timely feedback to help strengthen my thinking, my research, and to develop my passion for learning.

THIS PAGE INTENTIONALLY LEFT BLANK

NPS-AM-23-005

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Charting Progress in the Software Acquisition Pathway

December 2022

Capt Richard S. Wahidi, USAF

Thesis Advisors: Jeffrey R. Dunlap, Lecturer
Dr. Robert F. Mortlock, Professor

Department of Defense Management

Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AAF	Adaptive Acquisition Framework
ADM	acquisition decision memorandum
AgileEVM	agile earned value management
ALM	application life cycle management
ASN(RD&A)	Assistant Secretary of the Navy for Research, Development, and Acquisition
ATO	authority to operate
BV	business value
CapEx	capital expenditures
CFD	cumulative flow diagram
CI/CD	continuous integration, continuous delivery
CNS	capabilities needs statement
COTS	commercial-off-the-shelf
CSCI	computer software configuration item
CVE	common vulnerability or exposure
DA	Decision Authority
Dev	development engineers
DevOps	development-IT operations
DevSecOps	development-security-IT operations
DIB	Defense Innovation Board
DoD	Department of Defense
DON	Department of the Navy
DORA	DevOps Research and Assessment
EBV	earned business value
EVM	earned value management
EVMS	earned value management system
IT	information technology
JCIDS	Joint Capabilities Integration and Development System
KR	key result
LOC	lines of code
LOE	level of effort
MECE	mutually exclusive and collectively exhaustive
MTTR	mean-time-to-restore
MVCR	minimum viable capability release

MVP	minimum viable product
NDIA	National Defense Industrial Association
NPD	new product development
OKRs	Objectives and Key Results
Ops	IT operations engineers
OpEx	operational expenditures
OS	operational sponsor
OUSD(A&S)	Office of the Under Secretary of Defense for Acquisition and Sustainment
OUSD(R&E)	Office of the Under Secretary of Defense for Research and Engineering
PM	project/program manager
PMB	performance measurement baseline
PMO	project management organization
SDLC	software development life cycle
SMART	specific, measurable, actionable, realistic, and time-bound
SP	story point
SWAP	Software Acquisition & Practice
SWP	Software Acquisition Pathway
SWS	software work structure
UCD	user-centered design
UI	user interface
VA	value assessment
WBS	work breakdown structure
WIP	work in progress
WP	work package

I. INTRODUCTION

The software engineering process—the continuous process of planning, designing, developing, integrating, testing, deploying, and operating software—has become the most critical means of creating and delivering value in the Information Age economy (Kersten, 2018). Because it is both dynamic and indefinite, the need to leverage this engineering process has forced businesses, including the Department of the Navy (DON), to develop entirely new ways of work. However, only private sector companies have redesigned their operations and management around software. On the one hand, successful companies sense and respond to market demand by continuously delivering digital capability rapidly and iteratively (Defense Innovation Board [DIB], 2019b). On the other hand, the DON still delivers software at the pace of discretely planned projects, which hamstrings its ability to field modern digital capabilities effectively.

This may seem like a technical problem, but it is a management problem. The DON has already recognized the technological criticality of software. For this very reason, the DON recently implemented the Department of Defense (DoD) Software Acquisition Pathway (SWP), a software acquisition strategy that enables rapid, iterative, and indefinite software delivery. The DON's SWP drives a fundamental shift from a waterfall software engineering approach to an agile software engineering one, wherein the goal is to continuously deliver the highest value software to the operational environment as soon as practicable to iteratively shape subsequent development cycles. Iterative development requires iterative assessment, though, and the SWP has not yet provided comprehensive tools and methods to plan, track, and assess software acquisition outcomes in an agile way of work.

Consequently, while the DON's SWP enables iterative development and incremental delivery of software capability, the acquisition workforce executing the SWP must still rely on long-range, detailed planning metrics and management methods to assess day-to-day software development activities. To close this management gap, this researcher evaluates state-of-the-art software engineering practices and proposes a modern software acquisition management framework for DON SWP custom application programs where none presently exists.

A. BACKGROUND

Software is what powers Information Age technologies, so creating competitive advantage in today's world requires businesses to understand and adapt to the state-of-the-art practices of the software engineering industry. Because the agile software engineering movement began in 2001 (Beck et al., 2001), the software engineering industry has all but perfected the agile approach. Furthermore, the proliferated use of cloud computing in products, software development tool kits, and data centers in the past decade has both commoditized and virtualized all the computing resources necessary to produce and deliver software code, such that new digital products or digital product features can be shipped to customers at scale in minutes (DIB, 2019b). As a result, businesses aligned to agile software engineering best practices adapt, maneuver, and grow in the market at software speed, whereas businesses not yet aligned quickly expose themselves to risk.

For its part, the military already requires agility in all aspects of its business, from technology development in acquisitions to technology employment in combat operations. The former defense acquisition executive, the Honorable Frank Kendall, said it best in his 2015 *Performance of the Defense Acquisition System* assessment: "Simply delivering what was initially required on cost and schedule can lead to failure in achieving our evolving national security mission—the reason defense acquisition exists in the first place" (Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics [OUSD(AT&L)], 2015, p. iv). Thus, by driving a shift from waterfall to agile software engineering, the new SWP both aligns the DON to software engineering industry best practices as well as supports greater institutional capacity for change. Realizing the SWP's intended business agility is critically necessary for the DON to accomplish its never-ending, no-fail missions in the Information Age. Yet, the new SWP requires transformational change in not just the DON's technical practices, but in its management practices, too. The shift from waterfall to agile is a shift from plan-driven acquisition to adaptive acquisition—from reactive to proactive approaches throughout the software acquisition practice.

To manage performance, the DON SWP requires its programs to track, at a minimum, four metrics: lead time, cycle time, deployment frequency, and change failure rate (Assistant Secretary of the Navy for Research, Development, and Acquisition [ASN(RD&A)], 2022). However, these are software delivery performance metrics, which are applicable only after each program's first software release has occurred (Forsgren et al., 2018). The DON SWP has not yet provided metrics and management methods to cover the initial development and delivery of a new, custom software application.

Without additional guidance, it is unclear how DON SWP custom application programs should track and assess project progress and program performance in an entirely new way of work. For instance, while on-time and on-budget implementation of a detailed specification constituted success in a waterfall software engineering environment, the SWP does not identify how to manage cost and schedule resources when practicing agile methods. Furthermore, upon initially releasing their software to operations, DON SWP custom application programs become responsible for not only managing cost and schedule resources, but also for software deployment rate, response rate, code quality, functional performance, and nonfunctional attributes of the fielded software application (DIB, 2019a). But given the current lack of a software acquisition management framework, the DON SWP does not address how DON SWP programs should effectively trade off between and manage these performance factors. As a result, the DON SWP lacks sufficient metrics and metrics guidance to cover both initial development and deployment of a custom software application, as well as longer-term outcomes once the software application is in use.

Moreover, the DON SWP does not exempt custom application software programs from Earned Value Management (EVM) requirements. Thus, all cost-reimbursement and incentive-type software programs of $20 million or more must implement a work breakdown structure, performance measurement baseline, detailed work packages, and so forth, and deterministically perform activities in compliance with long-range, detailed plans, despite the stochastic nature of agile software engineering processes (Section 809 Panel, 2018). While DON SWP custom application programs can request exemption from EVM, they still need tools and techniques to manage cost and schedule resources in an agile way of work.

Some early agile practitioners attempted to modify EVM by truncating its planning horizon and adapting its formulas to agile software engineering practices. This resulted in a lean set of mathematical formulas—dubbed AgileEVM—that can be used for agile software project control and monitoring (Sulaiman et al., 2006). When extra care is taken to identify and track scope creep during iteration/sprint planning, research supports AgileEVM's utility in quantifying cumulative cost and schedule progress in agile software environments, particularly in those practicing the Scrum style (Hodson, 2016).

In the end, however, rapid and iterative software delivery requires a management framework designed around creating and delivering value, which is continually defined, refined, and assessed in the eyes of the user. Given the primacy of software in the Information Age economy, the most successful businesses visualize their end-to-end software development processes and clearly and consistently track all types of the processes' work—features, defects, risk, and debt (Kersten, 2018). In doing so, such organizations enable themselves to continually create and deliver functional software, as well as protect their abilities to do so by continually planning and managing nonfunctional, yet critical, improvement work (Kersten, 2018). On the other hand, there is yet no taxonomy for DON SWP programs to identify software development features, defects, risk, and debt work items in their program backlogs, let alone a comprehensive management framework for them to make all work items visible and plan them accordingly. Consequently, the DON's most critical means of creating and delivering value—the software development process—remains an elusive black box to its stakeholders. Yet now more than ever, DON SWP programs must open and master this black box to successfully execute their mission.

B. PURPOSE AND SIGNIFICANCE OF STUDY

This researcher systematically studies commercial project management, new product development, and modern software engineering practices to propose a management framework for DON SWP programs that are acquiring custom application software. Accordingly, the purpose of this study is to enable DON SWP custom application programs to capture the ground truth of software development work and steer it to accomplish the program's mission outcomes. To accomplish this purpose, the objectives of this systematic study are to identify (a) the salient differences between planning and managing software development work using waterfall and agile methods; (b) several classes of software development performance metrics based on the type of software acquired, phase of software system development, and intended business objectives; (c) non-EVM methods to track cumulative cost and schedule progress in agile environments; and (d) how to effectively structure and govern digital product development processes and practices in the Information Age.

The potential contributions of this research are far reaching. Primarily, it could facilitate the institutional adoption and effective execution of the DON's new software acquisition strategy, the DON SWP. Secondarily, it could either supplant or revamp EVM by identifying project monitoring techniques that avoid long-ranged, detailed planning and/or minimize business system requirements, which can potentially streamline software acquisition management, reduce the costs of compliance for commercial software developers, and lower barriers to entry in the DON custom application software marketplace. More broadly, by instantiating a simple yet robust software acquisition management framework, this research could enable the DON to elevate its software acquisition practices, leverage them to accelerate digital technology adoption, and continuously deliver new software capability at the speed of relevance.

C. RESEARCH QUESTIONS

The goal of this research is to propose a software acquisition management framework for DON SWP custom application programs. As such, the primary research question is

1. What metrics should the DON use to assess agile/incremental program performance?

However, given the extensive history of efforts to improve DoD software acquisition, as well as significant efforts to make EVM compatible with agile software engineering methods, this researcher also investigates several secondary questions:

2. What are the leading tools, monitoring and control methods, and management practices to track and review software acquisition progress and performance?
3. Should EVM be replaced or augmented as the standard for acquisition program performance?

4. What are the metrics being recommended by the Defense Science Board (DSB), Defense Innovation Board (DIB) Software Acquisition and Practices Study, and Section 809 Panel?

Answering these questions requires a systematic review of commercial software engineering, project management, and new product development best practices; evaluation of current DON and DoD software acquisition practices; a review of the research on EVM's implementation in agile software engineering environments; and analysis of prior DoD software acquisition advisory reports. The next section includes the approaches taken to answer these questions.

D. METHODOLOGY

The problem examined in this study is framed by one main research question and three secondary research questions. Answering the main research question, "What metrics should the DON use to assess agile/incremental program performance?," relied on a systematic review of the prescribed management practices and business objectives of the DON SWP, DoD SWP guidance, software engineering industry literature, DoD software acquisition studies, and agile software engineering metric guides. Next are the secondary research questions.

The question "What are the leading tools, monitoring and control methods, and management practices to track and review software acquisition progress and performance?" was answered by examining DON and DoD acquisition policy, program management guides, defense acquisition literature, DoD software acquisition advisory reports, and an overview of software development tools currently available to the DoD and the DON. The next question, "Should EVM be replaced or augmented as the standard for program performance?," was answered by carefully reviewing the literature on EVM published since 2001, the year when the agile software engineering movement first emerged (Beck et al., 2001). Finally, answering "What are the metrics being recommended by the DSB, DIB Software Acquisition and Practices Study, and Section 809 Panel?" relied upon analysis of all prior recommendations to streamline federal acquisition and improve defense software acquisition practice.

In summary, accomplishing this research involved comprehensive, systematic analysis of the extant software engineering, software project management, new product development, and defense software acquisition bodies of knowledge.

E. FRAMEWORK

Chapter I, Introduction, served several purposes. First, the chapter set the stage by acknowledging the primacy of software and software development processes in the production and distribution of Information Age technology. Second, it articulated the project's purpose and why this study may benefit the DON's future software acquisition efforts. Finally, it established the academic research framework by identifying the research questions to be answered and explaining how the answer for each question was developed.

Chapter II, Literature Review, includes a discussion of the relevant software engineering industry literature, prior research on software engineering management practices, and DoD acquisition and software acquisition advisory reports examined as part of this project. This section also includes an overview of software engineering, project management, new product development, and software acquisition history to provide additional context for the current disparities between software engineering technical practice and software engineering management practice within the DoD and the DON.

Chapter III, Agile Software Engineering Metrics, includes a discussion of the unique complexities of agile software engineering methodologies in comparison with their traditional waterfall counterpart. It then highlights the most common measures and metrics used in agile software engineering environments, including project progress, program performance, functional requirements, nonfunctional requirements, and development productivity. For DON SWP programs developing custom software on commercial computing infrastructure, a phased set of software development progress and performance management metrics is recommended. Finally, because metrics function as incentives, the chapter highlights numerous principles and patterns to enable DON SWP programs to effectively utilize agile software engineering metrics.

Chapter IV, Agile Software Engineering Management, discusses the current and best practice methods and tools available to visualize and manage progress and performance of acquisition programs. It also examines the interactions between EVM and agile software engineering and the recommendations of congressionally commissioned and DoD-level advisory reports on streamlining acquisition and improving software acquisition practices, and it highlights several principles and patterns to enable effective digital transformation leadership.

Chapter V, Conclusion, presents a condensed summary of this research project's findings, identifies the limitations with respect said findings and makes final recommendations to potentially improve the DON's software engineering management practices where necessary. According to this project's research, these recommendations may strengthen DON SWP programs' ability to execute their intended mission outcomes.

F. SUMMARY

This concludes Chapter I, Introduction. This chapter included an overview of the project's purpose, the academic research problem, and the project's scope by detailing the gap between the demands of agile software engineering environments and the management tool kit presently available to practice DON software acquisition. This chapter also discussed the anticipated research benefits, planned the research methodology, and provided an overview of the report's structure. Chapter II, Literature Review, discusses the existing academic literature associated with software engineering, project management, new product development, and DoD software acquisition. By reviewing how the gaps between commercial and defense software engineering practices developed, Chapter II builds a foundation for informed analysis and making recommendations to close these gaps in subsequent chapters.

II. LITERATURE REVIEW

This chapter provides an overview of software engineering, project management, new product development, entrepreneurial management, and software acquisition reform literature from the late 20th century through today. The intent of synopsizing the literature over this period is to provide a socio-technological foundation of the engineering management principles and practices that have influenced the professionalization of software engineering, as well as to inform analysis of the structural and cultural factors involved in designing and implementing an effective software engineering management framework. For the sake of clarity and brevity throughout this chapter, *agile* means agile software engineering, *agile project* means a software development project that practices agile software engineering, *agile project management* means the discipline of managing software development projects that practice agile software engineering, *waterfall* means waterfall software engineering, *waterfall project* means a software development project that practices waterfall software engineering, and *waterfall project management* means the discipline of managing software development projects that practice waterfall software engineering.

A. EARLY SOFTWARE ENGINEERING

The commercialization of the computer in the 1950s sparked a Cambrian explosion of data processing application demand that overwhelmed the programming community (Mahoney, 1990). In response to a growing software market crisis, the North Atlantic Treaty Organization Science Committee hosted a conference entitled "Software Engineering" in the fall of 1968 (Naur & Randell, 1969). The purpose this conference was to establish the foundations of a professional software engineering discipline based on systems engineering theory, unique properties of digital systems, and the extant technical and management practices of other engineering branches (Mahoney, 1990). Based on the unprecedented success of industrial engineering and its prevailing management thinking in the early-to-mid 20th century, the conference participants proposed a sequential software system engineering process that would be governed using mass-production techniques (Mahoney, 1990). For instance, software design and development tasks were to be separated, and managers were to measure and control all software development activities using manufacturing-oriented techniques (Mahoney, 1990). However, the conference participants disagreed upon which software system design approach was better: (a) a top-down approach that began design activities outside the system and progressively worked down to define each component in greater detail or (b) a bottom-up approach that began system design by building basic modules and gradually integrating them together to create increasingly complex combinations (Naur & Randell, 1969). Because bottom-up design purportedly risked creating a sub-optimal system, top-down design was favored, as it required first defining and specifying system components, which was intended to maximize the use of state-of-the-art technologies (Naur & Randell, 1969).

By 1970, however, Winston Royce (1970), a leading software engineer for space mission control software, strongly warned against using sequential system design processes. Royce observed that for building small, simple applications within one's own company, stage-gate sequencing of software engineering activities may suffice. But Royce (1970) argued that as problem complexity increases, stage-gate sequencing results in the following monolithic software development process, shown in Figure 1, which was "doomed to failure" (p. 328):

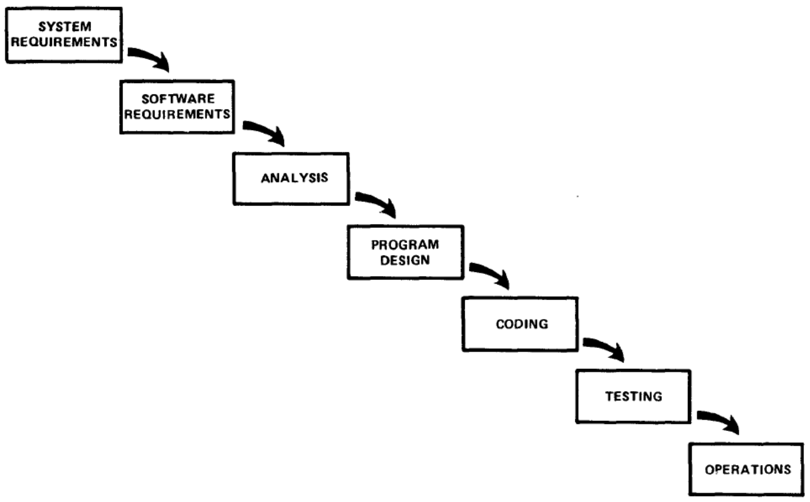
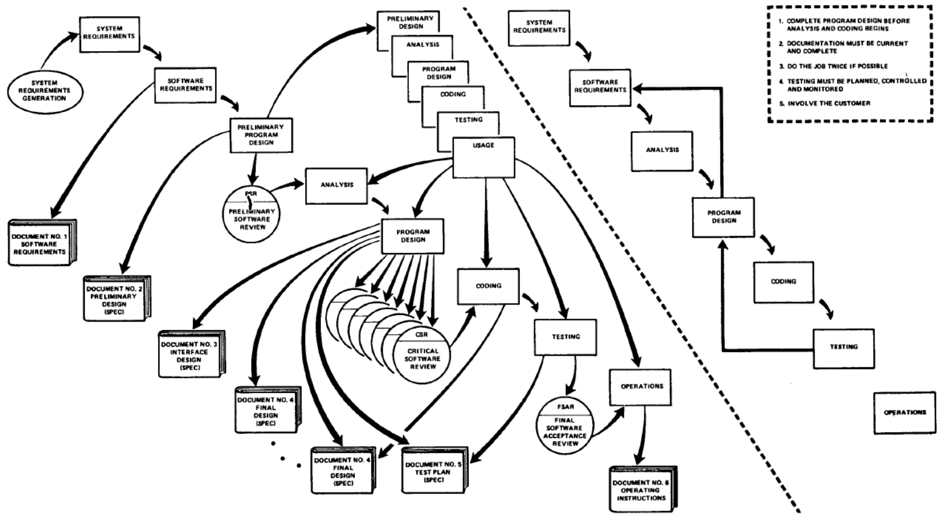


Figure 1. Waterfall Software Development Process. Source: Royce (1970).

Royce's (1970) rationale was that because software system requirements are emergent (i.e., they're identified based on how system modules interact rather than what modules do individually; Bahcall, 2019), there is a need to continually validate software system behavior during implementation. To do so, Royce argued that testing should inform program design, and program design should influence test requirements throughout the development process. Additionally, Royce argued that software design should involve the user, that resources should be allocated at the correct place at the correct time, and that isolating developers from software specification activities was unacceptable. Although the coining of the term waterfall is often misattributed to him (Bell & Thayer, 1976), Royce was the first software engineer to model a stage-gate, sequential software development life cycle (SDLC), warn against its limitations, and offer an alternative model to improve mission outcomes. To better account for the complexity of the software development process and the emergent behavior of software systems, Royce proposed an iterative approach to software development, as shown in Figure 2:



SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 5 TEST PLAN (SPEC)
CSR
CRITICAL SOFTWARE REVIEW
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER

Figure 2. Iterative Software Development Process. Source: Royce (1970).

Unfortunately, however, the DoD (1985) ended up adopting and standardizing the following stage-gate, sequential SDLC model in *Military Standard: Defense System Software Development*, as shown in Figures 3 and 4:

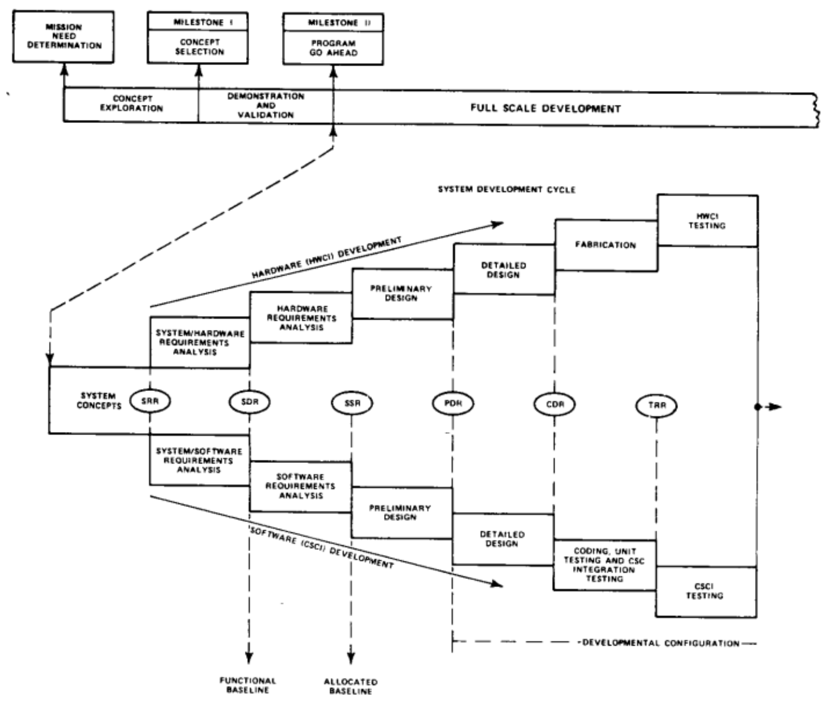

MISSION NEED DETERMINATION
MILESTONE I
CONCEPT SELECTION
MILESTONE II
PROGRAM GO AHEAD
CONCEPT EXPLORATION
DEMONSTRATION AND VALIDATION
FULL SCALE DEVELOPMENT
SYSTEM DEVELOPMENT CYCLE
HARDWARE (HWCI) DEVELOPMENT
HWCI TESTING
FABRICATION
DETAILED DESIGN
PRELIMINARY DESIGN
HARDWARE REQUIREMENTS ANALYSIS
SYSTEM/HARDWARE REQUIREMENTS ANALYSIS
SYSTEM CONCEPTS
SRR
SDR
SSR
PDR
CDR
TRR
SYSTEM/SOFTWARE REQUIREMENTS ANALYSIS
SOFTWARE REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
DETAILED DESIGN
CODING, UNIT TESTING AND CSC INTEGRATION TESTING
CSCI TESTING
SOFTWARE (CSCI) DEVELOPMENT
DEVELOPMENTAL CONFIGURATION
FUNCTIONAL BASELINE
ALLOCATED BASELINE

Figure 3. Software Development Within the System Life Cycle. Source: DoD (1985).

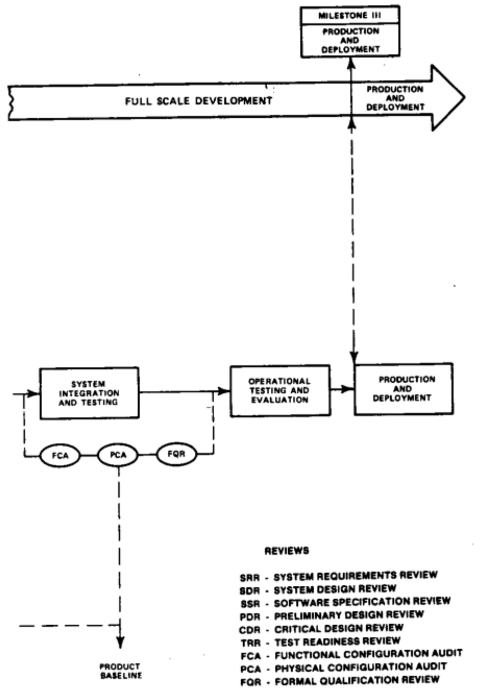

MILESTONE III
PRODUCTION AND DEPLOYMENT
FULL SCALE DEVELOPMENT
PRODUCTION AND DEPLOYMENT
SYSTEM INTEGRATION AND TESTING
OPERATIONAL TESTING AND EVALUATION
PRODUCTION AND DEPLOYMENT
FCA
PCA
FQR
PRODUCT BASELINE
REVIEWS
SRR - SYSTEM REQUIREMENTS REVIEW
SDR - SYSTEM DESIGN REVIEW
SSR - SOFTWARE SPECIFICATION REVIEW
PDR - PRELIMINARY DESIGN REVIEW
CDR - CRITICAL DESIGN REVIEW
TRR - TEST READINESS REVIEW
FCA - FUNCTIONAL CONFIGURATION AUDIT
PCA - PHYSICAL CONFIGURATION AUDIT
FQR - FORMAL QUALIFICATION REVIEW

Figure 4. Software Development Within the System Life Cycle (Continued). Source: DoD (1985).

While *Military Standard: Defense System Software Development* authorized more than one SDLC iteration to be in progress at the same time (DoD, 1985), it also constrained developers by mandating that "the contractor shall code and test units in a top-down sequence, unless alternate methodologies have been proposed … and have received contracting agency approval" (DoD, 1985, p. 31). By 1987, a DoD study recommended moving away from a "document-driven, specify-then-build" approach to a more user-centric, iterative process (Brooks et al., 1987, p. 3). However, no new project management tools were provided to replace the waterfall-oriented tool kit (Brooks et al., 1987).

B. DEFENSE PROJECT MANAGEMENT

As commercial and noncommercial development projects became increasingly complex in the 1950s, there was a need for effective techniques to synchronize planning, monitoring, and management of engineering activities (Stretton, 2007). The creation of the Project Management Institute in 1969 helped professionalize the project management discipline and circulate proven practices and specialized knowledge throughout industry (Stretton, 2007). Some of these included the Critical Path Method and the Program Evaluation and Review Technique (Stretton, 2007). But since the DoD outsourced technology development, it needed a common tool to capture timely, reliable snapshots of performance on highly costly development projects such as ballistic missiles (Abba, 2017). To standardize management practices of such defense acquisition programs, the DoD in 1967 established the Cost/Schedule Control Systems Criteria—the original guidelines for an effective EVM system (Abba, 2017).

EVM provided the DoD and defense contractors with a convention for tracking progress and assessing program performance and has helped maintain efficient performance in complex defense projects (Abba, 2017). EVM would become the DoD's preferred management control system for major defense acquisition programs, and its techniques and implementation standards have remained relatively intact (Abba, 2017). But to consistently capture accurate, timely data, the DoD eventually began to enforce EVM and EVM system requirements through contractual provisions, turning EVM into an audit-oriented oversight mechanism (Abba, 2017). Additionally, since EVM relied on detailed, upfront planning, its metrics and techniques had limited value when a project's primary objective was accelerating time to market (Kenney, 2021).

C. LEAN PRODUCTION

While DoD policy influenced the development of specific software engineering and project management practices, the lean production movement in the 1980s was entirely commercially driven. Lean production originated due to the need for a customer-oriented system for designing, manufacturing, and distributing automobiles in increasingly global markets (Poppendieck, 2011). But as its principles and patterns became adopted in other industries, lean production changed the rules of all new product development (NPD) processes, not just that of cars (Takeuchi & Nonaka, 1986). NPD is defined as the process by which companies imagine, create, and bring valuable products and services to market (Imai et al., 1984). In attempting to implement lean production, companies learned that it takes more than high quality, low cost, and customization to create competitive advantage in increasingly globalized and faster moving markets—it also took speed and flexibility (Takeuchi & Nonaka, 1986). Emphasizing speed and flexibility throughout NPD demanded changes in project design and project team culture, especially since sequential or *relay race* coordination processes conflicted with business goals of maximum speed and flexibility (Takeuchi & Nonaka, 1986).

One proposed solution was using flattened, cross-functional NPD teams, whereby a closely integrated team of marketing, engineering, manufacturing, and other business function members continuously collaborated like a rugby team, figuratively passing the ball back and forth until its intended product was brought to market (Takeuchi & Nonaka, 1986). Instead of standardizing development project schedules, the NPD design process would also be tailored to each individual team and its product needs. Figure 5 illustrates the NPD teaming approaches used at this time: Type A is the relay race approach; Type B is the phased overlap approach, where teams collaborated only to meet project milestones; and Type C is the integrated approach, where teams continuously collaborated for the sake of maximizing NPD outcomes (Takeuchi & Nonaka, 1986).

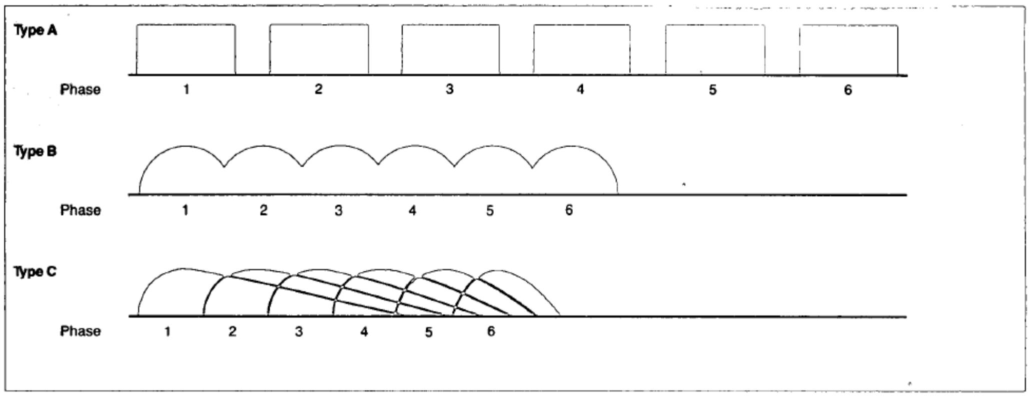


Figure 5. NPD Teaming Approaches. Source: Takeuchi and Nonaka (1986).

Implementing lean production took more than transforming NPD teaming approaches, however. To systematically drive new product ideas to market faster, with fewer mistakes, and with improved market adoption rates, the stage-gate NPD governance system was created, which conceived NPD as a production system and sought to maximize its quality by minimizing process variability (Cooper, 1990). To implement stage-gate governance, organizations divided their NPD process into stages, established quality control gates in between each stage, and required the project manager (PM) and project teams to satisfy predetermined exit criteria to proceed to subsequent stages, as shown in Figure 6 (Cooper, 1990):

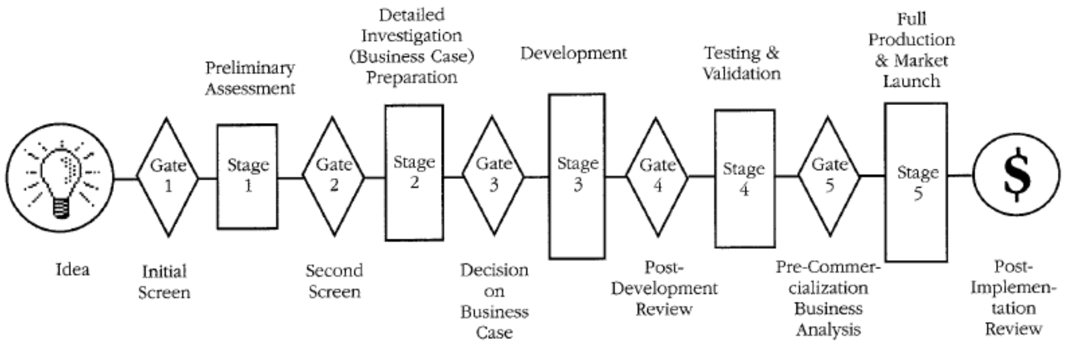

Idea
Gate 1
Initial Screen
Preliminary Assessment
Stage 1
Gate 2
Second Screen
Detailed Investigation (Business Case) Preparation
Stage 2
Gate 3
Decision on Business Case
Development
Stage 3
Gate 4
Post-Development Review
Testing & Validation
Stage 4
Gate 5
Pre-Commer-cialization Business Analysis
Full Production & Market Launch
Stage 5
$
Post-Implemen-tation Review

Figure 6. The Stage-Gate NPD Governance System. Source: Cooper (1990).

Stage-gate governance was intended to both inform and motivate PMs and project teams by clearly defining what inputs were required and how said inputs would be evaluated in advance (Cooper, 1990). Because weak market orientation and assessment were often cited as the reasons for new product failures, stage-gate governance also controlled quality by driving detailed marketing research into the NPD process (Cooper, 1990). Thus, just like waterfall software engineering, stage-gate governance prescribed detailed planning and design as the primary method to reduce risk (Cooper, 1990).

But while stage-gate governance improved time to market, businesses still lacked the means to dynamically sense and respond to new product ideas and consumer market segments (Kahn, 1996). To continually account for value in the eyes of the customer while meeting business needs, researchers recommended augmenting stage-gate governance with the value proposition framework shown in Figure 7 (Hughes & Chafin, 1998):

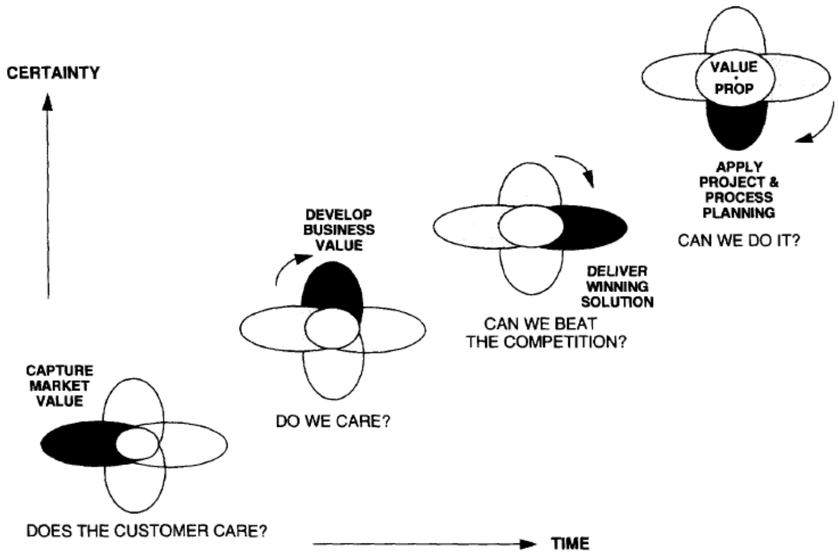


Figure 7. Value Proposition Framework. Source: Hughes and Chafin (1998).

The problem, however, was that the value metrics used at this time predominantly focused on discretely measuring and controlling costs (Hughes & Chafin, 1998). Without the metrics to implement the value proposition framework, businesses operating on stage-gate governance remained poorly equipped to adapt to globalized, accelerating markets.

D. THE INTERNET ERA

Because consumer software applications were highly modular and appealed to a wide variety of market segments, marketing professionals researching the software industry in the early 1990s learned that product managers typically worked within a company's engineering organization, not its marketing organization (Workman, 1993).

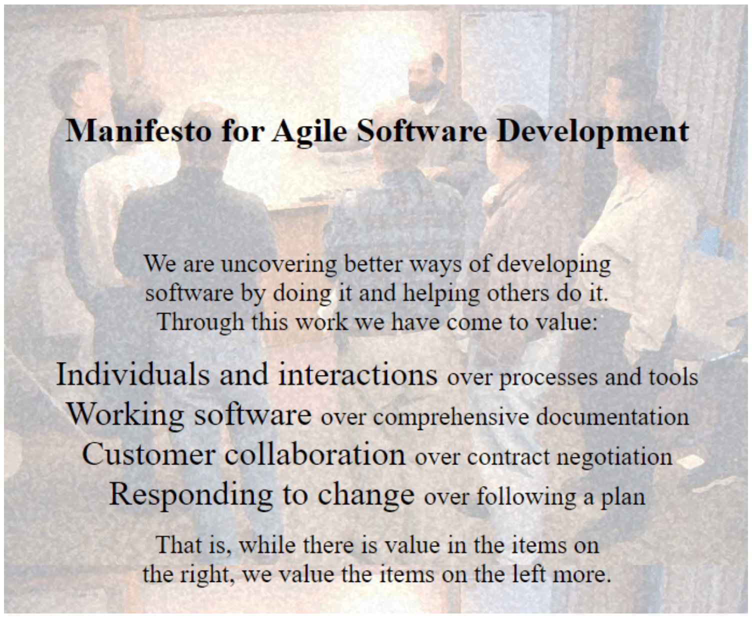
There, product managers assumed responsibility for coordinating, specifying, and managing product and market requirements, whereas PMs assumed responsibility for managing engineering and manufacturing activities (Workman, 1993). Additionally, because engineers themselves struggled to keep abreast of cutting-edge capabilities and competitors' latest approaches, the marketing function's influence on the engineering organization, and on the business's overall NPD process, was limited (Workman, 1993).

Naturally, predicting customers wants and needs in dynamic markets was impossible, but what marketing professionals learned was that software businesses needed the capacity to quickly change direction as new market information became available (Workman, 1993). Additionally, they learned that software businesses needed more robust approaches to continually develop new markets, discover breakthrough innovations, and mitigate unexpected technical and market risks (Wind & Mahajan, 1997). Unfortunately, while stage-gate governance reliably screened out NPD ideas that didn't meet a priori criteria, they also quickly funneled out product opportunities that could create new markets. Once commerce extended to the Internet, the need to reform and adapt marketing research practices to the software industry became a matter of survival (Wind & Mahajan, 1997).

Similarly, software engineers desperately struggled to cope with the new business environment. While waterfall software engineering tools and methods complied with stage-gate NPD governance systems, they required specifying detailed requirements upfront, designing a detailed solution, and then implementing the detailed design, which limited software engineers' ability to adapt to increasingly uncertain, complex, faster-moving markets. Invariably, bringing a new software product to market would take years. But because the underlying market problem tended to change, the detailed software solution also became irrelevant by the time it was available to customers. Out of desperation to replace his company's software products in 6 months to avoid going out of business, leading software engineer Jeff Sutherland adapted lean production practices to software development, enabling him and his colleague Ken Schwaber to deliver seemingly impossible software projects on time, under budget, and with fewer bugs than ever before (Rigby et al., 2016). Soon afterwards, Sutherland and Schwaber joined 15 other leading software engineers in a summit to change the course of the software engineering industry.

E. FROM WATERFALL TO AGILE

In 2001, 17 software engineers seeking to promote better ways of developing software published the *Manifesto for Agile Software Development* (the Agile Manifesto), the core doctrine for agile software engineering methodologies (Beck et al., 2001). The signatories of Agile Manifesto declared four values and 12 principles to guide agile practice, shown in Figure 8 and Figure 9:



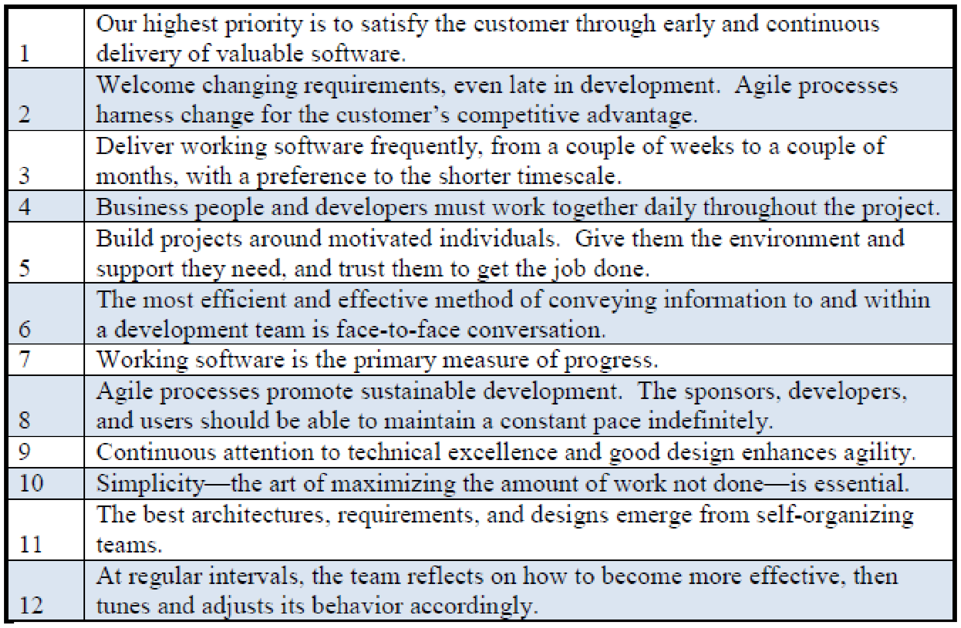
Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 8. The Agile Manifesto's Four Values. Source: Beck et al. (2001).



1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4	Business people and developers must work together daily throughout the project.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity—the art of maximizing the amount of work not done—is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figure 9. The Agile Manifesto's 12 Principles. Source: Beck et al. (2001).

The agile movement's intent was to lead the entire software industry away from documentation driven, heavyweight waterfall software engineering practices that were business as usual at the time (Lapham et al., 2011). The Agile Manifesto was a call to action to build organizational models based on people, collaboration, and the types of generative cultures in which its founders knew software engineers would love to work (Lapham et al., 2011). To drive the necessary paradigm shift, the Agile Manifesto's founders adapted lean principles to the software engineering domain (Hartmann & Dymond, 2006). For instance, the first lean principle—define value in the eyes of the customer—became "our highest priority is to satisfy the customer" and "deliver working software frequently" within the Agile Manifesto (Hartmann & Dymond, 2006, para. 17).

As a result, although waterfall and agile methodologies contained the same engineering tasks, the industry-wide impetus to rapidly and iteratively deliver small batches of working software to the user forced software engineers to improvise radically new ways of work (Lapham et al., 2011). On the one hand, waterfall projects comprehensively planned the design, sequentially performed engineering tasks according to plan, deferred plan changes via management controls, and delivered one monolithic software program at a predetermined completion date (Winterowd, 2013). On the other hand, agile projects progressively designed a system based on customer feedback, performed engineering activities in parallel, and delivered small batches of software during iterations/sprints, each of which was a microcosm of the SDLC and output a working software product (Winterowd, 2013). Even though the Agile Manifesto had not offered a new SDLC model, software engineers who practiced its principles and values converged upon the development approach shown in Figure 10 (Lapham et al., 2011):

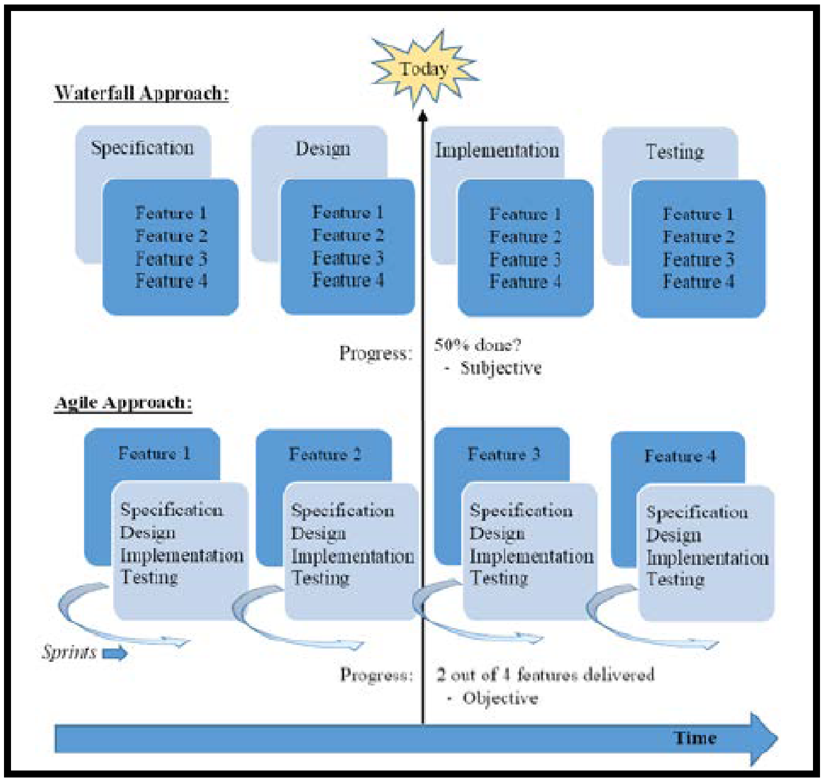


Figure 10. Transitioning from Waterfall to Agile Development. Source: Lapham et al. (2011).

Given the nature of iterative, incremental engineering, agile also required a new approach to NPD. In some respects, waterfall software engineering simplified NPD by anchoring on top-down system design (Naur & Randell, 1969). However, because each agile iteration spanned the SDLC, engineers sought to generate the maximum amount of useful information in each design step, whether through top-down or bottom-up system design choices (Reinertsen, 1997). For instance, to reduce risk for a new user-facing application, the user interface (UI) would generally be designed and tested first. Another fundamental difference between waterfall and agile software engineering lay in the terminology used to plan, manage, and control software work product entities. While waterfall development projects utilized Computer Software Configuration Items (CSCI) to identify, describe, and manage software system entities (DoD, 1985), agile environments used *stories*, which were defined as "promises for conversation" with the customer because they represented requested exchanges for something of value, the details of which were fleshed out through ongoing dialogue between the customer, product owner/manager, and development team (Rawsthorne, 2006, p. 2). Thus, stories replaced CSCIs as the smallest fundamental unit of value, requirements, and work through which software engineers, engineering managers, internal business stakeholders, and external customers interfaced in agile development projects (Rawsthorne, 2006).

But because stories consist of a description, size, validation criteria, and desired business value, managers were not sure how to measure them in the context of each project as well as the NPD process (Rawsthorne, 2006). Naturally, agile emphasized iterative development and incremental delivery of products rather than following a defined process correctly, so plan-driven metrics received lesser priority (Hayes et al., 2014). However, the right metrics could provide insight into a project's throughput as products were incrementally built, which was crucial to enable oversight in organizations that contracted out software development (Hayes et al., 2020). Thus, to understand and account for the context, work product, and outcomes of agile development teams, new management methods and metrics became critically necessary (Hayes et al., 2020).

F. AGILE PROJECT MANAGEMENT

Once the agile movement began, researchers questioned what project management and NPD processes and practices were most appropriate for agile environments (Karlström & Runeson, 2005). While stage-gate governance required long-range, detailed planning and allocated fixed resources before engineering work began, agile methods iteratively designed and developed products using a process that was permeable to change (Karlström & Runeson, 2005). Because agile methods aimed to deliver the highest priority features at any given time, agile environments also needed to establish feedback loops that could continually validate market demand for work in progress (WIP), specifically by determining which more important features to add or scale up as well as which less important features to drop or scale down (Karlström & Runeson, 2005).

One research effort studied three agile projects and observed that “microplanning” before agile iteration/sprints and making trade-offs for feature changes frustrated PMs until they became accustomed to its philosophy for dynamically managing scope changes and leveraging frequent product feedback (Karlström & Runeson, 2005, p. 46). Furthermore, the study noted that re-baselining Gantt charts and specifications became impractical in volatile, fast-paced business environments, so project management and NPD governance systems that could tolerate frequent requirement changes were necessary (Karlström & Runeson, 2005). That said, despite far more frequent scope changes, the researchers noted that implementing smaller sized stories improved software engineers’ ability to focus, gain cadenced control over their work, and develop a better understanding of a system’s complex, technical inner workings (Karlström & Runeson, 2005).

As for managers, PMs in all three cases observed that agile projects resulted in more tangible evidence of project progress and performance; improved communications within the product team and across the business; and improved cost control, product functionality, and timely delivery (Karlström & Runeson, 2005). But while engineers felt greater control and reduced uncertainty due to iteratively delivering in small steps and gaining fast product feedback, PMs initially believed that they lost control without their usual planning and management tools to describe the state of ongoing work (Karlström & Runeson, 2006). Moreover, PMs observed it was imperative not to misapply management tools and methods intended for waterfall projects (Karlström & Runeson, 2006), for waterfall and agile use diametrically opposed project designs, as shown in Figure 11 (Patel, 2021).

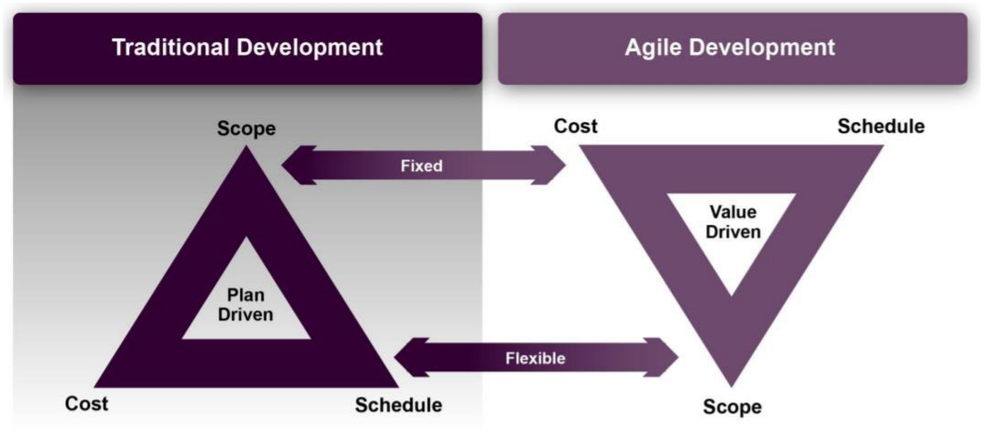


Figure 11. Traditional Versus Agile Project Design. Source: Patel (2021).

While waterfall, or traditional, projects are designed with fixed scope and flexible cost and schedule constraints, agile projects are designed as a series of mini projects, each of which has fixed cost and schedule constraints but flexible scope (Patel, 2021). The purpose of designing agile projects in this manner is to create a process that enables building the best product possible with all remaining project resources (Patel, 2021). To do so, agile practitioners adopted new tools and methods to support flexible software requirements. Typically, agile environments identify, plan, and manage software requirements using a backlog, which is not a detailed software specification but a dynamic ledger that is continually updated by adding, removing, scaling up, scaling down, and/or re-sequencing work items based on current customer priorities and developers' cumulative knowledge of their emergent product (Winterowd, 2013).

Thus, when using agile planning methods, project progress could be measured against work completed in the backlog. That said, measuring backlog progress didn't address how to forecast the total estimated cost and schedule of agile projects beyond the use of "yesterday's weather" metrics (i.e., the amount of activity development teams could likely complete in next iterations based on their current throughput levels; Hodson, 2016). As a result, although the lack of long-ranged, detailed planning made EVM inapplicable, managers who relied on EVM's total estimated cost and schedule metrics were initially reluctant to adopt agile practices (Rawsthorne, 2006). Over time, however, managers learned that building a product in one big batch required measuring project progress by proxies (e.g., documents), whereas iteratively building the product one feature at a time required measuring attributes of the product itself (Miller, 2020). Furthermore, leading software engineers observed that NPD governance systems incorrectly treated software development as a repetitive manufacturing process (Kersten, 2018). Thus, while agile enabled development teams in engineering organizations to much more effectively develop technology, businesses themselves could not become agile until they systematically adapted their philosophy and practice of project management and NPD.

G. LEAN DEVELOPMENT

While lean production practices set the standard for cutting-edge NPD in the late 20th century, misapplying them to software engineering often worsened outcomes for several reasons (Reinertsen, 1997). First, unlike manufacturing work, software engineering fundamentally involved nonrecurring design and development, so it tended to contain significant uncertainty in task arrival times and task durations (Reinertsen, 1997). Second, while more homogenous, repeatable manufacturing work could use first-in, first-out (FIFO) task sequencing because job order didn't matter, queueing discipline was necessary to dynamically manage nonhomogeneous software engineering workflows (Reinertsen, 1997). Third, while manufacturing processes output physical products that could only be in one place at any time, software engineering processes output information that could be in several places at once (Reinertsen, 2009). Fourth, because lean production practices targeted and minimized variability, they created toxic environments for creative knowledge work such as software engineering, which needed to tolerate some variability to enable innovation in early development (Reinertsen, 2009). Consequently, misapplying manufacturing tools and methods exacerbated software development queues, or WIP (Thomke & Reinertsen, 2012). Furthermore, because software engineering work was bits and bytes scattered on various disk drives, not piles of physical objects on the factory floor, neither developers nor managers could see when they were operating with dangerously high WIP levels (Reinertsen, 2009). The result was a vicious cycle for agile environments.

Software system requirements are emergent (i.e., new software system requirements become apparent only as their system modules interface and/or through customer use; Pelrine, 2011), so agile development teams routinely discovered new requirements during iterations/sprints. Naturally, development teams needed marginal capacity to be able to adapt to emergent requirements. But because lean production assumed all work was predictable and repetitive, managers tended to fully allocate all capacity during iteration/sprint planning (Reinertsen, 2011), which led to chronically overloaded teams, prolonged cycle times, and ever-growing WIP (Thomke & Reinertsen, 2012). Unlike manufacturing processes, where adding 5% more WIP would take 5% more time to complete, adding 5% more WIP could take anywhere from 5%–100% longer in software because of the nonlinear relationship between capacity utilization and cycle time in software engineering processes, as shown in Figure 12 (Thomke & Reinertsen, 2012):

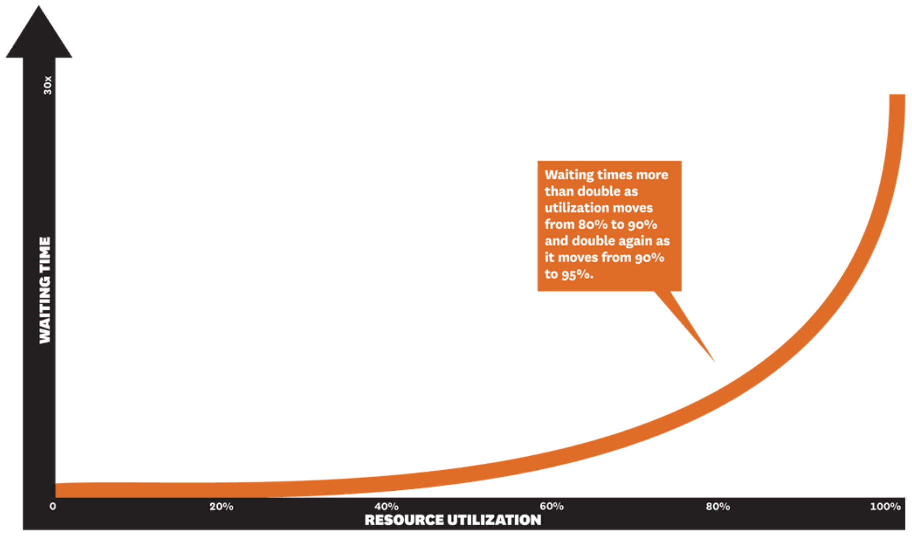


Figure 12. Waiting Time Versus Resource Utilization. Source: Thomke and Reinertsen (2012).

Thus, as WIP accumulated and awaited available capacity, overall project durations and NPD time to market grew (Thomke & Reinertsen, 2012). Because excessive WIP also delayed market feedback, they reduced opportunities for businesses to dynamically adapt their product to market conditions before it was too late, which ultimately defeated the purpose of practicing agile methods in the first place (Thomke & Reinertsen, 2012). To avoid this vicious cycle, a new set of practices, entitled lean development, was synthesized to maximize value and minimize waste in software engineering work (Reinertsen, 2009). Lean development differed from lean production in several ways. First, PMs needed to treat the software development process as a queuing system, wherein the backlog formed the waiting line, developers functioned as servers, total time in the system was time to market, and PMs exercised queuing discipline to dynamically schedule development tasks based on operational priority and technological dependencies (Reinertsen, 2009). Second, development teams needed to persistently maintain some marginal capacity to better cope with the inherent variability of software engineering work (Reinertsen, 2009). Third, software development WIP needed to be consistently tracked and made visible to all stakeholders (Reinertsen, 2009). Fourth, because software engineering work could much more easily undergo parallel processing, the sequences in which NPD activities were conducted needed to be individually tailored based on optimal economic value (Reinertsen, 2009). Generally, risky steps that could be inexpensively eliminated, such as a risky UI concept that could jeopardize NPD investments, needed to be tested as early as possible (Reinertsen, 2009).

Overall, 20th century managerial frameworks leveraged conformance and efficiency to optimally accomplish static business goals, as exemplified by lean production (Reinertsen, 2009). However, Information Age markets demanded a shift from static to dynamic business goals, whereby businesses operated by continually sensing and responding to new market information to make the best possible decisions in ever evolving circumstances (Reinertsen, 2009). Lean development resulted in two major breakthroughs. First, it enabled the software engineering industry to lead the shift from static to dynamic business goals. Second, it led to a powerful NPD methodology for startups, enabling them to create new products and capture new markets at software speed (Reinertsen, 2009).

H. ENTREPRENEURIAL MANAGEMENT

To survive and flourish under fierce Information Age competition, commercial firms needed to conduct NPD in less time and under tighter budgets (Gansler & Lucyshyn, 2013). But even for commercial-off-the-shelf (COTS) tax preparation software, the typical technology and product development timeline was as follows: start coding in September, release the first beta the following June, then release the second beta in July (Ries, 2011). Because beta testing only checked for critical failure modes, such as crashing people's computer, by that point only major bugs could be fixed—the product design itself had become locked (Ries, 2011). For simple COTS software, both this technology development approach and the project team were maladapted to their business environment (Ries, 2011).

To provide better alternatives, two high-technology (high-tech) entrepreneurs, Steve Blank and Eric Ries, suggested that because the project team developing this COTS software did not yet know who their customer was or what the product should be, it was a startup (Ries, 2011). Furthermore, because stage-gate NPD governance systems assumed prior operating history and a relatively stable market for which to conduct planning and forecasting, they were ill-suited for Information Age startups (Ries, 2011). To close this gap, Blank and Ries combined customer-driven marketing, agile software engineering, and lean development management practices to create Lean Startup, a NPD methodology which increased the odds of building a successful venture in several ways (Ries, 2011).

First, Lean Startup offered a simple definition of a startup: "A startup is a human institution [organization] designed to create a new product … under conditions of extreme uncertainty" (Ries, 2011, p. 27). This definition applied to anyone working in any organization, whether a government agency, a venture-backed company, a nonprofit, or a for-profit company (Ries, 2011). Second, Lean Startup broadly defined a product: "any source of value that customers experience during their interaction with a company whether at a brick-and-mortar store, an e-commerce website, a consulting service, and/or a nonprofit agency" (Ries, 2011, p. 28). Third, to quickly discover and validate demand in new markets or market segments, Blank and Ries argued that startups should quickly build and deploy a minimum viable product (MVP) containing only critical features to create a feedback loop with potential customers, enabling startups to continually search and execute on business opportunities through iterative market development and incremental product delivery (Blank, 2013).

Lean Startup was a breakthrough paradigm shift from yearlong NPD cycles that presupposed knowledge of customers' problems and product needs, required excessive time and resources upfront, and tended to fail in the market without providing the means to correct course (Blank, 2013). Instead, product managers practicing Lean Startup favored rapid and iterative customer feedback throughout NPD, which systematically improved business outcomes in two critical ways. First, because Lean Startup defined the means of survival as the amount of pivot opportunities left, it incentivized startups to effectively use scarce resources and rapid learning to succeed (Ries, 2011). Second, because it forced startups to methodically formulate value and growth hypotheses, gather empirical evidence to test them, and regularly prove to sponsors why they should pivot, persevere, or stop receiving funding and/or operating (Ries, 2011), Lean Startup also drove scientific rigor in NPD governance and business decision-making (Goldratt, 1997).

I. THE CLOUD ERA

By 2011, Silicon Valley luminary and high-tech venture capitalist Marc Andreessen (2011) famously penned an article claiming that "software is eating the world" (para. 2). Andreessen's article signaled that software had eclipsed hardware in economic utility, for businesses' ability to rapidly reach and acquire new customers—not their ability to mass-produce a consumer good—had become the most critical determinant of survival in the Information Age. According to Andreessen (2011), the commoditization and virtualization of the technology stack as well as the global adoption of high-speed Internet meant that people could launch a software startup and operate in markets all over the world without needing to invest in infrastructure and/or train new employees (Andreessen, 2011).

Because of the speed by which it was created and propagated relative to hardware, software would also eat "much of the value chain of industries that are widely viewed as primarily existing in the physical world" (Andreessen, 2011, para. 24). Thus, Andreessen (2011) argued that digital transformation had become imperative for every organization, and he challenged every company to learn how high-tech companies operate; what the consequences of commoditized, ubiquitous computing are for their industry; and how their industry could collectively become digitally native. Because the DoD primarily buys software, Andreessen (2011) suggested that to continually acquire the most cutting-edge software available, the DoD's technology development, project management, and NPD practices should not conflict with those of the software engineering industry.

J. MODERN SOFTWARE ENGINEERING

The state-of-the-art practice in the software engineering industry is exemplified by the Development-IT Operations (DevOps) movement (Kim et al., 2021). DevOps is a set of cutting-edge cultural norms, technical practices, and architectural approaches that enables businesses to deliver digital products and services to market rapidly and continuously via a highly agile, secure, and reliable underlying system of software engineering work (Kim et al., 2021). Cloud computing served as the primordial soup for this DevOps "system" of work by enabling the mass commoditization, virtualization, and proliferation of all the computing resources and tools necessary to create, distribute, and manage software for the software engineering industry (Surbiryala & Rong, 2019). Cloud-native software engineering innovations include automated provisioning and management of computing infrastructure, on-demand integrated development environments for building software, continuous integration and continuous delivery (CI/CD) pipelines that automate the software build and delivery process, and so forth.

As these innovations converged within an organization's IT ecosystem, they enabled software engineers to develop new ways of doing creative design and development work, rapidly prototype and demonstrate new product ideas to customers, and substantially accelerate the speed by which businesses could deliver new and/or improved products to market. Moreover, just as software engineers developed new tools and practices to manage a cloud-native software development process, agile PMs, product managers, and product owners developed new integrated application life cycle management (ALM) tools and practices to support cloud-native project management and NPD (Rossberg, 2019). Because these ALMs connected to software development workflows, they enabled PMs/product managers/product owners and software developers to share end-to-end visibility of all work, and they also enabled PMs/product managers/product owners to dynamically manage priorities, reallocate resources, and communicate with both technical teams and business stakeholders using real-time NPD information (Özkan & Mishra, 2019).

The software engineering industry considers DevOps a logical continuation of the agile movement that began in 2001, for DevOps principles and patterns naturally emerge when self-organizing teams focus not only on continually shipping high-quality code to customers, but also checking code into trunk daily, maintaining code in a deployable state, and demonstrating features to customers in production-like testing environments (Kim et al., 2021). To extend DevOps principles and patterns throughout the business, however, engineering organizations needed to be restructured (Kim et al., 2021). Because businesses segregate their capital expenditures (CapEx) and operational expenditures (OpEx), engineering organizations were typically bifurcated into one branch designed for CapEx-funded new development work (Dev) and another designed for OpEx-funded support work of existing offerings (Ops; Kim et al., 2021). Unfortunately, because Dev's metrics incentivized maximizing new development throughput, whereas those of Ops incentivized maximizing non-functional systems engineering attributes such as reliability, security, availability, the goals and behaviors of Dev and Ops were fundamentally misaligned in this organizational construct (Kim et al., 2021). The bifurcation of Dev and Ops created structural and cultural barriers, resulting in a "wall" between these software engineers (Kim et al., 2021). Consequently, when IT organizations planned work in large batches and didn't operate in an integrated, cloud-native IT ecosystem, the inherent separation between Dev and Ops personnel and processes inevitably led to a vicious circle of deteriorating quality, increased system outages, gradual buildup of technical debt, and slower and slower release of new products and features for the overall business (Kim et al., 2021).

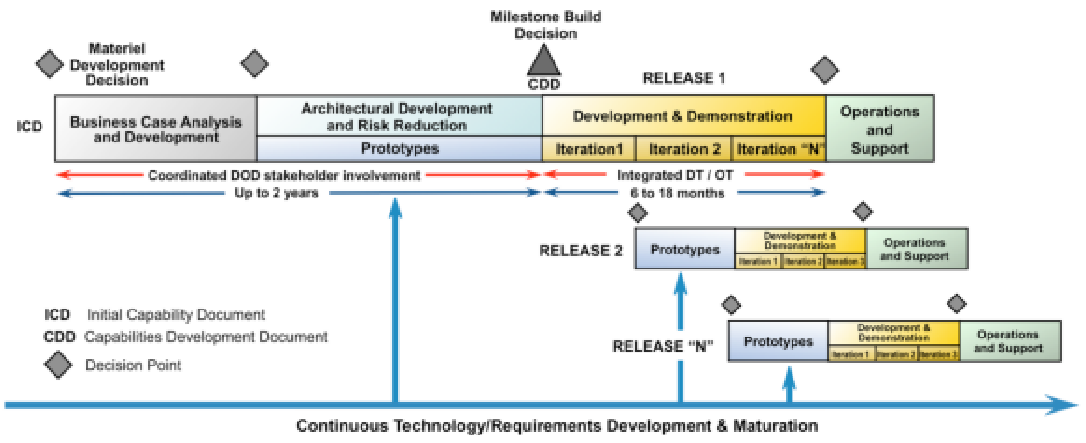
This downward spiral occurred because of a core, chronic conflict of diametrically opposed goals and incentives: Dev assumed responsibility for deploying features and changes into production as quickly as possible to respond to market demands, whereas Ops assumed responsibility for providing customers with stable, secure, reliable IT service (Kim et al., 2021). Configured this way, Dev optimized for large-batch developments and/or time to market, whereas Ops optimized for operational availability, reliability, security, and stability of the IT service, often by making production changes more difficult (Kim et al., 2021). Consequently, despite serving in the same technology organization, Dev and Ops created negative externalities for each other (Singer & Friedman, 2014); and their software engineers cooperated only in reaction to mission-critical issues (Kim et al., 2021).

To bridge these structural and cultural divides, the DevOps movement seeks to create one integrated technology organization that enables its business to rapidly innovate and adapt to the competitive landscape while providing stable, reliable, and secure service to the customer (Kim et al., 2021). Like the agile movement in 2001, the DevOps movement also called upon building the types of generative cultures that creative knowledge workers would love to work in, beginning by resolving the core, chronic conflict between Dev and Ops (Kim et al., 2021). To enable the DevOps transformations, the DevOps movement proposed using four balanced metrics to measure performance of the IT organization: Lead Time for Changes, Deployment Frequency, Change Failure Rate, and Mean-Time-to-Restore (Forsgren et al., 2018).

K. SOFTWARE ACQUISITION REFORM

The DoD has made numerous efforts to reform its software acquisition process and practices. In 1987, the DSB Task Force on Military Software recommended "aggressively looking for opportunities to buy…[commercial] tools, methods, environments, and application software" to establish parity with commercially driven digital technology development (Brooks et al., 1987, p. 2). The same report also recognized that specific metrics are necessary to detect, measure, and manage the health of software development projects, which should include program size, software complexity, personnel experience, testing progress, and incremental-release content (Brooks et al., 1987). In 2000, the DSB Task Force on Defense Software similarly recognized that certain core metrics are essential to identify and manage emergent problems in software development projects (Hansen & Nesbit, 2000). To standardize assessment of software project health, the Task Force on Defense Software recommended mandating the use of the following core metrics: (a) progress via earned value reporting, (b) staffing via tracking vacancies and turnovers, (c) requirements via tracking implementation and volatility rates, (d) quality via tracking defect and test completion rates, and (e) product stability via tracking inspection and rework rates (Hansen & Nesbit, 2000).

In 2009, the DSB proposed a new IT acquisition process inspired by commercial agile software development (Defense Science Board [DSB], 2009). Shown in Figure 13, the new IT acquisition process was designed to dynamically prioritize requirements based on operational impact, increase opportunities to adopt the latest available technology, and enable iterative, incremental delivery of capability in 18 months or less (DSB, 2009):



Milestone Build Decision
Materiel Development Decision
CDD
RELEASE 1
ICD
Business Case Analysis and Development
Architectural Development and Risk Reduction
Prototypes
Development & Demonstration
Iteration1
Iteration 2
Iteration "N"
Operations and Support
Coordinated DOD stakeholder involvement
Up to 2 years
Integrated DT / OT
6 to 18 months
RELEASE 2
Prototypes
Development & Demonstration
Iteration 1
Iteration 2
Iteration 3
Operations and Support
ICD Initial Capability Document
CDD Capabilities Development Document
Decision Point
RELEASE "N"
Prototypes
Development & Demonstration
Iteration 1
Iteration 2
Iteration 3
Operations and Support
Continuous Technology/Requirements Development & Maturation

Figure 13. An Iterative, Incremental IT Acquisition Process. Source: DSB (2009).

The DSB's 2009 report also noted that, in addition to building agility into software acquisition processes and practices, relevant metrics that could continuously track the timely, cost-effective delivery of required capabilities were needed (DSB, 2009). However, no specific metrics were recommended (DSB, 2009). By 2018, the DSB recognized that the way it conceives, designs, and manages software acquisition programs must change, stating that, "The classic acquisition metrics include cost, schedule, and performance. The classic phases of acquisition include development, production, and sustainment. However, modern software is in continuous development" (Office of the Under Secretary of Defense for Research and Engineering [OUSD(R&E)], 2018, pp. 20–21). Thus, measuring and managing software acquisition programs via on-time, on-cost, single delivery of software "creates a misalignment between the DoD's processes and the reality of contemporary best practices" (OUSD[R&E], 2018, p. 21). While the DSB's 2018 report acknowledged that each software acquisition program needed a program-appropriate framework, it recommended using sprint burn-down, epic and release burn-down, velocity, control, and/or cumulative flow diagram charts to help estimate the status of software deliveries (OUSD[R&E], 2018). In 2019, the DIB published its Software Acquisition and Practice (SWAP) study, the DoD's most comprehensive software acquisition reform study to date (DIB, 2019b). The DIB SWAP study highlighted software engineering industry best practices; DoD digital transformation case studies; several performance metrics and metrics standards based on the type of software and computing infrastructure involved; and it established multiple lines of effort to restructure how the DoD procures, develops, and manages software (DIB, 2019b). In 2020, the DoD then published Department of Defense Instruction (DoDI) 5000.87, *Operation of the Software Acquisition Pathway*, its newest software acquisition strategy (Office of the Under Secretary of Defense for Acquisition and Sustainment [OUSD(A&S)], 2020c). The Software Acquisition Pathway (SWP) envisions a continuous software acquisition process, as shown in Figure 14:

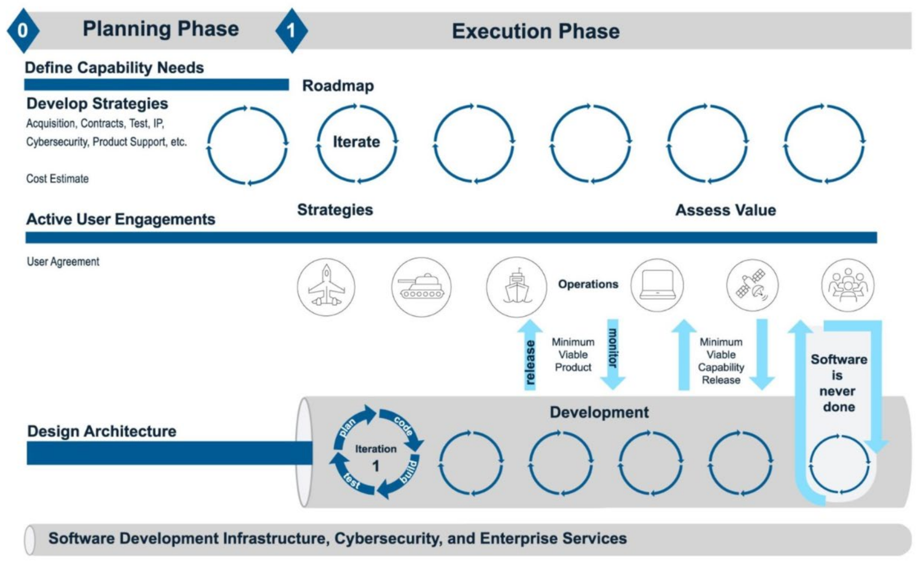


Figure 14. The DoD SWP's Continuous Software Acquisition Process. Source: OUSD(A&S, 2020c).

To streamline the acquisition of custom software, the DoD SWP is exempt from Joint Capabilities Integration and Development System (JCIDS) requirements, it encourages maximum use of existing enterprise software and tooling, it requires regular feedback from software users to validate and prioritize design choices, and it requires the first software delivery to occur no later than 1 year from the date development efforts are initially funded (OUSD[A&S], 2020c). To create a feedback loop with intended users and reinforce user-centered design (UCD), the DoD SWP also requires building and delivering a minimally engineered product (i.e., MVP; OUSD[A&S], 2020c). But most importantly, the DoD SWP requires regularly conducting Value Assessments (VA) with an operational sponsor to identify and assess the mission impact of software acquisition efforts, to identify and assess risks, as well as to inform future resourcing decisions (OUSD[A&S], 2022g).

L. CHALLENGES IN SOFTWARE ENGINEERING

Generally, most organizations have recognized the need to digitally transform (i.e., maximize their adoption of digital technology; Kersten, 2018). But while manufacturing and industrial management have benefitted from over 100 years of operational experience and are extremely mature in terms of performance management methods, metrics, and data collection tools, the software engineering industry lacks consensus on how to measure the software development process (Forsgen & Kersten, 2018). Relatively speaking, software engineering is a young field, and software engineering management is even less mature (Forsgren & Kersten, 2018). For instance, there is still no standard metric for software program size—lines of code (LOC), function points, and use case points are only some alternatives—so software projects estimate and manage tasks differently since no objective yardstick for complexity or effort yet exists (Efe & Demirörs, 2013). Moreover, even after 20 years, there are still no standard metrics to manage agile software engineering environments in commercial industry (Maddox & Walker, 2021).

As for the DoD, Development-Security-IT Operations (DevSecOps) provides SWP programs the technology infrastructure and practices that enable parallelization and automation of many development, certification, and deployment activities (O’Hearn, 2022), and it spawns a large amount of telemetry data that enables stakeholders to assess software delivery performance (Nichols et al., 2022). But while DevSecOps metrics provides rich insight into software development, they do not satisfy program control metrics for assessing progress, such as burn rates against spend targets, schedule for integration activities, and schedule for future releases (Nichols et al., 2022). Thus, SWP programs may outpace program control capabilities because DevSecOps produces working software at the end of each iteration/sprint, whereas the project management organization (PMO) needs up-to-date program control information to make new commitments, which can unfortunately take weeks or months to obtain (Nichols et al., 2022). Moreover, because the DoD is one of largest, most complex institutions in the world, the organizations and people operating within it are also highly susceptible to the "bureaucratic dysfunction of goal displacement: focusing on the rules rather than their ends" (Schwartz, 2020, p. 186), so program control metrics for SWP programs are not likely to change fast.

Despite these commercial and non-commercial software engineering management challenges, all leading software engineers agree: The purpose of digital transformation is to enhance an organization's ability to create and deliver business value, and the purpose of any metric is to inform and enable business decisions (Forsgen & Kersten, 2018) In other words, technology and metrics are simply a means to an end, whereas business value and business decisions are fundamentally context dependent. As for SWP programs, while the PM/product manager/product owner may need to collect and maintain at least two sets of metrics by necessity, it has always been crucial to bridge the gap between how the DoD and the individual services will conduct acquisition oversight and how the software development process will generate insight (Lapham et al., 2011).

M. SUMMARY

To provide context for the disparities between software engineering technical practice and defense management practice, this chapter included a review of project management, NPD, and software acquisition reform literature since the creation of the software engineering industry profession in the late 1960s. Early software engineering was heavily influenced by industrial management, which conceived and treated software development as a predictable manufacturing process that required specific inputs of money, time, and programming activity to output a desired result. Stage-gate NPD governance systems were also designed after manufacturing practices and reinforced planning rigor by standardizing the expected project inputs and decision rules at concentrated quality control checkpoints. While cross-functional teaming and concurrent design activity accelerated project teams' time to market performance, stage-gate governance limited larger organizations' ability to competitively adapt to market forces while sustaining a high-quality, cost-effective product portfolio.

Once the Internet became commercialized, marketing professionals became marginalized in the software engineering industry, going from directly planning and directing NPD decisions to advising them via marketing research reporting and committee reviews. To develop and capture increasingly diverse consumer software market segments, product managers embedded in engineering organizations and coordinated, managed, and executed product requirements throughout the NPD process. But given increased globalization and digitalization of commerce, more and more software engineers and marketing professionals struggled with outdated methodologies in an increasingly uncertain, complex, and fast-paced business environment.

After applying lean principles and values to the software development process, leading software engineers founded the agile software engineering movement, which largely moved from sequential to concurrent software engineering and small-batch requirements to enable accelerated SDLC iterations and continual adaptation to dynamic demand. Initially, PMs resisted agile methodology until they became accustomed to agile project design and dynamically planning and managing development tasks. Once agile principles and patterns were applied to the marketing discipline, the high-tech industry also founded Lean Startup, an NPD methodology that enabled new organizations to effectively build and deploy new products and services under conditions of extreme uncertainty. In combination, agile and Lean Startup substantially accelerated digital NPD, most notably for high-tech startups operating on the public cloud.

However, successfully incorporating agile and Lean Startup across larger enterprises required new mental models for capacity management, variability, and early development. Predominantly, this required making software WIP visible, limiting WIP in software engineering workflows, and recognizing that lean production practices were applicable only to software deployment activities, not design and development. Moreover, as cloud computing became commoditized and software engineering tooling became increasingly cloud-native, it became imperative for IT organizations to eliminate all silos between Dev and Ops, primarily by unifying their goals and incentives and deliberately building one highly agile, secure, and reliable system of software engineering work.

As for the DoD, its earliest software acquisition reform efforts recognized the need to keep parity with commercial industry's digital technology and development methodology. By 2020, however, the DoD formally recognized software development as a continuous process and issued a software acquisition strategy for custom application and embedded software built around agile methods. But while modern software engineering technology and methodology has significantly expanded the amount of performance data available to managers, the software engineering industry has not yet standardized any performance metrics for concurrent software engineering processes. Therefore, as DoD custom software development outpaces the defense acquisition governance process, it is critical to note that there is no one-size-fits-all tool kit to manage software development. Indeed, because software development is a continuous process, the optimal management tools and methods are entirely context dependent. The next chapter includes a discussion of measurements and metrics considerations for use in the DoD's SWP.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AGILE SOFTWARE ENGINEERING METRICS

This chapter includes a discussion of the most common measurements and metrics used in commercial agile software engineering environments and analysis of how to adapt them to the specific objectives of the SWP. To effectively implement agile measurements and metrics, the chapter also synopsizes several management principles and patterns from modern software engineering and digital NPD. For the sake of clarity and brevity throughout this chapter, *agile* means agile software engineering, *agile project* means a software development project that practices agile software engineering, *agile project management* means the discipline of managing software development projects that practice agile software engineering, *waterfall* means waterfall software engineering, *waterfall project* means a software development project that practices waterfall software engineering, and *waterfall project management* means the discipline of managing software development projects that practice waterfall software engineering.

A. BACKGROUND

Before discussing common agile metrics, it is necessary to distinguish between a *measurement* and *metric*. Measurement is the act of using an instrument to assign a value on a scale to an object or event (Kahneman et al., 2021). For instance, one measures the length of a carpet in inches using a tape measure, and one measures the temperature in degrees Fahrenheit or Celsius by consulting a thermometer (Kahneman et al., 2021). The purpose of a measurement is to obtain an accurate and precise value—"to approach truth and minimize error" (Kahneman et al., 2021, p. 39). Measurements provide simple, readily obtained facts. On the other hand, a metric is "the measurement portion of a control system" (Reinertsen, 2009, p. 222). Metrics consist of measurements, but not all measurements should be metrics. The purpose of a metric is to "induce the departments to do what is good for the company as a whole" (Goldratt, 1997, p. 107), for example, to inform business decisions and/or accomplish a business objective. As such, because metrics function as incentives and involve managerial judgment, they must be used with great care and close attention. Having defined these terms, this section now highlights how waterfall and agile software development approach metrics differently.

In waterfall projects, all stakeholder needs are identified and thoroughly analyzed, a comprehensive solution is designed, risks are identified and assessed, and a full budget is estimated and allocated in advance (Nicolette, 2015). Thus, a complete plan is formulated at the start of the development project. On the other hand, in agile projects, a desired future end state is envisioned; only short-term analysis is performed to initiate the engineering process; and teams collaboratively explore the problem/solution space through iterative development, incremental delivery, and frequent market feedback (Nicolette, 2015). So, while the design is highly uncertain in agile projects, the development team discovers and implements it through an adaptive process.

As noted, development project plans are merely a proxy for the intended product (Perri, 2018). Generally, metrics also compare expected versus actual levels of performance (Nicolette, 2015). From the perspective of the development project, however, waterfall project plan expectations are comprehensively defined in the past, even when plans are re-baselined (Nicolette, 2015). As such, as waterfall projects progress, they always look to the past to identify performance goals, maintain alignment to them, and to measure their success (Nicolette, 2015). Thus, waterfall projects tend to rely on backward-facing metrics, as shown in Figure 15 (Nicolette, 2015):

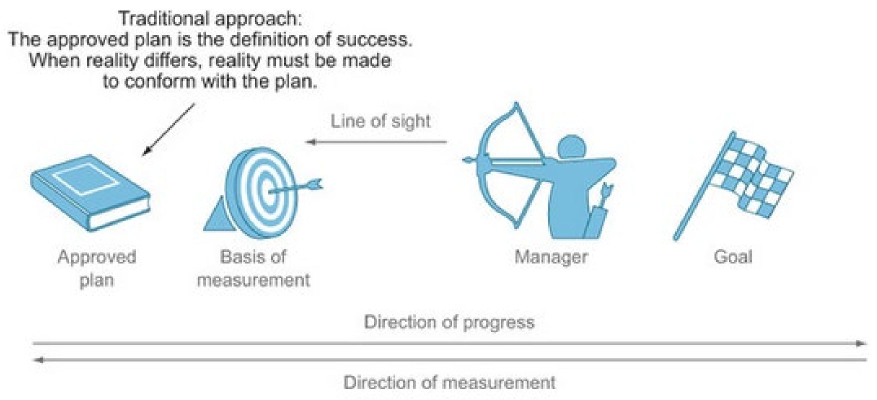


Figure 15. Traditional Project Metrics. Source: Nicolette (2015).

However, within agile projects, teams define performance expectations based on their current understanding of the desired future end state (Nicolette, 2015). Naturally, the team's understanding evolves from each iteration/sprint to the next (Nicolette, 2015). Thus, as agile projects progress, they continually look to the future to identify performance goals, maintain alignment to them, and to measure success (Nicolette, 2015). As a result, to cope with dynamic performance expectations, agile projects tend to rely on forward-facing metrics, as shown in Figure 16 (Nicolette, 2015):

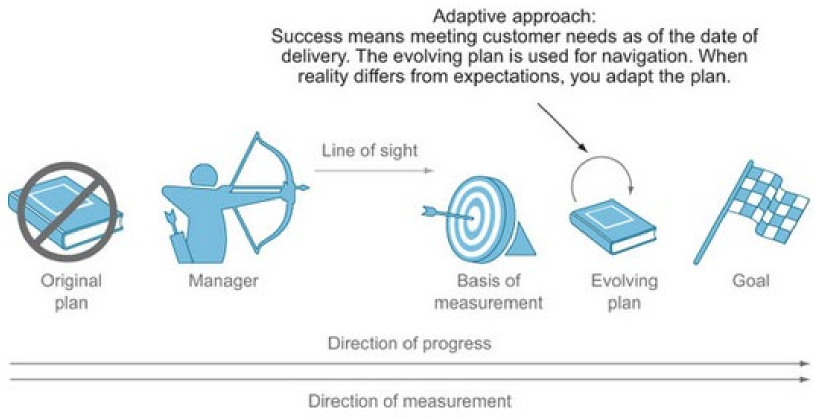

Adaptive approach:
Success means meeting customer needs as of the date of delivery. The evolving plan is used for navigation. When reality differs from expectations, you adapt the plan.
Line of sight
Original plan
Manager
Basis of measurement
Evolving plan
Goal
Direction of progress
Direction of measurement

Figure 16. Agile Project Metrics. Source: Nicolette (2015).

Additionally, as noted, waterfall projects are designed with fixed scope and flexible cost and schedule constraints, whereas agile projects are designed using a series of mini projects, each of which has flexible scope and fixed cost and schedule constraints (Patel, 2021). Now that the salient differences between waterfall and agile metrics and project design have been identified, the researcher identifies the most common commercial practices in agile planning and metrics.

In most schools of agile software engineering, commercial business problems are identified and broken down to represent a basic unit of customer need called *stories* (Forsgren et al., 2018**)**. From there, development teams estimate and assign the relative level of effort (LOE) needed to implement and ship each story by using a scale of “story points” (SP; Forsgren et al., 2018**)**. SPs are an abstract measure for what it takes to realize a story in current and/or future iterations (Rawsthorne, 2008). To this end, instead of measuring how “long” the story is, SPs measure the difficulty of each story relative to others by asking a question like, “Given good code to work with, how difficult is this story for our team?” (Rawsthorne, 2008). Thus, agile development teams use SPs for relative sizing, and because SPs are highly contextual, each team tends to use them differently.

Because agile environments commonly plan work in iterations/sprints, or fixed-time intervals, many agile development teams also measure their *velocity*, or iteration throughput (Forsgren et al., 2018**)**. Velocity is a simple enough metric to calculate. After completing each iteration, the team counts the total number of approved SPs to determine its iteration velocity (i.e., velocity). After a few iterations, teams also begin tracking their average number of SPs completed per iteration, or average velocity (Forsgren et al., 2018**)**. Agile teams internally calculate velocity for the sake of tracking their productivity. For instance, if the team has been completing 30 SPs per iteration on average, it can plan its work accordingly in future iterations. Additionally, by using SPs and their average velocity, teams could roughly extrapolate by when they could expect to complete all planned and estimated outstanding project work (Forsgren et al., 2018**)**. But while estimating and tracking SPs and velocity have become common agile practices, their utility is limited to individual development teams and the teams' unique projects; additionally, these practices are primarily used for the purpose of capacity planning and management. That said, depending on the software development phase or objective, there are many other commonly used agile software engineering measurements and metrics.

Once the agile movement gained a critical mass of followers in the software engineering industry, software developers and their managers realized that at least three classes of metrics are necessary in any agile environment: (a) code-level metrics such as code quality, (b) productivity metrics such as velocity, and (c) economic metrics such as business value (BV) (Oza & Korkala, 2012). For instance, code-level metrics provide a wealth of insight into software development, but they seldom account for the customer in a manner that drives product decisions (Oza & Korkala, 2012). On the other hand, while productivity and economic metrics more readily support product decision-making, they often heavily rely on judgment and/or tacit knowledge due to the fast-paced, dynamic nature of agile software development (Oza & Korkala, 2012). In any case, using a combination of code-level, productivity, and economic metrics became a best practice to drive business decisions in agile environments (Oza & Korkala, 2012).

However, to effectively align, motivate, and focus software development teams, agile practitioners also realized that agile metrics must clearly and consistently connect daily work to the organization's current goals (Oza & Korkala, 2012). Naturally, using an overly complex set of metrics inhibits, rather than enhances, sustained mission focus. To this end, some agile practitioners argued for using only one performance metric—BV—and subordinating all code-level, productivity, and any other metrics to this economic metric (Hartmann & Dymond, 2006). According to this management approach, code-level, productivity, and other classes of metrics and measurements should be considered "diagnostics," for BV is the only viable metric to measure progress and assess performance (Hartmann & Dymond, 2006). Overall, these agile metric practices seem sensible to apply to DON SWP programs: a combination of code-level, productivity, and economic metrics provides a robust set of quantitative and qualitative information, and ensuring the preferred economic metric—BV—takes always takes precedence helps drive and sustain focus on operational outcomes.

Before addressing SWP program execution, this section addresses the conditions in which agile methods are most effective. All things being equal, agile software engineering is not an inherently faster development methodology than waterfall software engineering (Tate & Bailey, 2022). The practical value of practicing agile methods is that small batches of work and fast feedback improve the ability to rapidly adapt to an uncertain, complex, fast-moving business environment (Smith & Reinertsen, 1997). To develop this ability, however, the DON SWP program's culture must always prioritize speed to all throughout the software development process (DIB, 2019b).

Time to market, which is defined as the total time from which a new DON SWP program is launched to its first deployment of useful capability, is the best measurement to instill an agile mindset because it determines the first point at which new software creates real market value and captures user feedback to iteratively shape future capability delivery (DIB, 2019a). Time to market captures more than programming performance—it also measures the time necessary to satisfy critical regulatory requirements for release approval, which is often the most time-consuming part of delivering new defense software capabilities (DIB, 2019a). Again, agile methods are not necessarily faster than waterfall methods. However, by doing less total work upfront, agile methods enable organizations to push code to market much sooner, enabling them to continually develop and deliver new and/or improved software capability based on rapid feedback and day-to-day product use.

Thus, DON SWP programs must establish urgency to put working software into the user's hands as soon as practicable. To drive the mindset for market speed in all SWP programs, the DoD SWP requires delivery of operationally useful capability (i.e., the MVCR) no later than one year from the date software development is first funded (OUSD[A&S], 2020c). No later than one year after funding development is therefore the time to market standard for all new DoD SWP programs (OUSD[A&S], 2020c). Having established an overview of agile practices, this discussion now turns to adopting them to execute the SWP.

B. PRE–MINIMUM VIABLE CAPABILITY RELEASE

The set of technologies required to build and operate a software system, from the operating system platform to the mission application, is called the technology stack, or tech stack, in the software engineering industry (Hering, 2018). The tech stack determines the type of systems that can be built, the level of customization possible, as well as the computing resources needed to develop, deploy, and operate software (Hering, 2018). To recommend the appropriate metrics, the researcher assumes that DON SWP programs are acquiring custom application software running on commercial hardware/operating systems, or Type C software as defined in the DIB SWAP study (DIB, 2019b). Additionally, the researcher assumes that the custom application software is developed, assured, deployed, and supported primarily by contractors.

The DoD SWP seeks to adopt commercial software engineering and digital NPD best practices to the greatest extent practicable, and its explicit intent is to "facilitate rapid and iterative delivery of [custom] software capability to the user" (OUSD[A&S], n.d.-f). To that end, the DoD SWP exempts all SWP programs from JCIDS requirements, encourages leveraging existing enterprise IT services as much as possible, and requires building and demonstrating an MVP to the customer, user, or designated representative as soon as possible to create the feedback loops that enable agility and reinforce UCD (OUSD[A&S], 2022b; OUSD[A&S], n.d.-d).

The DoD SWP has a *Planning Phase* an *Execution Phase*, as shown in Figure 17.

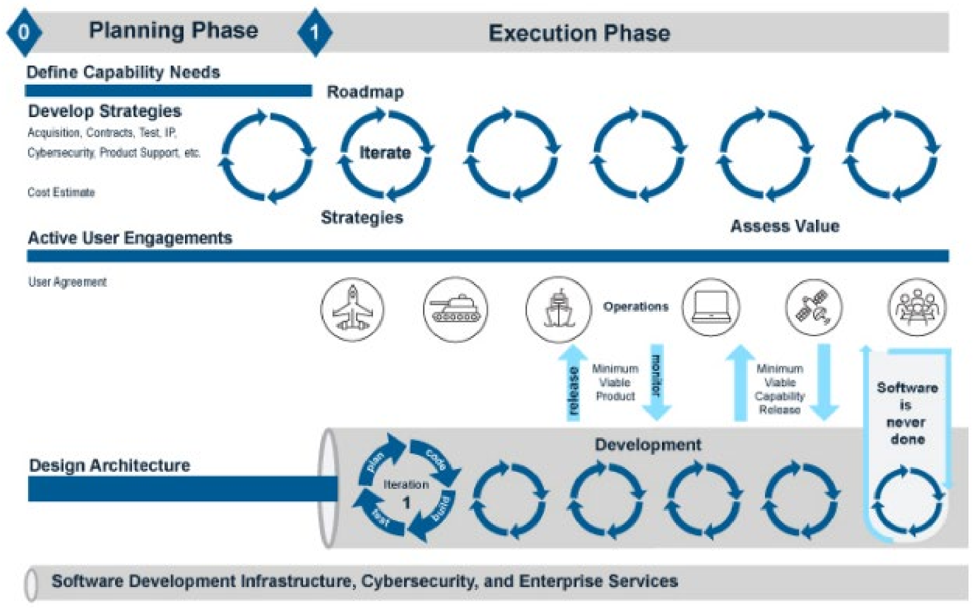


Figure 17. The DoD SWP's Planning and Execution Phases. Source: OUSD[A&S] (2020c).

The Planning Phase begins as soon as the Decision Authority (DA) has reviewed the draft Capabilities Need Statement (CNS) and signed the Acquisition Decision Memorandum (ADM) authorizing use of the DoD SWP (OUSD[A&S], 2020c). The purpose of the Planning Phase is to learn the user's capability needs and to plan the approach to develop and deliver the software that satisfies those needs (OUSD[A&S], 2020c). Planning Phase key artifacts include the CNS; User Agreement; Program Strategies, including acquisition, contracting, intellectual property, test, cybersecurity, and product support strategies; and Cost Estimate (OUSD[A&S], 2020c). According to the DoD Adaptive Acquisition Framework (AAF), given the strategic intent to move fast, all SWP programs are expected to spend 60–180 days in the Planning Phase, depending on their context and acquisition complexity (OUSD[A&S], n.d.-a). Once these key artifacts are sufficiently developed, SWP programs enter the Execution Phase, the purpose of which is to rapidly and iteratively design, develop, deliver, and operate high quality software capabilities that meet the users' highest priority needs (OUSD[A&S], 2020c).

To do this, SWP programs build two key artifacts through active user engagement, UCD, and agile planning: the Product Roadmap and Program Backlogs

(OUSD[A&S], n.d.-c). The Product Roadmap is a high-level, time-phased map that shows projected software capability deliveries, and the Program Backlogs function as a dynamic ledger to identify, plan, prioritize, and allocate near-term software development work (OUSD[A&S], 2020c). According to the AAF SWP guidance, the CNS might cover a Future Years Defense Program horizon, but the Product Roadmap should cover the next 12–18 months, whereas the Program Backlogs should cover nearer-term release requirements that are continuously re-prioritized (OUSD[A&S], n.d.-a). The Product Roadmap and Program Backlogs also complement each other, for the former provides an enterprise view of the product vision, the latter provides team-level context of unmet software development requirements, and SWP programs iteratively and continuously build and refine these artifacts via active user engagement, UCD, and agile planning throughout the Execution Phase, as shown in Figures 18 and 19.

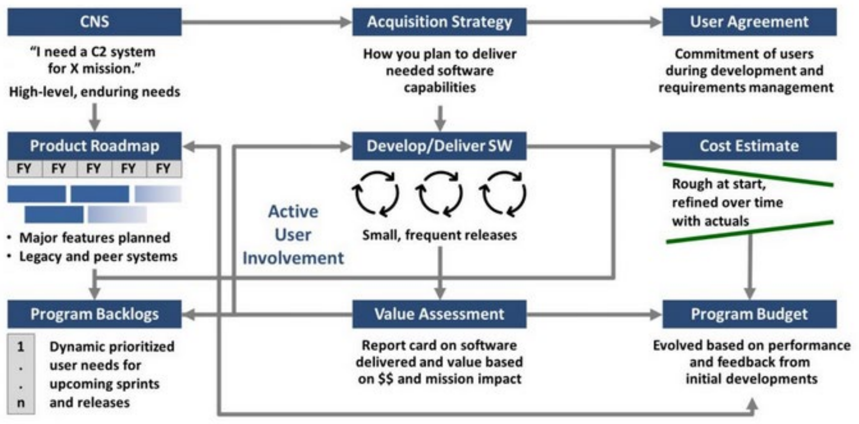

CNS
"I need a C2 system for X mission."
High-level, enduring needs
Acquisition Strategy
How you plan to deliver needed software capabilities
User Agreement
Commitment of users during development and requirements management
Product Roadmap
FY FY FY FY FY
• Major features planned
• Legacy and peer systems
Active User Involvement
Develop/Deliver SW
Small, frequent releases
Cost Estimate
Rough at start, refined over time with actuals
Program Backlogs
1 . . n
Dynamic prioritized user needs for upcoming sprints and releases
Value Assessment
Report card on software delivered and value based on $$ and mission impact
Program Budget
Evolved based on performance and feedback from initial developments

Figure 18. Planning, Acquiring, and Assessing Capability Needs in the DoD SWP. Source: OUSD(A&S, n.d.-c).

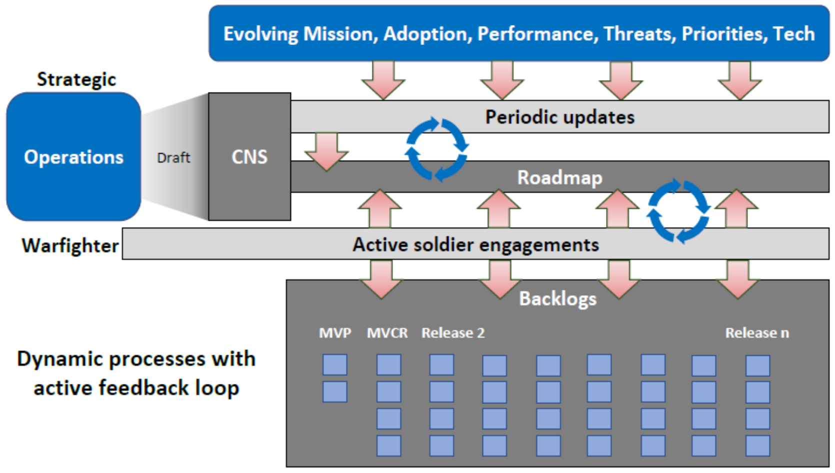

Evolving Mission, Adoption, Performance, Threats, Priorities, Tech
Strategic
Operations
Draft
CNS
Periodic updates
Roadmap
Warfighter
Active soldier engagements
Backlogs
MVP
MVCR
Release 2
Release n
Dynamic processes with active feedback loop

Figure 19. Dynamically Planning, Prioritizing, and Implementing Software Development Requirements via User Community Engagement. Source: OUSD(A&S, n.d.-b).

To distinguish high priority work items from lower priority requirements in the queue, AAF guidance suggests that SWP programs structure their Program Backlogs as shown in Figure 20 (OUSD[A&S], 2022):

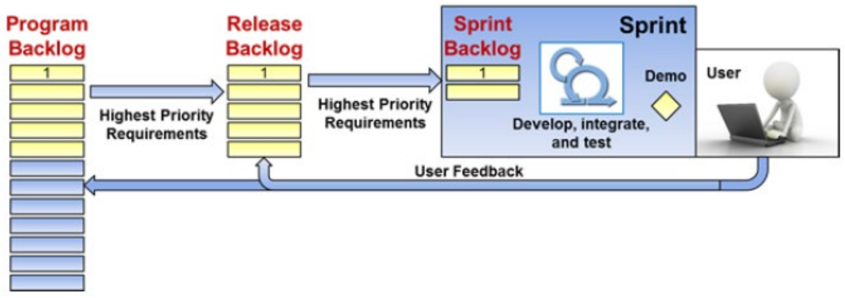

Program Backlog
1
Highest Priority Requirements
Release Backlog
1
Highest Priority Requirements
Sprint Backlog
1
Sprint
Demo
Develop, integrate, and test
User
User Feedback

Figure 20. Managing Software Development Requirements via Program Backlogs. Source: OUSD(A&S, 2022b).

To design, implement, and deliver the highest priority needs, the PM/product manager/product owner allocates software development requirements to upcoming iterations/sprints through the DoD SWP program's contracts as shown in Figure 21:

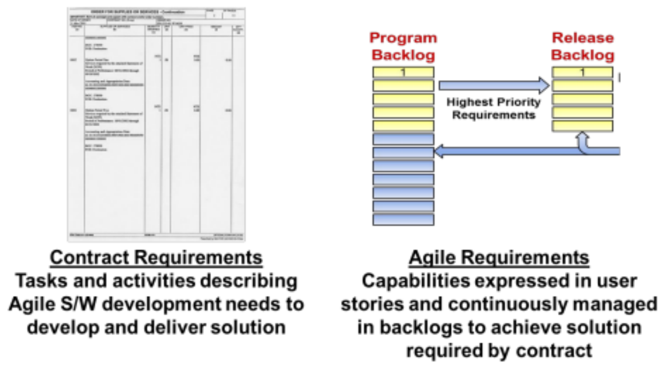

Program Backlog
Release Backlog
1
1
Highest Priority Requirements
Contract Requirements
Tasks and activities describing Agile S/W development needs to develop and deliver solution
Agile Requirements
Capabilities expressed in user stories and continuously managed in backlogs to achieve solution required by contract

Figure 21. Capacity-Based Contracting. Source: OUSD(A&S, 2019).

Finally, to help improve the DoD SWP, the OUSD(A&S; n.d.-g) requires each SWP program to semiannually report its performance under 12 metrics, shown in Table 1:

Table 1. DoD SWP Semiannual Reporting Metrics. Adapted from OUSD(A&S; n.d.-g).

ID	Metric	Definition
(a)	Average Lead time for Authority to Operate (ATO)	"Average number of days to obtain [ATO] by release" (OUSD[A&S], n.d.-g)
(b)	Continuous ATO In-Place	"Indicator of program's ability to achieve a continuous [ATO] or similarly expedited [Approving Official] approval process" (OUSD[A&S], n.d.-g)
(c)	Mean Time to Resolve Experienced Cyber Incident or Common Vulnerability or Exposure (CVE)	"The mean response time a program was able to resolve a Cyber Incident or [CVE] from the time of identification through resolution" (OUSD[A&S], n.d.-g)
(d)	Mean Time to Detect Cyber Incident	"The mean time from Cyber Incident start to time of identification" (OUSD[A&S], n.d.-g)
(e)	Average Deployment Frequency	"The average frequency of releases into an operational environment" (OUSD[A&S], n.d.-g)
(f)	Average Cycle Time	"The average duration time to deliver a capability or feature into operation, measured from the time the need is identified for a specific build (moved from the backlog to a planned release) to the time the code is committed (development activity finished)" (OUSD[A&S], n.d.-g)
(g)	Average Lead Time for Change	"The average duration to deliver a capability or feature into operation, measured from the time the code is committed (development activity finished) to the time it is available for release to operations (production)" (OUSD[A&S], n.d.-g)
(h)	Minimum Lead Time for Change	"The minimum duration to deliver a capability or feature into operation, measured from the time the code is committed (development activity finished) to the time it is available for release to operations (production)" (OUSD[A&S], n.d.-g)
(i)	Maximum Lead Time for Change	"The maximum duration to deliver a capability or feature into operation, measured from the time the code is committed (development activity finished) to the time it is available for release to operations (production)" (OUSD[A&S], n.d.-g)
(j)	Change Fail Rate	"The percentage of releases to the production/operational environment that requires subsequent remediation" (OUSD[A&S], n.d.-g)
(k)	Mean Time to Restore (MTTR)	"The mean time to restore the system in response to a downtime event or a defect that requires subsequent remediation" (OUSD[A&S], n.d.-g)
(l)	Value Assessment (VA) Rating	"The [PMO's] perceived rating based on the last feedback received from the operational sponsor" (OUSD[A&S], n.d.-g)

Unfortunately, there are some challenges with the DoD SWP's semiannual reporting metrics. First, the requirement to track and semiannually report these 12 metrics applies as soon as software acquisition programs enter the DoD SWP (OUSD[A&S], n.d.-g), even though Average Deployment Frequency, Average Cycle Time, Average Lead Time for Change, Minimum Lead Time for Change, Maximum Lead Time for Change, Change Failure Rate, and MTTR measure deployment and response rate for an operationalized software system. Given that accomplishing the MVCR may take up to 1 year after first funding development, imposing these metrics creates an unnecessary burden during the Planning Phase and early Execution Phase of each SWP program. Second, since the first VA cycle will not close out until well into the post-MVCR phase (OUSD[A&S], 2020c), prematurely tracking and reporting VA Rating creates similar waste.

DON SWP programs are also required to have a DA-approved metrics approach at the time of entering the DON SWP, to prioritize automation in collecting and reporting metrics, to adopt metrics reviews that become part of the VA process, and to use, at minimum, the following four metrics: "(1) Average Deployment Frequency; (2) Average and Minimum/Maximum Lead Time to commit code to production; (3) Average Cycle Time; (4) Change Failure Rate" (ASN[RD&A]), 2022, pp. 6–7). By requiring maximum use of automated telemetry and incorporating software delivery performance metrics from the DevOps movement, the DON SWP—as well as the DoD SWP, given their overlapping metrics—has attempted to adopt state-of-the-art software delivery practices (Kim et al., 2021). However, there are also challenges with the DON SWP's required metrics.

First, until DON SWP programs gain the authority to operate (ATO), satisfy operational acceptance requirements, and attain any other required certifications, they will be unable to deploy code to operations (i.e., production environments; Tate & Bailey, 2022). The DON SWP's required metrics are applicable only when DON SWP programs have both developed the technical capability and obtained the appropriate authorities to deploy code to production environments, which is no earlier than deployment of the Minimum Viable Capability Release (MVCR). Second, while metric reviews are required to become part of the VA process, the first VA will not be conducted until after the software system is fielded, which both the DoD SWP and DON SWP anticipate can take up to a year from the date development activities are first funded (OUSD[A&S], 2020c; ASN[RD&A], 2022). As such, the DON's SWP provides no metrics or metric guidance to manage the software development process from the onset of the Execution Phase through deployment of the MVCR (i.e., the pre-MVCR phase).

That said, there are some potential strategies for DON SWP programs to manage progress in the pre-MVCR phase while attempting to comply with the DON SWP. According to the DON SWP, programs must accomplish their first release, or MVCR, no later than 1 year, or 52 weeks, from first funding development. Thus, according to the AAF's suggested approach for structuring Program Backlogs, the initial Release Backlog covers a period of 52 weeks. Assuming the DON SWP program uses a fixed time interval for each iteration/sprint, it could divide 52 weeks by this fixed time interval to calculate the anticipated number of iterations/sprints necessary to implement all initial Release Backlog requirements. For instance, if the DON SWP program uses 2-week intervals for each iteration/sprint, then there are 52 / 2 = 26 iterations/sprints available to complete the DON SWP program's initial Release Backlog requirements. Knowing this, the DON SWP program can then use velocity and velocity-based metrics to track and assess productivity for each development team during the pre-MVCR phase. Because the metric is so commonly used in agile environments, the PM/product manager/product owner can also automatically collect and track velocity-based information through ALM tools (DIB, 2019b). Velocity-based metrics could therefore help efficiently and effectively measure and manage productivity during the pre-MVCR phase.

Moving on, one technique that was developed to track and assess progress within agile software projects is Earned Business Value (EBV; Rawsthorne, 2006). DON SWP programs are advised to use EBV to measure progress towards building and demonstrating their MVP to users in a testing environment. To apply this method, the DON SWP program first organizes all initial software development work into a custom Work Breakdown Structure (WBS) that uses three legs: (a) product: work to build the MVP, (b) platform: work that enables building the MVP, and (c) deployment: work that enables demonstrating the MVP (Rawsthorne, 2006). For example, the notional WBS for a custom application development project would be structured as shown in Figure 22:

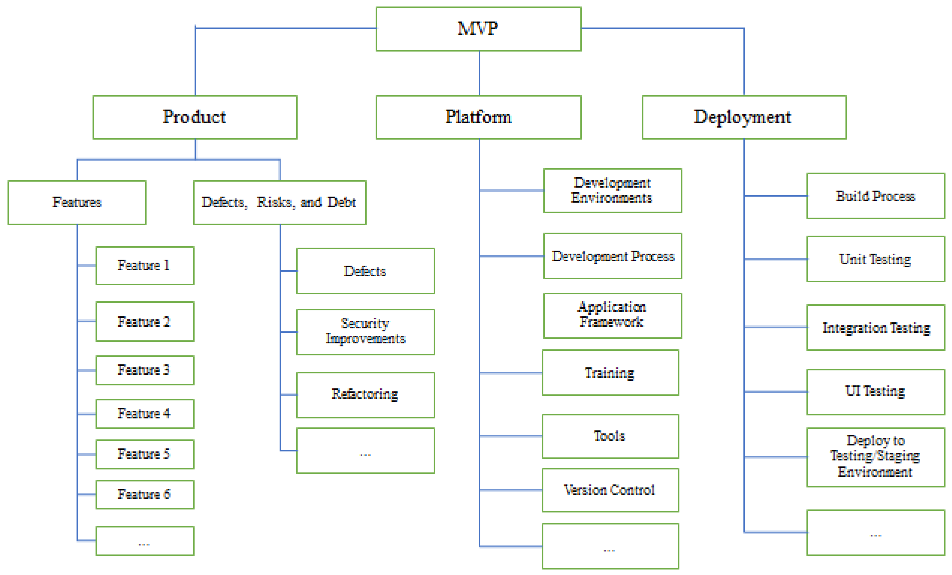


Figure 22. Notional WBS. Adapted from Rawsthorne (2006).

Next, the government PM/product manager/product owner assigns relative weights to each WBS leg and bucket, whereby only completing Product leg features and Deployment leg tasks can generate BV (Rawsthorne, 2006). Once BV weights have been assigned to the relevant legs and buckets, the notional WBS would appear as in Figure 23:

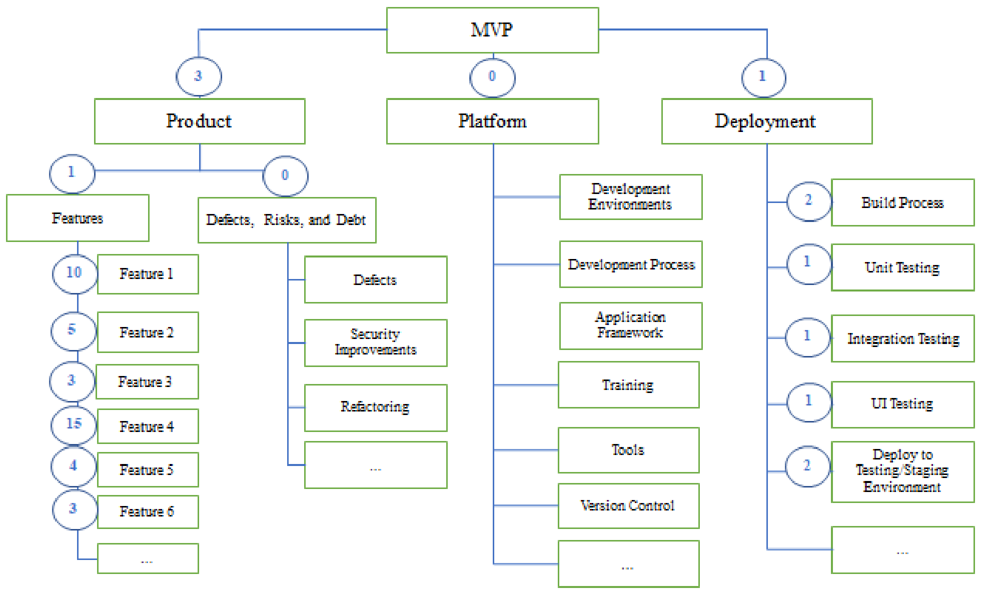
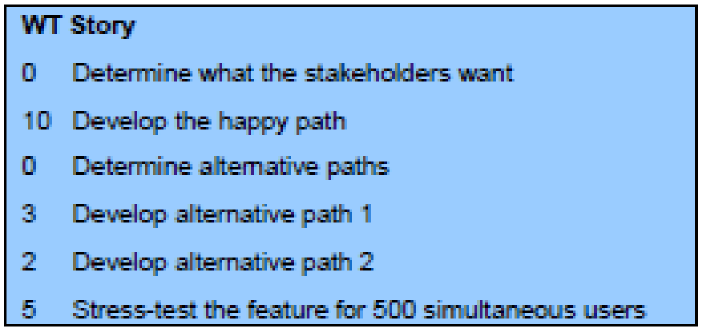


Figure 23. Notional, Weighted WBS. Adapted from Rawsthorne (2006).

Finally, additive weights are assigned to the stories for each Product feature (Rawsthorne, 2006). For instance, Figure 24 shows the notional stories for Feature 1:



WT	Story
0	Determine what the stakeholders want
10	Develop the happy path
0	Determine alternative paths
3	Develop alternative path 1
2	Develop alternative path 2
5	Stress-test the feature for 500 simultaneous users

Figure 24. Notional Stories for Feature 1. Source: Rawsthorne (2006).

Having mapped all initially known work, EBV then calculates the cumulative percentage of all implemented BV as DON SWP programs progress towards their MVP (Rawsthorne, 2006). Specifically, EBV enables DON SWP programs to track and assess their initial progress via the formula shown in Figure 25 (Rawsthorne, 2006):

$$EBV(\,Project) = \sum_{Completed} BV(\,Story)$$

Figure 25. The EBV Formula. Source: Rawsthorne (2006).

EBV is recursively calculated based on multiplying weight percentages of weight percentages down to the lowest WBS element (Rawsthorne, 2006). For instance, the calculations in Figure 26 evaluate the notional project's EBV once the first two stories for Feature 1 have been completed:

$$
\begin{aligned}
&EBV \\
&= \frac{MVP}{MVP} \times \frac{Product}{Product + Platform + Deployment} \times \frac{Features}{Features + Defects, Risks, and Debt} \times \\
&\frac{Feature\ 1}{Feature\ 1 + Feature\ 2 + \ldots Feature\ 6} \times \frac{Story\ 1 + Story\ 2}{Story\ 1 + Story\ 2 + \ldots Story\ 6} \\
&= \frac{1}{1} \times \frac{3}{3+0+1} \times \frac{1}{1+0} \times \frac{10}{10+5+\ldots 3} \times \frac{0+10}{0+10+\ldots 5} \\
&= 1 \times \frac{3}{4} \times 1 \times \frac{10}{40} \times \frac{10}{20} \\
&= 9.38\%
\end{aligned}
$$

Figure 26. Calculating a Project’s EBV. Source: Rawsthorne (2006).

In other words, after completing these two Feature 1 stories, the MVP can be considered 9.38% complete. While the government PM/product manager/product owner’s assigning relative weights to WBS elements is subjective, this method ensures all work to build and demonstrate the MVP to users in a testing environment is weighed and prioritized in alignment with the highest operational priorities (Rawsthorne, 2006).

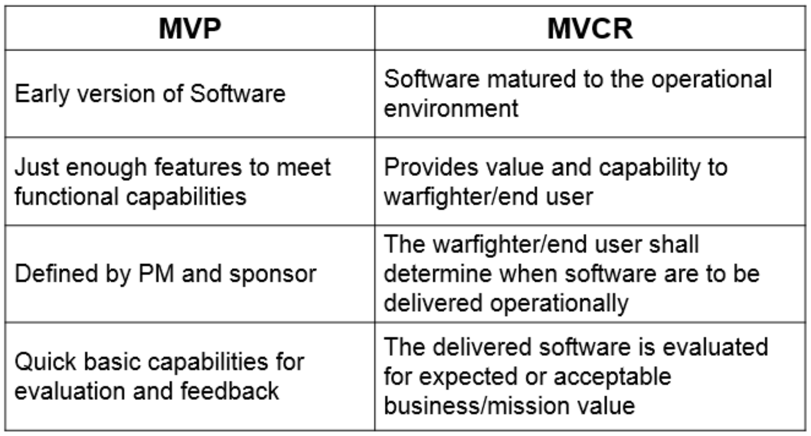
Overall, EBV has several practical benefits for measuring and managing progress. First, because EBV is calculated using unitless percentages, it is not biased towards dollars, duration, man-hours, effort, or any other specific unit of measure (Norton, 2020). Second, EBV does not use cost metrics, so it enables estimating and measuring project progress in a manner that creates no cost accounting overhead, unlike EVM. Third, EBV’s product oriented WBS and relative weighting scheme can help create and maintain a holistic view of project progress, including the relative BV of all essential software development work. Ultimately, EBV is intended for use only on one software deployment comprised of multiple iterations/sprints (Rawsthorne, 2006). Given that it helps organize all initially known work and enables measuring the value of said work in a simple, mission-focused manner, EBV is a practical tool that enables DON SWP programs to track and assess progress towards their MVP. While DON SWP programs will be able to automatically track work item completion using ALM tools, configuring, calculating, and updating EBV itself may have to be done manually, given that the WBS will be completely tailored to each Type C DON SWP program’s custom application MVP.

With respect to quality, DON SWP programs can readily track the rate and count of defects using ALM and/or software development tools (Norton, 2020). Naturally, maximizing quality should always be a desired outcome. That said, based on leading agile practices, defects should be treated as a natural byproduct of software development work, whereby instead of impractically seeking to avoid defects, the rate and count of defects are used to understand and manage the quality of the software development process (Harrison & Lively, 2019). To detect and contain excessive quantities and/or poor trends of defects, one useful metric is the ratio of defects created/work units completed in a set period (e.g., per software development iteration/sprint; Norton, 2020). Ideally, the lower this ratio is over time, the better.

As for tracking quality of the codebase, one useful metric is the percentage of automated unit testing for all implemented code (i.e., test coverage; Norton, 2020). While test coverage itself is not a measure of quality, having test coverage is an indicator of well-designed code because unit tests—tests that validate a method's behavior without calling any other method—are possible only when the code is constructed in a way that indicates loose dependencies or loose coupling (Norton, 2020). Thus, the amount of code successfully covered by unit tests can indicate how well the code is written (Norton, 2020), and based on the DIB SWAP study's *Metrics for Software Development* supplement, the target test coverage rate for Type C DON SWP programs is 90% (DIB, 2019a). To manage software development process and code quality during the implementation of the initial Release Backlog, DON SWP programs should therefore track ratio of defects created/work units completed in a set period and test coverage, respectively.

But while these metrics, velocity-based metrics, and EBV can help respectively track quality, productivity, and economic performance enroute to the MVP, reviewing these metrics does not necessarily support the VA process in the post-MVCR phase. The purpose of the VA is to learn, from the end user's perspective, how much short-term and long-term mission impact the delivered software has (i.e., assess the delivered software's BV; OUSD[A&S], 2020c), and the DON SWP requires that "metrics review should become part of the [VA] process" (ASN[RD&A], 2022, p. 6). In other words, the way DON SWP programs review metrics in the pre-MVCR phase should help them prepare to conduct VAs in the post-MVCR phase. Naturally, DON SWP programs are externally constrained by ATO and other release requirements during the pre-MVCR phase, so delivery to operations is not applicable. That said, based on agile best practices, the best times to measure BV are either when demonstrating completed features to the user(s) at the end of each iteration/sprint or after releasing completed features to production, with release to production being most preferred (Hartmann & Dymond, 2006). Fortunately, showing the MVP creates opportunities to conduct metrics review with the user, so DON SWP programs should use EBV to track progress until their MVP demonstration event.

The MVP demonstration is an important milestone in the pre-MVCR phase for several reasons. First, MVPs are minimally engineered products that begin the process of iterating and retesting (Ries, 2017). Thus, the MVP is a working prototype of the intended product. Second, the MVP demonstration helps understand what the user actually wants, not what DON SWP programs think the user wants (Ries, 2017). Third, the MVP demonstration creates a feedback loop around the user's highest priority operational needs, which is critical to make the most of the MVCR's impact and that of all subsequent software releases (OUSD[A&S], n.d.-e). Fourth, while the MVCR creates mission effects for the user, the MVP demonstration shows what the custom application does to enable those mission effects. The relationship between the MVP and MVCR is shown in Figures 27 and 28.



MVP	MVCR
Early version of Software	Software matured to the operational environment
Just enough features to meet functional capabilities	Provides value and capability to warfighter/end user
Defined by PM and sponsor	The warfighter/end user shall determine when software are to be delivered operationally
Quick basic capabilities for evaluation and feedback	The delivered software is evaluated for expected or acceptable business/mission value

Figure 27. Comparing the MVP and MVCR. Source: Garrison (2022).

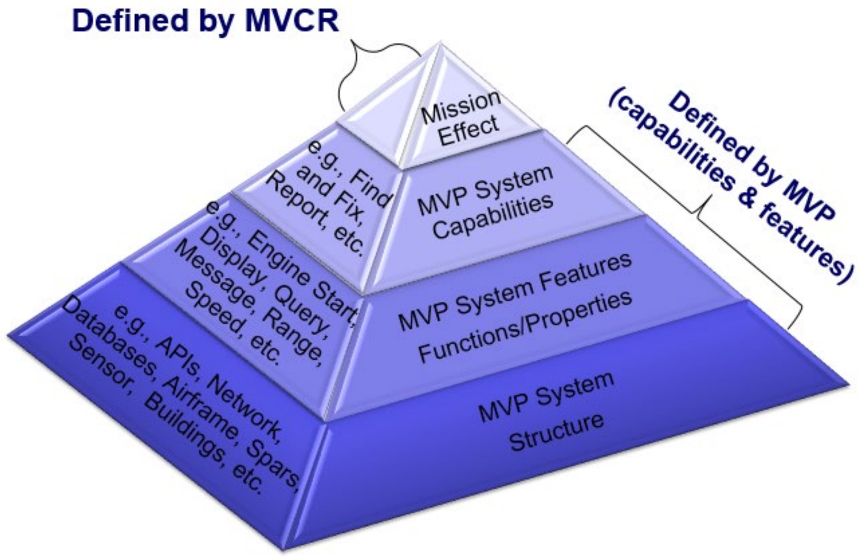

Defined by MVCR
Mission Effect
Defined by MVP (capabilities & features)
e.g., Find and Fix, Report, etc.
MVP System Capabilities
e.g., Engine Start, Display, Query, Message, Range, Speed, etc.
MVP System Features Functions/Properties
e.g., APIs, Network, Databases, Airframe, Spars, Sensor, Buildings, etc.
MVP System Structure

Figure 28. Representing the System MVP and MVCR. Source: Garrison (2022).

Second, neither the DoD nor the DON SWP prescribed MVP metrics because they intended the MVP review process to be based on customer and/or end user feedback (OUSD[A&S], 2020c; ASN[RD&A], 2022). Thus, given that the purpose of the VA is for the customer and/or end user to assess BV, DON SWP programs could use the MVP review process to help prepare for the VA process in the post-MVCR phase.

Now, to make the most of its prototype and its subsequent feedback loop, the DON SWP program should create and demonstrate its MVP in a production-like testing environment as soon as practicable. While this timeline will vary from one DON SWP program to another, assume, for illustrative purposes, that this becomes possible during Iteration/Sprint 20. Then, to align to agile best practices, the DON SWP program should demonstrate the MVP to the user immediately after Iteration/Sprint 20, newly completed functionality immediately after Iteration/Sprint 21, and so on through delivery of the MVCR, ideally during Iteration/Sprint 26. Once a DON SWP program has demonstrated the MVP, it should forgo EBV and graduate to tracking and assessing product-oriented metrics through delivery of the MVCR. According to the DoD's *Agile Metrics Guide: Strategy Considerations and Sample Metrics for Agile Development Solutions*, the most effective product-oriented metrics are the number and/or percentage of accepted user stories/features (OUSD[A&S], 2020b).

Tracking and updating product-oriented metrics in this manner enables DON SWP programs to align themselves to agile best practices; establish a pre-MVCR metrics review process that becomes part of the VA process in the post-MVR phase, as required by the DON SWP (ASN[RD&A], 2022); and, most importantly, ensure that DON SWP programs deliver the highest-priority software capability to operations in their MVCR. Because user stories/features are automatically tracked in the Program Backlogs, the data to track these product-oriented metrics are also automated and readily available.

All in all, to effectively manage contractor-led development of Type C software in the pre-MVCR phase, the DON SWP program should use velocity-based metrics, EBV, and the rate of created defects/completed work items in a set period and test coverage to respectively track and assess productivity, value, and quality from the time software development work is first funded to demonstrating its prototype to the user in a production-like testing environment (e.g., its MVP). But once the MVP is accomplished, DON SWP programs gain the critical ability to regularly capture feedback for completed features. Thus, after showing the MVP, DON SWP programs should replace EBV with product-oriented metrics to track and assess the number and/or percentage of accepted user stories/features immediately after every iteration/sprint through delivery of the MVCR.

By regularly demonstrating and tracking completed features to users in this manner, DON SWP programs will enable themselves to iteratively reduce risk and practice UCD; maximize their use of automated telemetry and establish a metrics review process that later becomes part of the VA process, as required by the DON SWP; and implement a robust combination of productivity, code-level, and economic performance indicators that aligns to agile metrics best practices (Oza & Korkala, 2012). That said, development projects are capital investment plans, and each development project plan is merely a proxy for the intended product (Perri, 2018). Additionally, technology is just a tool, but it is the product that serves as a vehicle enabling the exchange of value (Perri, 2018). Thus, once the DON SWP program deploys its software system to operations, its metric considerations necessarily become much more complex. The next section includes a discussion of metrics and metrics considerations to manage software development in the post-MVCR phase.

C. POST–MINIMUM VIABLE CAPABILITY RELEASE

This section includes recommended metrics for DON SWP programs to manage performance throughout the post-MVCR phase. Once DON SWP programs have deployed their custom software system to operations, there are several metrics considerations. First, the programs must track and semiannually report the metrics required by the DoD SWP. Second, they must track the metrics required by the DON SWP. Third, they must establish goals and metrics for said goals, and they must track their progress throughout the first VA cycle. Finally, the DON SWP programs must adopt a comprehensive management framework to plan, track, and assess longer-term software acquisition outcomes.

As noted in the previous section, to help improve the DoD SWP itself, the DoD SWP requires each SWP program to track and semiannually report its performance for 12 metrics: (a) average lead time for ATO; (b) continuous ATO in-place; (c) mean time to resolve experienced cyber incident or CVE; (d) mean time to detect cyber incident; (e) average deployment frequency; (f) average cycle time; (g) average lead time for change; (h) minimum lead time for change; (i) maximum lead time for change; (j) change fail rate; (k) MTTR; and (l) VA rating (OUSD[A&S], n.d.-g). While tracking and semiannually reporting these metrics is a matter of compliance, it is worth noting that several of these metrics, such as metrics f–k, were adopted from the DIB SWAP study's *Metrics for Software Development* supplement (DIB, 2019a). Thus, establishing the production telemetry to automatically track and collect these metrics aligns DoD SWP programs to DoD software acquisition best practices.

The DON SWP, as noted earlier, prescribed the use of four metrics: "(1) Average Deployment Frequency; (2) Average and Minimum/Maximum Lead Time to commit code to production; (3) Average Cycle Time; (4) Change Failure Rate" (ASN[RD&A]), 2022, pp. 6–7). These metrics overlap with metrics e–j required by the DoD SWP. Additionally, the DON SWP's metrics are partially aligned to the set of metrics that the DevOps Research and Assessment (DORA) *State of DevOps* research program has used to assess the software delivery performance of over 25,000 IT organizations: (a) Lead Time for Changes, (b) Deployment Frequency, (c) Change Failure Rate, and (d) MTTR

(Forsgren et al., 2018). Collectively, these four metrics have been dubbed the "DORA metrics" (Kim et al., 2021).

By partially including the DORA metrics, the DON SWP has attempted to adopt state-of-the-art software delivery practices. How these metrics are used, however, is critically important. By analyzing IT organizations via the DORA metrics, DORA's research has shown software delivery performance is highly correlated to business-level performance and growth, and that the highest performing IT organizations quickly, regularly, reliably, and responsively deploy and operate software (Forsgren et al., 2018). In other words, DORA's research has scientifically demonstrated that building a highly agile, secure, and reliable software engineering work system (i.e., practicing DevOps) is strongly connected to successful business outcomes.

Naturally, building such a system of work takes time, concerted effort, and continuous improvement, but this is where the DORA metrics help. The first two DORA metrics, Lead Time for Changes and Deployment Frequency, provide insight into the velocity of the software development process and how responsive it is to users' evolving needs, whereas the last two DORA metrics, Change Failure Rate and MTTR, indicate how stable the provided services and responsive the technology organization are to production incidents (Forsgren et al., 2018). However, when technology organizations track all four DORA metrics and widely radiate (i.e., make readily visible) their current performance levels, they motivate speed with discipline throughout the software delivery process, rallying the entire IT organization around continuous improvement of its culture, architecture, and technical practices (Kim et al., 2021).

Thus, when used holistically and displayed widely, the DORA metrics are a powerful benchmark to assess software delivery health, improve software delivery performance, and motivate DevOps practices (Forsgren et al., 2018). To serve as an effective benchmark for IT organizations, DORA has clearly defined its metrics:

- **Lead Time for Changes:** the time elapsed from code committed to code in use in production
- **Deployment Frequency:** the average rate of code deployments over a time period

- **Change Failure Rate:** the percentage of code deployments that result in a production failure, requiring rollback and/or other intervention to resolve
- **MTTR:** the average amount of time required to restore a degraded service (Forsgren et al., 2018).

Fortunately, the DoD SWP has, on one hand, adopted all four DORA metrics, and it has also adopted very similar definitions for each DORA metric (OUSD[A&S], n.d.-g). On the other hand, the DON SWP only requires the use of three of the four DORA metrics: Lead Time for Changes, Deployment Frequency, and Change Failure Rate (Forsgren et al., 2018; ASN[RD&A]), 2022). The DON SWP doesn't require the use of MTTR, which can potentially be a problem (ASN[RD&A]), 2022). As noted, to understand and manage software delivery performance, the DORA metrics must be holistically tracked (Forsgren et al., 2018). Furthermore, to motivate DevOps practices, the DORA metrics and current performance levels under them must be transparently shared throughout the technology organization (Forsgren et al., 2018). Given that the DoD SWP requires the use of MTTR anyway, DON SWP programs are advised to track and display all four DORA metrics together.

Additionally, to improve software delivery performance via the DORA metrics, DON SWP programs are advised to adopt the performance standards proposed in the DIB SWAP study's *Metrics for Software Development* supplement (DIB, 2019a). Based on DoD software acquisition best practices, the DIB (2019a) recommended a target Lead Time of less than 1 day, target Change Failure Rate of less than 10%, and target MTTR of less than 1 day for Type C software acquisition programs.

As for Deployment Frequency, because each software acquisition program has highly unique system requirements, the DIB (2019a) did not propose performance standards for this metric. However, because faster Deployment Frequency correlates with smaller batch sizes, which DORA's research has shown accelerates feedback, increases motivation and sense of urgency, and reduces cycle time, risk, and cost and schedule growth (Forsgren et al., 2018), the DIB (2019a) recommended a target time of less than 3 months to identify and deploy new functions as well as a target time of less than 1 week to find and fix new issues for Type C software acquisition programs.

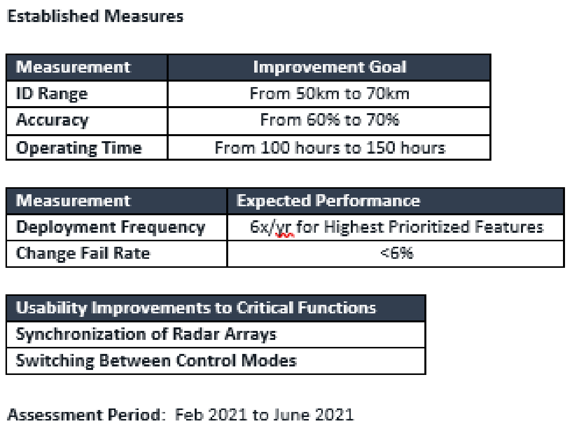
Thus, although the DON SWP prescribed the use of Lead Time, Change Failure Rate, Deployment Frequency, and Cycle Time, it did not clearly define these metrics, provide guidance for how to use said metrics, or recommend standards to monitor and assess performance against them (ASN[RD&A], 2022). To adopt state-of-the-art software delivery and DoD software acquisition practices, Type C DON SWP programs should holistically track and display Lead Time, Change Failure Rate, Deployment Frequency, and MTTR; they should adopt the DoD SWP's definitions for these metrics, and they should adopt the standards for Lead Time, Change Failure Rate, and MTTR proposed by the DIB (2019a) within *Metrics for Software Development*. Additionally, Type C DON SWP programs should track the time to identify and deploy new functions and the time to find and fix issues, as well as adopt the proposed performance standards for these metrics. Finally, DON SWP programs should adopt the DoD SWP's definition of Cycle Time to ensure clarity and consistency. Following these recommendations will enable DON SWP programs to not only adopt leading software delivery practices but also make the most of the DON SWP's required metrics. That said, DON SWP programs still have more metrics considerations for the post-MVCR phase.

Upon accomplishing MVCR, SWP programs begin their first VA cycle and have up to 1 year to complete it (OUSD[A&S], n.d.-i). The purpose of the VA is for the Operational Sponsor (OS), the "individual [office] that holds the authority and advocates for needed end user capabilities and associated resource commitments" (OUSD[A&S], 2020c, p. 22) to assess the outcomes of all delivered software in each VA cycle. Thus, the VA process is a critical source of feedback, for it enables the DA and the PMO to assess the SWP program's progress, update strategies and designs, and make informed resourcing decisions throughout their post-MVCR phase (OUSD[A&S], n.d.-i).

To drive rigor in doing so, the DoD SWP encourages using both objective and subjective goals in each VA cycle, requires VAs to be conducted at least once per year, and requires the OS to assess goals using a standardized rating scale (OUSD[A&S], n.d.-i). However, to plan and manage their VA cycles, SWP programs are afforded broad discretion to set relevant goals, select appropriate performance metrics, and drive improvements based on their mission needs.

For instance, the timing of each VA cycle is negotiated between the OS and the PMO (OUSD[A&S], n.d.-i). Additionally, the VA template lists a variety of measures of effectiveness to consider in goal setting, including (a) Software Development Performance, (b) Increase in Mission Effectiveness, (c) Cost Efficiencies, (d) User Workload Reduction, (e) Manpower Reduction, (f) Equipment Footprint Reduction, and (g) User Adoption (OUSD[A&S], n.d.-i). In any case, the SWP program and OS identify all performance goals at the start of each VA cycle within the User Agreement, and they continually update them each VA cycle to ensure performance improvements align to mission priorities (OUSD[A&S], n.d.-i).

To model initial goal setting, the VA template lists a notional group of objective goals and supporting metrics, as well as subjective goals, as shown in Figure 29 (OUSD[A&S], n.d.-i).



Established Measures

Measurement	Improvement Goal
ID Range	From 50km to 70km
Accuracy	From 60% to 70%
Operating Time	From 100 hours to 150 hours

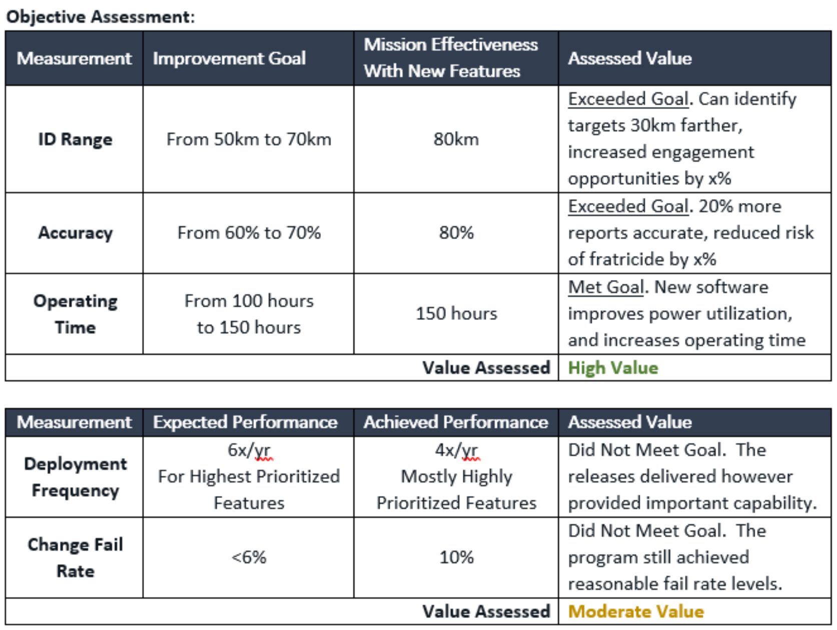
Measurement	Expected Performance
Deployment Frequency	6x/yr for Highest Prioritized Features
Change Fail Rate	<6%

Usability Improvements to Critical Functions
Synchronization of Radar Arrays
Switching Between Control Modes

Assessment Period: Feb 2021 to June 2021

Figure 29. Notional VA Cycle Goals. Source: OUSD(A&S, n.d.-i).

To model assessment at the end of the VA cycle—a 4-month period, in this case—the VA template also shows how the OS rated each notional goal in Figures 30 and 31.

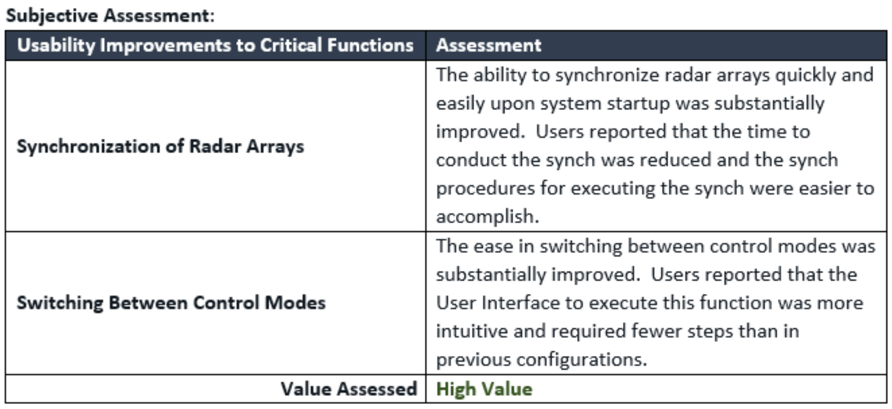


Objective Assessment:

Measurement	Improvement Goal	Mission Effectiveness With New Features	Assessed Value
ID Range	From 50km to 70km	80km	Exceeded Goal. Can identify targets 30km farther, increased engagement opportunities by x%
Accuracy	From 60% to 70%	80%	Exceeded Goal. 20% more reports accurate, reduced risk of fratricide by x%
Operating Time	From 100 hours to 150 hours	150 hours	Met Goal. New software improves power utilization, and increases operating time
		Value Assessed	**High Value**

Measurement	Expected Performance	Achieved Performance	Assessed Value
Deployment Frequency	6x/yr For Highest Prioritized Features	4x/yr Mostly Highly Prioritized Features	Did Not Meet Goal. The releases delivered however provided important capability.
Change Fail Rate	<6%	10%	Did Not Meet Goal. The program still achieved reasonable fail rate levels.
		Value Assessed	**Moderate Value**

Figure 30. Notional Objective Goal Assessment. Source: OUSD(A&S, n.d.-i).

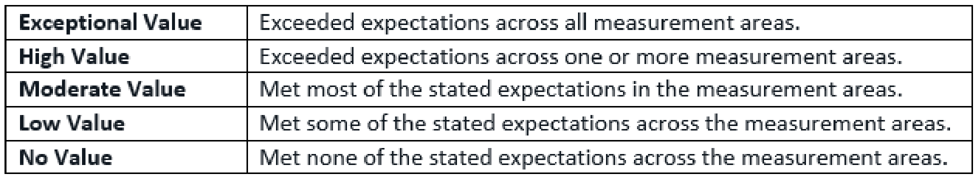


Subjective Assessment:

Usability Improvements to Critical Functions	Assessment
Synchronization of Radar Arrays	The ability to synchronize radar arrays quickly and easily upon system startup was substantially improved. Users reported that the time to conduct the synch was reduced and the synch procedures for executing the synch were easier to accomplish.
Switching Between Control Modes	The ease in switching between control modes was substantially improved. Users reported that the User Interface to execute this function was more intuitive and required fewer steps than in previous configurations.
Value Assessed	**High Value**

Figure 31. Notional Subjective Goal Assessment. Source: OUSD(A&S, n.d.-i).

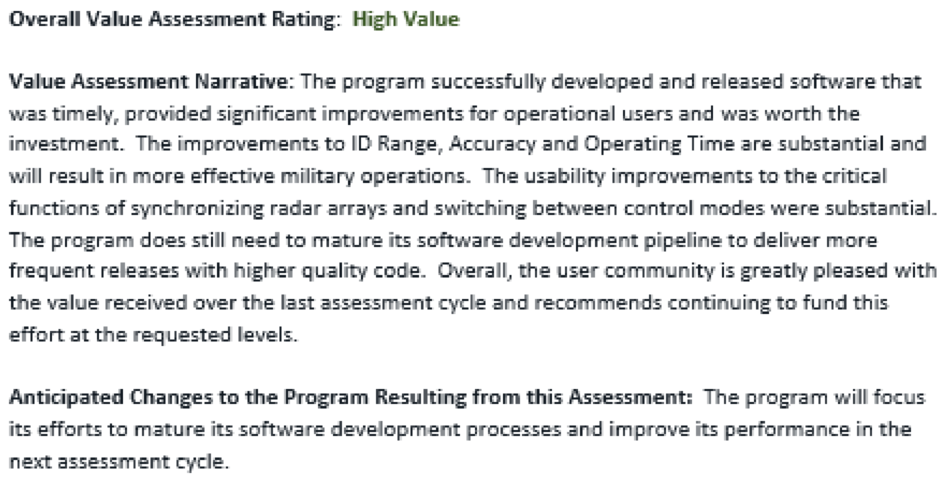
Thus, a VA cycle is complete, and the OS assesses the SWP program's performance under each of its goals. To facilitate clear, consistent VAs, the VA template has standardized the rating scale, as shown in Figure 32 (OUSD[A&S], n.d.-i).



Exceptional Value	Exceeded expectations across all measurement areas.
High Value	Exceeded expectations across one or more measurement areas.
Moderate Value	Met most of the stated expectations in the measurement areas.
Low Value	Met some of the stated expectations across the measurement areas.
No Value	Met none of the stated expectations across the measurement areas.

Figure 32. VA Rating Scale. Source: OUSD(A&S, n.d.-i).

Finally, to close out each VA, the OS provides an overall VA Rating for the entire VA cycle, general feedback to the SWP program, and requested changes to consider in the next VA cycle, as shown in the notional example in Figure 33 (OUSD[A&S], n.d.-i).


Overall Value Assessment Rating: High Value

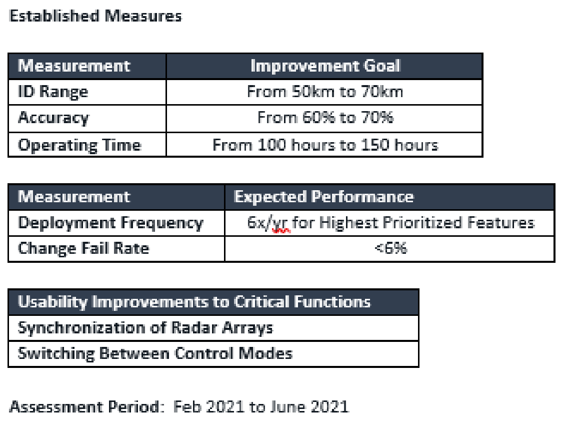
Value Assessment Narrative: The program successfully developed and released software that was timely, provided significant improvements for operational users and was worth the investment. The improvements to ID Range, Accuracy and Operating Time are substantial and will result in more effective military operations. The usability improvements to the critical functions of synchronizing radar arrays and switching between control modes were substantial. The program does still need to mature its software development pipeline to deliver more frequent releases with higher quality code. Overall, the user community is greatly pleased with the value received over the last assessment cycle and recommends continuing to fund this effort at the requested levels.

Anticipated Changes to the Program Resulting from this Assessment: The program will focus its efforts to mature its software development processes and improve its performance in the next assessment cycle.

Figure 33. Notional Overall VA Rating. Source: OUSD(A&S, n.d.-i).

Overall, the VA process captures timely, actionable, outcome-oriented feedback, providing the user community a voice and SWP programs a means to continually improve their performance. That said, while the VA template suggests using both objective and subjective goals, it doesn't clarify when and how to use each type effectively. For instance, the VA template may have used the terms "Objective Assessment" and "Subjective Assessment" merely to distinguish quantitative goals from qualitative goals, implying that they should be separated. However, this approach limits the potential for objective goals to elicit superior performance.

Naturally, to drive concrete performance improvements, DON SWP programs must ensure that objective goals, including their supporting metrics, are specific, measurable, actionable, realistic, and time-bound (SMART; Harrison & Lively, 2019). But while the notional, objective goals listed in the VA template provide SMART benchmarks, they lack a unifying vision or theme. As a result, the notional objective goals listed in Figure 34 are somewhat mechanical:


Established Measures

Measurement	Improvement Goal
ID Range	From 50km to 70km
Accuracy	From 60% to 70%
Operating Time	From 100 hours to 150 hours

Measurement	Expected Performance
Deployment Frequency	6x/yr for Highest Prioritized Features
Change Fail Rate	<6%

Usability Improvements to Critical Functions
Synchronization of Radar Arrays
Switching Between Control Modes

Assessment Period: Feb 2021 to June 2021

Figure 34. Notional VA Cycle Goals. Source: OUSD(A&S, n.d.-i).

Clearly, SMART metrics are necessary but not sufficient for teams to improve performance in highly uncertain agile environments. These notional, objective goals lack several elements. First, to focus teams on mission outcomes, objective goals must provide direction by envisioning a future end state (Wodtke, 2016). Second, to intrinsically motivate knowledge workers, objective goals must also clearly articulate an inspirational purpose (Doerr, 2018). Finally, because agile is based on principles and values (Beck et al., 2001), objective goals must also reflect desired behaviors to reinforce cultural norms.

To make the most of their objective goals, and ultimately the VA process, DON SWP programs should formulate their objective goals by using Objectives and Key Results (OKRs). OKRs are a goal-setting framework and management best practice from the software engineering industry (Doerr, 2018). Within it, teams first define what is to be achieved through a "significant, concrete, action oriented, and (ideally) inspirational" (Doerr, 2018, p. 7) qualitative goal, entitled an Objective; then, they pair it with SMART metrics, entitled Key Results (KRs), which provide success criteria to benchmark and monitor how to accomplish the intended Objective successfully (Doerr, 2018). When combined, each Objective and its set of KRs form an OKR, an outcome-oriented goal that is well-suited for objective goal setting in the DoD SWP VA process for many reasons.

First, when formulating OKRs, each Objective and its KRs must be designated a period, entitled the OKR cycle, in which to meet all intended outcomes (Doerr, 2018). To do so easily, DON SWP programs may formulate objective goals such that all OKR and VA cycles align. Second, OKRs are assessed at the end of each OKR cycle, whereby only

Objectives with no outstanding KRs are considered complete (Doerr, 2018). OKRs are therefore intended for challenging performance goals, just like the VA process. Third, while Objectives "can be long-lived, rolled over for a year or longer, [KRs] evolve as the work progresses" (Doerr, 2018, p. 8). Thus, OKRs facilitate a goal setting, tracking, and assessment process that naturally complements agile software engineering practices.

Interestingly, the notional objective goals listed in the VA template partially form two OKRs for a 4-month cycle, as shown in Tables 2 and 3:

Table 2. Partial OKR # 1. Adapted from OUSD(A&S, n.d.-i).

Measurement	Improvement Goal
ID Range	From 50km to 70km
Accuracy	From 60% to 70%
Operating Time	From 100 hours to 150 hours

Assessment Period: Feb 2021 to June 2021.

Table 3. Partial OKR # 2. Adapted from OUSD(A&S, n.d.-i).

Measurement	Expected Performance
Deployment Frequency	6x/yr For Highest Prioritized Features
Change Fail Rate	<6%

Assessment Period: Feb 2021 to June 2021.

Currently, these are two sets of KRs that lack a clearly, concisely stated qualitative goal (i.e., Objective). Given its operational context, an appropriate Objective for the first set of KRs could be "Sharpen the Warfighter's Edge." Additionally, to suit its software delivery context, an appropriate Objective for the second set of KRs could be "Small, Speedy Software Deliveries." Thus, had these notional objective goals been formulated using the OKR framework, they'd respectively appear as shown in Tables 4 and 5:

Table 4. Formulating OKR # 1. Adapted from OUSD(A&S, n.d.-i).

Objective: Sharpen the Warfighter's Edge

KR # 1	Improve ID Range From 50km to 70km
KR # 2	Improve Accuracy From 60% to 70%
KR # 3	Improve Operating Time From 100 hours to 150 hours

Assessment Period: Feb 2021 to June 2021.

Table 5. Formulating OKR # 2. Adapted from OUSD(A&S, n.d.-i).

Objective: Small, Speedy Software Deliveries	
KR # 1	6x/yr Deployment Frequency For Highest Prioritized Features
KR # 2	<6% Change Fail Rate

Assessment Period: Feb 2021 to June 2021.

Furthermore, at the end of each 4-month VA/OKR cycle, these OKRs would be respectively assessed as shown in Tables 6 and 7

Table 6. Assessing OKR # 1. Adapted from OUSD(A&S, n.d.-i).

Objective: Sharpen the Warfighter's Edge			
KR # 1	Improve ID Range From 50km to 70km	Achieved ID Range: 80 km	Assessed Value: Exceeded KR. Can identify targets 30km farther, increased engagement opportunities by x%.
KR # 2	Improve Accuracy From 60% to 70%	Achieved Accuracy: 80%	Assessed Value: Exceeded KR. 20% more reports accurate, reduced risk of fratricide by x%.
KR # 3	Improve Operating Time From 100 hours to 150 hours	Achieved Operating Time: 150 hrs	Assessed Value: Met KR. New software improves power utilization and increases operating time.

Assessment Period: Feb 2021 to June 2021.

Table 7. Assessing OKR # 2. Adapted from OUSD(A&S, n.d.-i).

Objective: Small, Speedy Software Deliveries			
KR # 1	6x/yr Deployment Frequency For Highest Priority Features	Deployment Frequency Achieved: 4x/yr	Assessed Value: Did Not Meet KR. The releases delivered however provided important capability.
KR # 2	<6% Change Fail Rate	Change Fail Rate Achieved: 10%	Assessed Value: Did Not Meet KR. The program still achieved reasonable fail rate levels.

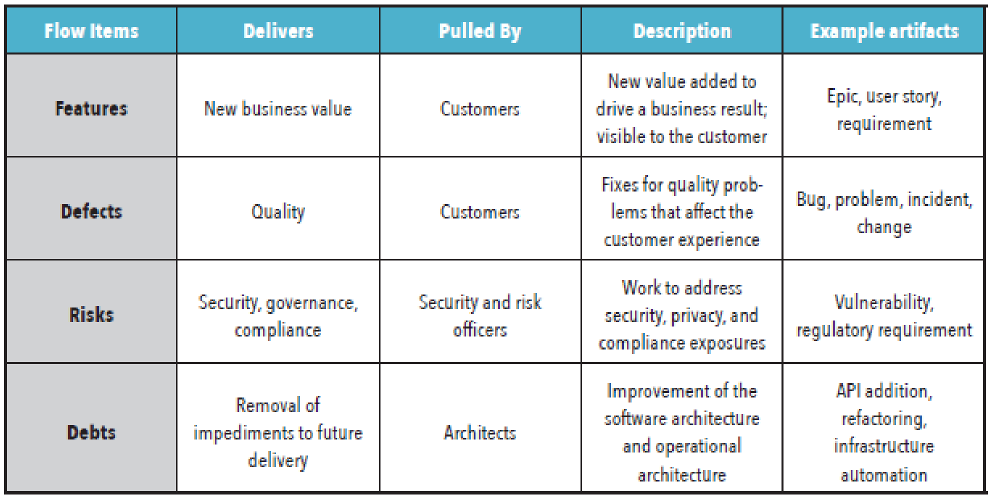
Assessment Period: Feb 2021 to June 2021.

Because all its KRs were met, OKR # 1 is considered complete. As for OKR # 2, it would either be reformulated and/or carried over into subsequent OKR/VA cycles until all its KRs are accomplished. To be sure, these may seem like simple adjustments to the notional objective goals which were formulated and assessed in the VA template. However, by providing a pithy vision, the Objectives unify each set of KRs/metrics in a forceful, expressive manner that the notional objective goals otherwise completely lacked. Moreover, whenever people "have conflicting priorities or unclear, meaningless, or arbitrarily shifting goals, they become frustrated, cynical, and demotivated" (Doerr, 2018, p. 10). OKRs solve this problem by enabling DON SWP programs to link their objective goals to the broader mission; respect clear targets and deadlines while providing freedom to maneuver and adapt to circumstances; enable teams to rapidly learn and celebrate small, meaningful wins via each KR; and, most importantly, motivate teams to strive for what might currently seem beyond reach (Doerr, 2018). Put simply, the VA template currently underutilizes objective goals, whereas OKRs are intended to make the most of them. Furthermore, OKRs are the primary goal-setting framework for Google, LinkedIn, Spotify, and many other leading high-tech companies, each of which leverages agile software development and iterative goal setting and assessment to synergistically drive and continually improve business performance (Doerr, 2018). To fully tap into the potential of its software acquisition workforces, developers, and the SWP, DON SWP programs should formulate and assess their objective goals, in each VA cycle, by using the OKR framework.

Once DON SWP programs have closed out their first VA cycle, they will establish the goals to drive program performance for their second VA cycle, then their third VA cycle, and so on for as long as their software application is in use. However, DON SWP must also avoid re-creating management approaches each VA cycle. Thanks to the DevOps movement, the software engineering industry has significantly benefitted from a proliferation of digital data collection capabilities (Kim et al., 2021). However, what is needed is telematics, or the systematic approach of instrumenting the end-to-end value chain in which a software application is developed, delivered, and operated (Hering, 2018).

When measuring the end-to-end performance of a value stream, it is important to not overly rely on one proxy metric, such as the number of LOC committed or the frequency of code deployments, as doing so both leads to goal displacement and fails to account for the complexity of software development. Ultimately, all software development work items must be tracked and made so visible that they are effectively tangible, and a robust approach is needed to connect all functional, non-functional, value-added, and non-value-added but important requirements clearly and consistently to business outcomes (Kim et al., 2021). Fortunately, the software engineering industry has already developed the Flow Framework® to do just this (Kersten, 2018). The Flow Framework® is a telematics framework designed to continually track four backlog Flow

Items: features, defects, risks, and debts (Kersten, 2018). The Flow Items are defined in Figure 35:



Flow Items	Delivers	Pulled By	Description	Example artifacts
Features	New business value	Customers	New value added to drive a business result; visible to the customer	Epic, user story, requirement
Defects	Quality	Customers	Fixes for quality prob-lems that affect the customer experience	Bug, problem, incident, change
Risks	Security, governance, compliance	Security and risk officers	Work to address security, privacy, and compliance exposures	Vulnerability, regulatory requirement
Debts	Removal of impediments to future delivery	Architects	Improvement of the software architecture and operational architecture	API addition, refactoring, infrastructure automation

Figure 35. The Flow Framework®'s Flow Items. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

Crucially, the Flow Framework®'s taxonomy for Flow Items is mutually exclusive and collectively exhaustive (MECE; Kersten, 2018). Consequently, it is a software engineering management approach that ensures all functional and nonfunctional requirements that either create or impact the ability to create software-defined BV are clearly, consistently defined and tracked, as seen in Figure 36.

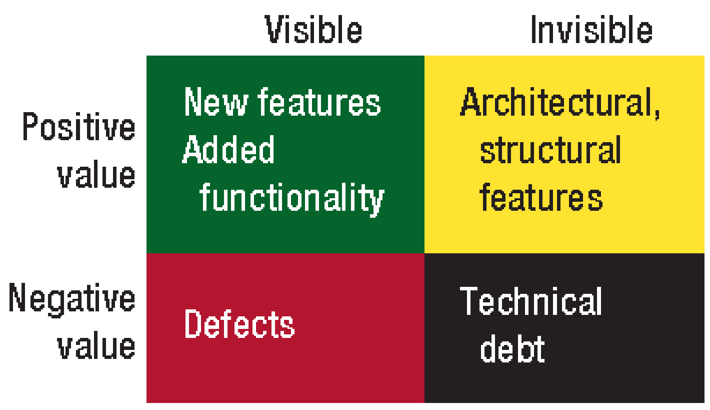
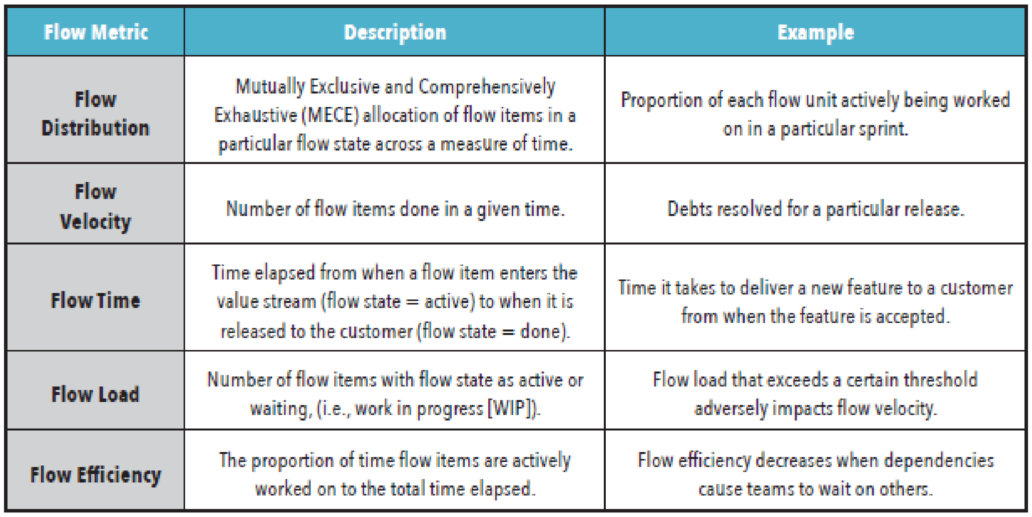

Visible
Invisible
Positive value
Negative value
New features Added functionality
Architectural, structural features
Defects
Technical debt

Figure 36. The Types of Software Development Work. Source: Kruchten et al. (2012).

Technical debt is the cost of software rework (i.e., the Flow Item® debt) that needs to be incurred at a future time, often resulting from implementing a simple solution to complete urgent development work instead of a better approach that would otherwise take longer to complete (Kersten, 2018). The goal of the Flow Framework® is to provide a holistic picture of BV produced by the software development process using five Flow Metrics, as shown in Figure 37 (Kim et al., 2021):



Flow Metric	Description	Example
Flow Distribution	Mutually Exclusive and Comprehensively Exhaustive (MECE) allocation of flow items in a particular flow state across a measure of time.	Proportion of each flow unit actively being worked on in a particular sprint.
Flow Velocity	Number of flow items done in a given time.	Debts resolved for a particular release.
Flow Time	Time elapsed from when a flow item enters the value stream (flow state = active) to when it is released to the customer (flow state = done).	Time it takes to deliver a new feature to a customer from when the feature is accepted.
Flow Load	Number of flow items with flow state as active or waiting, (i.e., work in progress [WIP]).	Flow load that exceeds a certain threshold adversely impacts flow velocity.
Flow Efficiency	The proportion of time flow items are actively worked on to the total time elapsed.	Flow efficiency decreases when dependencies cause teams to wait on others.

Figure 37. The Flow Framework®'s Flow Metrics. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

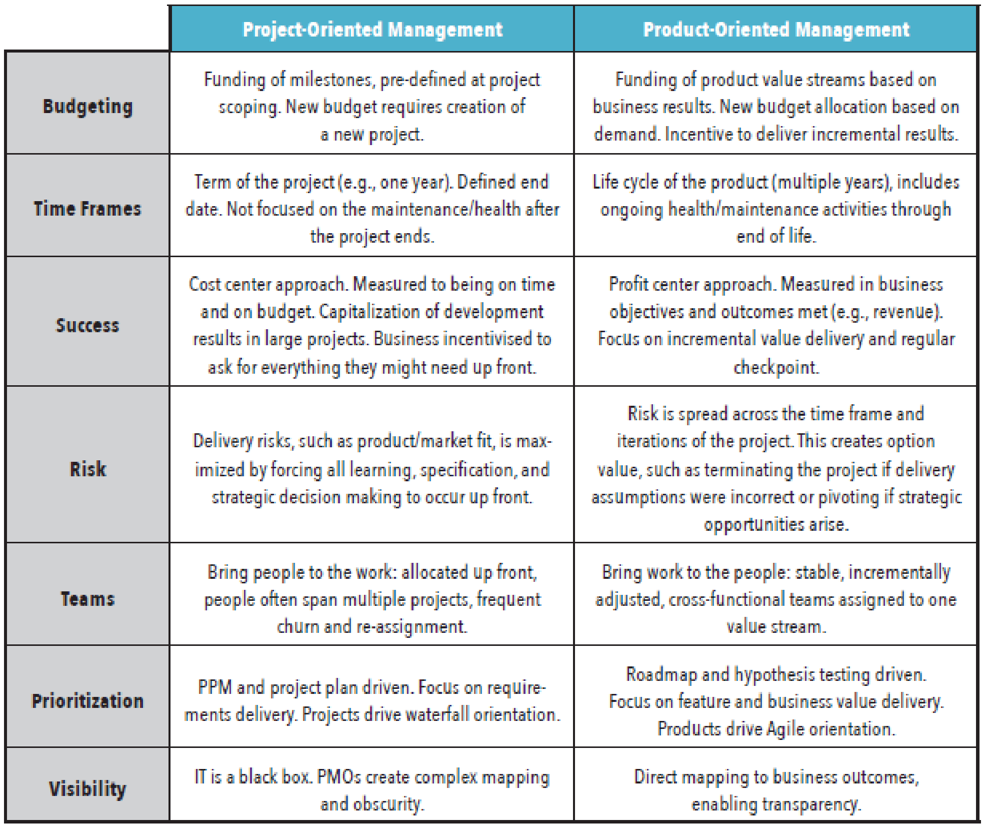
Just as the DORA metrics are a balanced set of metrics intended to holistically measure software delivery performance, the Flow Metrics are a balanced set of metrics intended to holistically measure software development performance (Kersten, 2018). Additionally, while the DORA metrics are technical in nature, Flow Metrics were designed to abstract technical details and communicate the state of software development work in a business-level context. In doing so, Flow Velocity® helps determine whether value delivery is accelerating or decelerating (Kersten, 2018) by asking, "How much value did we deliver?" (Flow Framework®, 2022). Flow Efficiency® helps identify excessive queues and the rate of waste growth in upstream work activities (Kersten, 2018) by asking, "Do we know where our bottlenecks are?" (Flow Framework®, 2022). Flow Time helps determine if time to value is getting longer or shorter (Kersten, 2018) by asking, "How fast did we deliver value?" (Flow Framework®, 2022). Flow Load® helps determine when demand is outgrowing available capacity, thereby enabling control of queue growth (Kersten, 2018), by asking, "Is demand impacting capacity?" (Flow Framework®, 2022). Finally, Flow Distribution® helps prioritize workload composition based on the intended mission outcomes at the time (Kersten, 2018) by asking, "Are we aligned to the business priority?" (Flow Framework®, 2022). Naturally, given how comprehensive the Flow Framework® is, implementing it requires a relatively mature software development and delivery process. That said, there are several reasons why DON SWP programs should adopt the Flow Framework® to measure and manage longer-term software acquisition outcomes.

First, because it uses a MECE work item taxonomy, the Flow Framework® provides DON SWP programs a comprehensive, clear, and consistent analytical framework to identify, plan, and track all software development activities that deliver value, enhance the ability to deliver value, or overall protect the ability to do so. For example, without an analytical framework to identify and track the accumulation of technical debt, DON SWP may focus only on implementing features and/or defect fixes, eventually becoming hamstrung by an overly complex codebase. On the other hand, with the aid of a decision support tool like the Flow Framework®, managers can proactively identify and trade off between functional and nonfunctional requirements, regularly refactoring the codebase to keep technical debt under control.

Second, the Flow Framework® provides a clear, consistent standard for SWP programs to measure process performance. On the other hand, while both the DoD and

DON SWP use Lead Time and Cycle Time, their metrics are not interchangeable (OUSD[A&S], n.d.-g; ASN[RD&A], 2022). Unfortunately, process performance metrics such as Lead Time and Cycle Time have been confused with each since the lean production movement of the 1980s (Kim et al., 2021). To avoid repeating the same mistakes within DoD software acquisition, the DoD SWP should standardize its process performance metrics, specifically by adopting those of the Flow Framework®.

Finally, the Flow Framework® provides the governance structure necessary for technology organizations to shift from project-oriented to product-oriented management practices, making it a critical enabler of continuous software acquisition. As noted, development project plans are merely proxies for the intended product (Perri, 2018), and project-oriented management focuses on the delivery of discrete projects according to a set of milestones, resources, and budget criteria to accomplish stable goals (Kersten, 2018). This management approach works for the pre-MVCR phase, but once DON SWP programs deliver their software to operations, project-oriented management practices become insufficient. To continually develop and deliver new and/or improved custom software and accomplish their intended mission outcomes, DON SWP programs must undergo a paradigm shift from project- to product-oriented management (Kersten, 2018) and adopt fundamentally different governance practices, as shown in Figure 38.



	Project-Oriented Management	Product-Oriented Management
Budgeting	Funding of milestones, pre-defined at project scoping. New budget requires creation of a new project.	Funding of product value streams based on business results. New budget allocation based on demand. Incentive to deliver incremental results.
Time Frames	Term of the project (e.g., one year). Defined end date. Not focused on the maintenance/health after the project ends.	Life cycle of the product (multiple years), includes ongoing health/maintenance activities through end of life.
Success	Cost center approach. Measured to being on time and on budget. Capitalization of development results in large projects. Business incentivised to ask for everything they might need up front.	Profit center approach. Measured in business objectives and outcomes met (e.g., revenue). Focus on incremental value delivery and regular checkpoint.
Risk	Delivery risks, such as product/market fit, is maximized by forcing all learning, specification, and strategic decision making to occur up front.	Risk is spread across the time frame and iterations of the project. This creates option value, such as terminating the project if delivery assumptions were incorrect or pivoting if strategic opportunities arise.
Teams	Bring people to the work: allocated up front, people often span multiple projects, frequent churn and re-assignment.	Bring work to the people: stable, incrementally adjusted, cross-functional teams assigned to one value stream.
Prioritization	PPM and project plan driven. Focus on requirements delivery. Projects drive waterfall orientation.	Roadmap and hypothesis testing driven. Focus on feature and business value delivery. Products drive Agile orientation.
Visibility	IT is a black box. PMOs create complex mapping and obscurity.	Direct mapping to business outcomes, enabling transparency.

Figure 38. Project Management vs. Product Management. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

Making this paradigm shift also requires structural change, as shown in Figure 39.

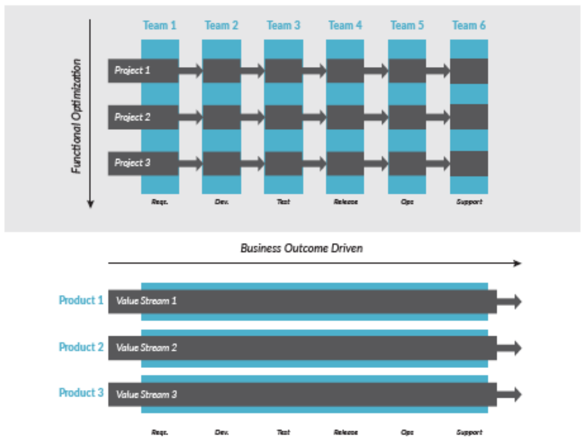


Figure 39. Project-led vs. Product-led Organization. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

Naturally, changes in organizational culture and structure take significant time and effort to implement. However, with strong organizational alignment and focus, such efforts may be dramatically more successful. The Flow Framework® is intended to drive organizational alignment and focus by managing software development through automated, transparent dashboards, charts, and other tools, as shown in Figures 40–44.

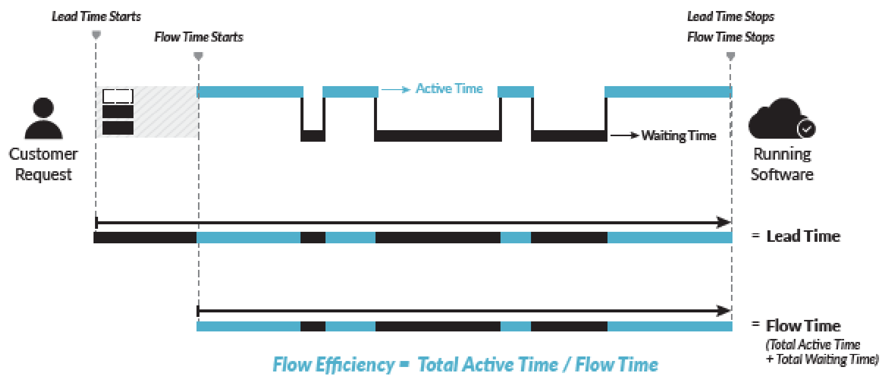


Figure 40. Flow Efficiency Chart®. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

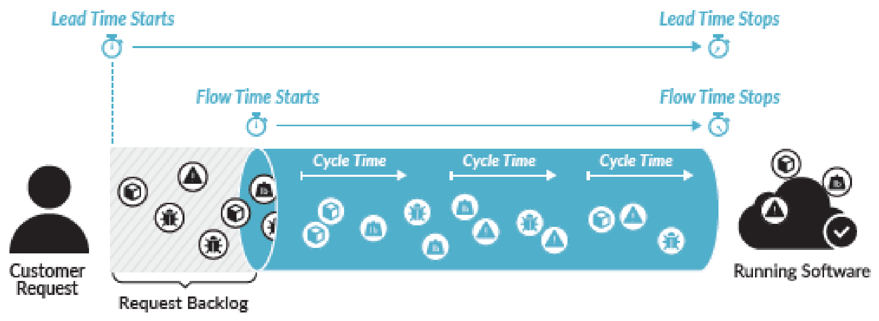


Figure 41. Comparison of Cycle Time, Flow Time, and Lead Time. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

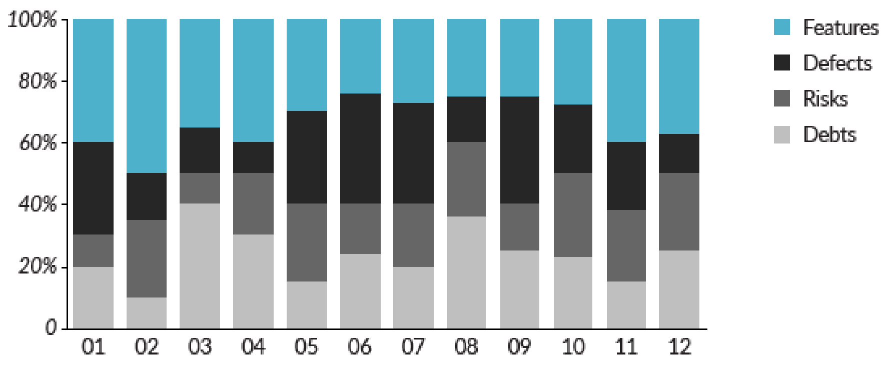


Figure 42. Flow Distribution® Dashboard. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

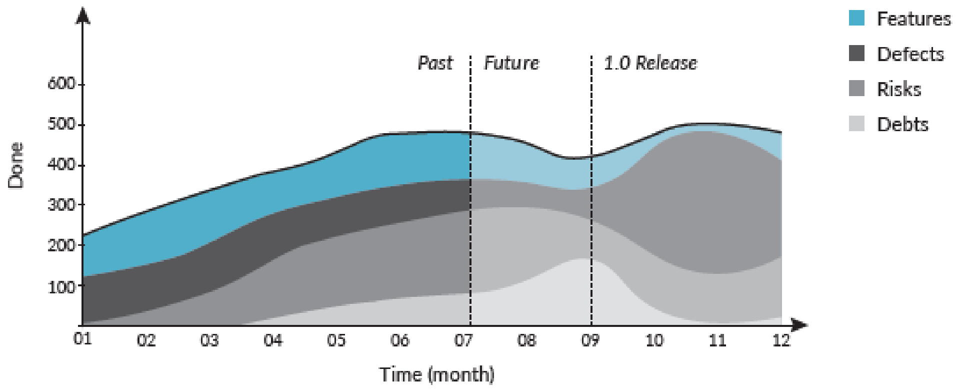


Figure 43. Flow Distribution® Timeline. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

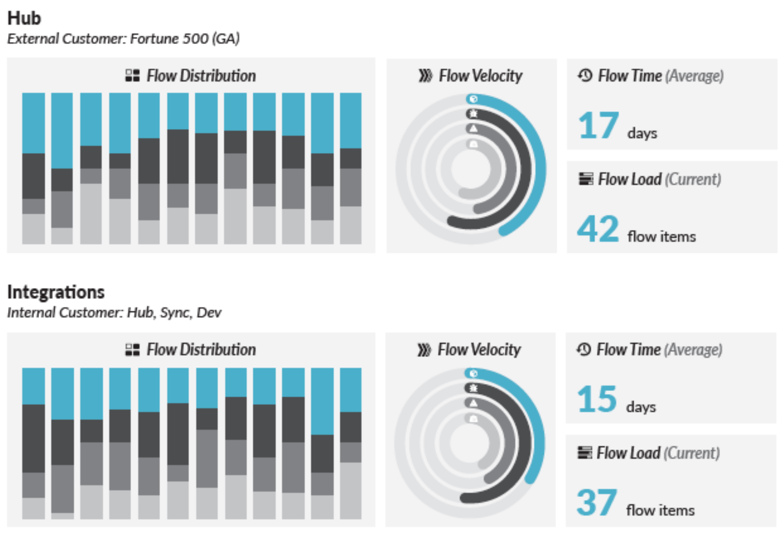


Figure 44. Notional Value Stream Dashboard. Source: Kersten (2018). Copyright © 2018 Tasktop Technologies Incorporated. All rights reserved. Published with permission.

As seen, once DON SWP programs deliver their custom software systems to operations, the Flow Framework® provides a robust telematics framework to dynamically track, manage, and scale software acquisition outcomes indefinitely. Such a framework is critical for both rapid startup and effective, long-term software acquisition decision-making. But while the Flow Framework® offers the necessary structure and tools to manage software acquisition outcomes, practicing software acquisition well also requires careful management and leadership. The next section includes a discussion of management best practices from the software engineering industry that can enable DON SWP programs to achieve effective software acquisition.

D. METRICS PRINCIPLES AND PATTERNS

Because agile software engineering is principles-based and there are many schools of practice, it is important to note that there are no "one size fits all" approaches to measurements and metrics (Miller, 2020, p. 4). As previously mentioned, there are also still no standardized agile metrics, even after 20 years (Maddox & Walker, 2021). That said, several management principles and patterns have been recognized.

First, since the agile movement began, managers have often used velocity as a proxy metric to compare software development team productivity. This should be avoided, for velocity is not an absolute measure (Forsgren et al., 2018). Software development teams usually have significantly different contexts, so their velocities are not comparable (Forsgren et al., 2018). Additionally, once teams become aware that their productivity is being evaluated, they inevitably work to game their velocity metrics by inflating their SP estimates and/or prioritizing completing as many of their own stories as possible over cross-team collaboration. As a result, comparing team velocity both distorts the utility of velocity for each team and discourages software developers—creative knowledge workers—from openly communicating and collaborating in their agile environment (Forsgren et al., 2018). Thus, team velocity should not be compared.

Second, software development performance metrics must focus teams on business-level performance (Forsgren et al., 2018). Prior to the DevOps movement, otherwise bifurcated IT organizations measured their Dev department(s) by throughput and their Ops department(s) by reliability (Forsgren et al., 2018). Because these performance metrics were locally focused, they incentivized Dev's software engineers to hurriedly implement code and hand it off to Ops, whereas Ops' software engineers become motivated to adopt heavyweight change management processes that could limit change (Forsgren et al., 2018). However, once the entire organization was visibly measured by the four balanced DevOps metrics of Lead Time for Changes, Deployment Frequency, Change Failure Rate, and MTTR (Kim et al., 2021), all software engineers became motivated to make the IT organization's software development process as responsive and stable as possible.

Third, software development performance metrics must focus teams on organizational outcomes, not outputs. When software development performance metrics measured output instead of outcome, such as LOC, they motivated people to put in busywork instead of helping to achieve organizational goals (Forsgren et al., 2018). The SWP's VA process is intended to ensure that SWP programs regularly assess their performance outcomes. But because outputs are needed to produce outcomes, managers must ensure that day-to-day output metrics continually link to outcome metrics, starting with clearly, consistently distinguishing the two as shown in Figure 45:

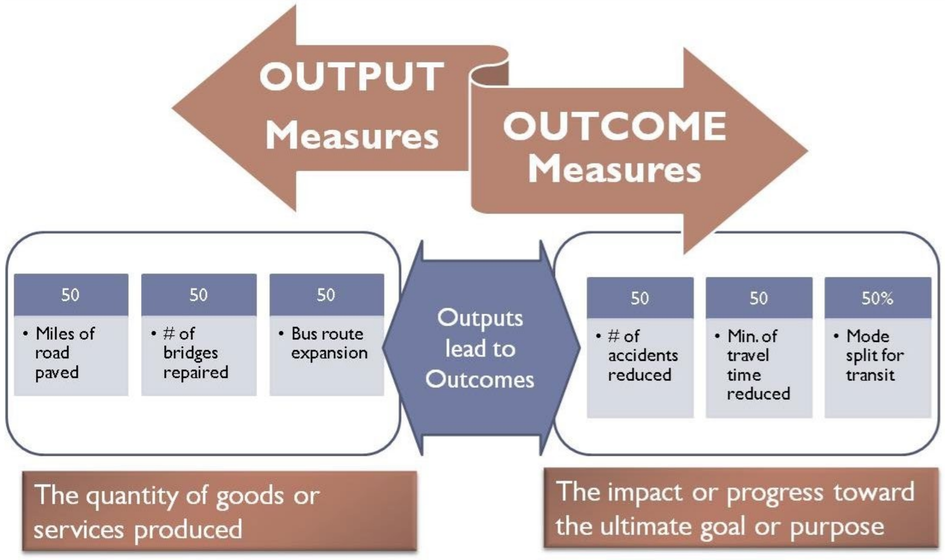
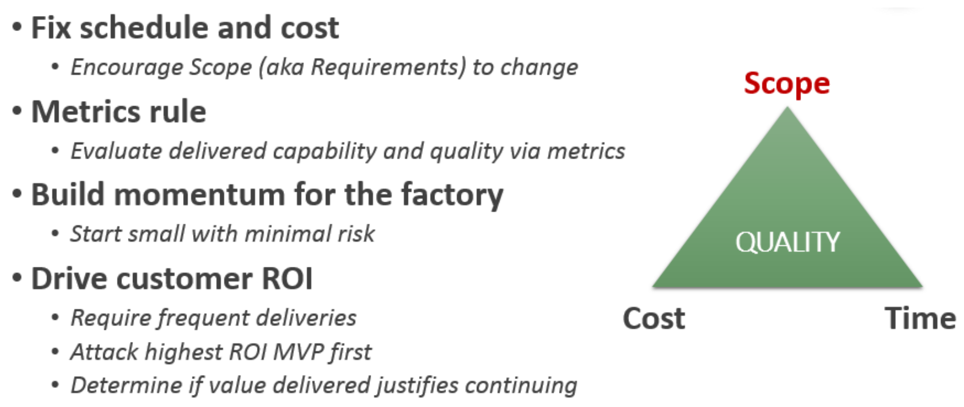


Figure 45. Output Measures Versus Outcome Measures. Source: Gavrilovic (2013).

In a software development context, outputs are quantities of delivery, such as the count of features delivered, so performance measures focused on output tell us how fast we are moving (Norton, 2020). On the other hand, outcomes are the impacts of delivery, such as how many customers are using delivered features, so performance measures focused on outcome tell us if we are headed in the right direction (Norton, 2020). Naturally, speed (output) is beneficial only if one is headed in the right direction (outcome; Norton, 2020). Thus, to help SWP programs move in the right direction, all software development metrics must link to VA outcomes. To effectively adopt software engineering industry best practices in doing so, SWP program VA goals should be set and managed using the OKR framework, and day-to-day SWP program activities should, to the greatest extent practicable, drive progress towards the KRs of each VA (i.e., OKR) cycle (Wodtke, 2016).

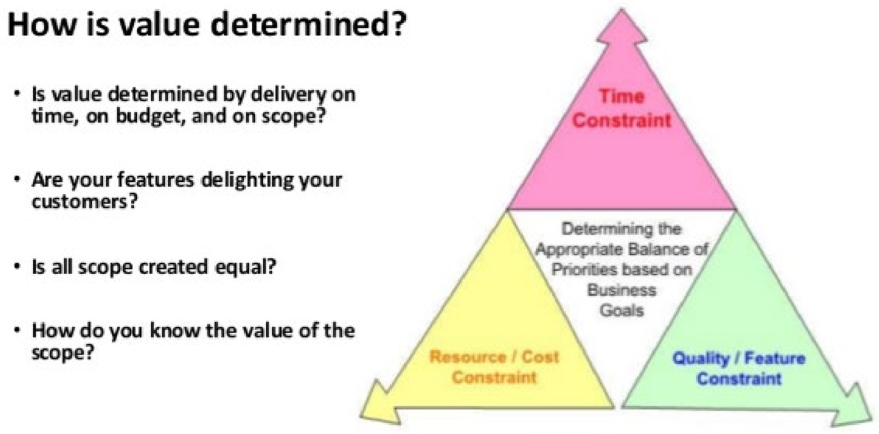
Fourth, software development progress metrics must make the correct assumptions about agile planning and agile project design. As noted, waterfall and agile software engineering methodologies utilize diametrically opposed project management paradigms, and a critical enabler of agile adoption is increasing capacity to manage dynamic requirements and requirement priorities. As such, agile planning and project design should be characterized as shown in Figure 46:



- **Fix schedule and cost**
  - *Encourage Scope (aka Requirements) to change*
- **Metrics rule**
  - *Evaluate delivered capability and quality via metrics*
- **Build momentum for the factory**
  - *Start small with minimal risk*
- **Drive customer ROI**
  - *Require frequent deliveries*
  - *Attack highest ROI MVP first*
  - *Determine if value delivered justifies continuing*

Figure 46. Agile Project Design. Source: Carpenter and Carrigan (2022).

Finally, all software development management methods must make the correct assumptions about value: It is iteratively defined in the eyes of a customer, whether external or internal. Value can be difficult to measure well because products and services are not inherently valuable (Perri, 2018). Rather, it is what products and services do for the customer or user that creates value—solving a problem, for example, or fulfilling a desire or need (Perri, 2018). Thus, metrics that attempt to measure value must ask the right questions, as shown in Figure 47:



How is value determined?

- Is value determined by delivery on time, on budget, and on scope?
- Are your features delighting your customers?
- Is all scope created equal?
- How do you know the value of the scope?

Time Constraint
Determining the Appropriate Balance of Priorities based on Business Goals
Resource / Cost Constraint
Quality / Feature Constraint

Figure 47. Determining Value in Agile Projects. Source: Burns (2017).

In context of the SWP, how value is defined and assessed for the MVP is critical, as this process develops and matures the SWP program's cultural norms. To make the most of the SWP's adoption of Lean Startup, the process by which the MVP is created, assessed, and refined should be treated as an experiment (Ries, 2011). In doing so, the

SWP program iteratively learns and refines its knowledge of the user's problem and what the user values through the Build-Measure-Learn process shown in Figure 48 (Ries, 2011):

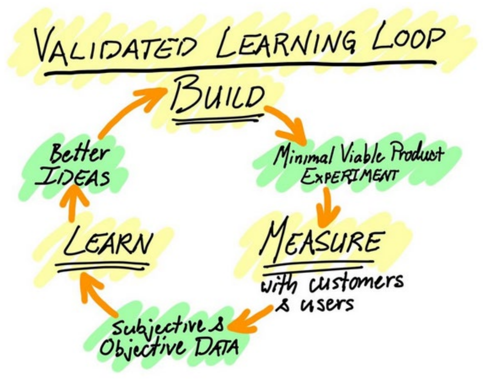


Figure 48. Build-Measure-Learn Loop. Source: Patton and Economy (2014).

Ideally, the metrics that attempt to measure the MVP's value will establish the cultural norms that later drive the SWP program's VA process. In effect, each VA cycle will constitute a Build-Measure-Learn cycle. Overall, because agile software development is a continuous process, how managers use metrics in agile environments is critical. Metrics are a means to inform and enable business decision-making. Thus, when used effectively, agile metrics do not compare teams unfairly; they focus software development teams on business-level, outcome-oriented goals; they make the right assumptions about agile project design; and they motivate software development teams to continually build, measure, and learn what the customer values.

E. SUMMARY

The DoD SWP requires the tracking and semiannual reporting of 12 metrics, several of which are not applicable prior to the initial release of software to operations. Similarly, the DON SWP requires the use of four software delivery performance metrics that are not applicable in the pre-MVCR phase. Additionally, the DON SWP requires metric reviews to become part of the VA process, yet it does not offer metrics or management methods to track and assess performance prior to conducting its first VA.

To align to agile best practices in the pre-MVCR phase, DON SWP programs should use velocity-based metrics to track productivity; EBV to track and assess progress towards building and demonstrating their MVP; and the ratio of created defects/ completed work items in a set period and test coverage to monitor quality of the software development process and codebase, respectively. Furthermore, once DON SWP programs build and demonstrate their MVP in a production-like testing environment, they should replace EBV with product-oriented metrics and track the number and/or percentage of accepted user stories/features immediately after every iteration/sprint through delivery of the MVCR. Tracking progress against recently implemented features will enable DON SWP programs to capture rapid user feedback new functionality, ensuring that they deliver the highest quality, highest priority software capability in their MVCR. Additionally, using this robust combination of productivity, code-level, and economic performance metrics enables DON SWP programs to adopt agile metric best practices, maximize their use of automated telemetry, and establish a metrics review process that becomes part of the VA process throughout the post-MVCR phase, as required by the DON SWP.

Once DON SWP programs deploy their software to operations, they must begin tracking and semiannually reporting all 12 metrics required by the DoD SWP; tracking the four metrics required by the DON SWP; tracking the metrics towards the goals established for their first VA cycle; and adopting metrics to manage longer-term software acquisition outcomes, such as technical debt reduction and/or architectural upgrades. While the DON SWP prescribed using three of the four DORA metrics, DORA intended for them to be used together to understand and manage the health of the software delivery process. Given that DON SWP programs will already be tracking all four DORA metrics—Lead Time for Changes, Deployment Frequency, Change Failure Rate, and MTTR—to comply with the DoD SWP, they should track these metrics holistically. To improve their software delivery performance via use of the DORA metrics, DoD SWP programs should also adopt the performance standards for Type C software proposed by the DIB SWAP study.

With respect to the VA process, DON SWP programs should formulate and assess their objective goals using the OKR framework, an iterative goal-setting framework widely used within the software engineering industry. To do so, they must clearly, concisely articulate a vision statement and pair it with a set of SMART outcomes.

Moreover, to effectively plan and manage all types of software engineering work—features, defects, risks, and debt (Kersten, 2018)—DON SWP programs need a comprehensive analytical framework that enables long-term, comprehensive management of the software development process. To meet this need, DON SWPs should adopt the Flow Framework®, a state-of-the-art management tool kit used within the software engineering industry.

Finally, to use agile metrics effectively, managers must ensure that they do not compare teams unfairly, that they focus teams on business-level outcomes, that they make the right assumptions about project design, and that they motivate software development teams to continually learn what the customer values. That said, metrics are only one aspect of measuring progress and performance. The next chapter includes a discussion of the most effective tools and methods to manage software acquisition programs.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AGILE SOFTWARE ENGINEERING MANAGEMENT

This section includes a discussion of the current and best practice methods and tools available to visualize and manage progress and performance of software acquisition programs. It also includes an examination of the interactions between EVM and agile software engineering, including the trade-offs necessary to implement EVM in agile software engineering environments. Finally, this section includes an evaluation of advisory report recommendations to reform DoD software acquisition and/or streamline acquisition practice. For the sake of clarity and brevity throughout this chapter, *agile* means agile software engineering, *agile project* means a software development project that practices agile software engineering, *agile project management* means the discipline of managing software development projects that practice agile software engineering, *waterfall* means waterfall software engineering, *waterfall project* means a software development project that practices waterfall software engineering, and *waterfall project management* means the discipline of managing software development projects that practice waterfall software engineering.

A. BACKGROUND

Because waterfall and agile software engineering environments use diametrically opposed project designs for cost, schedule, and scope parameters, mismatching management practices can distort agile ways of work and expectations for program success (Patel, 2021). For example, while EVM enables waterfall projects to earn and track their value by completing preset requirements, it disincentivizes agile teams from discovering and/or refining requirements via customer feedback (Wrubel et al., 2014). Naturally, agile environments need more flexible governance. That said, teams often fail to adopt agile methods by hurriedly discarding existing practices (Hayes et al., 2014).

The Agile Manifesto does not condemn the use of plans or planning, documentation standards, progress tracking, performance management, or other project governance methods and tools (Wrubel et al., 2014). Rather, it challenges software engineers to consistently build high quality software, rapidly and iteratively deliver working software to the customer, and continually adapt to the technical changes resulting from these frequent interactions through greater development speed and discipline, all of which requires new ways of thinking and managing the end-to-end flow of work (Hayes et al., 2014). Thus, instead of eliminating governance, the Agile Manifesto calls for tailoring and/or adopting project management and program assessment tools and methods to suit agile environments (Wrubel et al., 2014).

B. TODAY'S PROGRAM MANAGEMENT TOOL KIT

This section includes a discussion on currently available tools and best practices to manage software acquisition programs. Normally, acquisition programs are managed by measuring how much work is done, how much work is left, what issues there are, and whether the program is on schedule (Hayes et al., 2014). However, agile software development workflows are managed using a wide variety of tools and techniques, such as burn-down charts, burn-up charts, cumulative flow diagrams (CFDs), velocity- and defect-based metrics, version control tools, CI/CD tools, as well as recurring post-iteration/sprint reviews (Hayes et al., 2014). Software acquisition programs practicing agile can benefit from significantly increased transparency into day-to-day progress and performance (Hayes et al., 2014). That said, using these tools and techniques effectively requires learning to manage via insight as opposed to oversight, as well as adopting new ways of thinking about requirements and risk.

As noted, agile projects are designed fundamentally differently, and shifting from defining and decomposing fixed scope up front to iteratively building the system from the bottom up requires acceptance of greater levels of uncertainty (Fox, 2020). Unlike waterfall software development and stage-gate defense acquisition decision support systems that use attendant milestones and detailed cost estimates to execute towards a defined product, agile environments execute LOE processes, where one does not fund for a specific capability to be delivered at a target time and target cost (Fox, 2020). Instead, agile environments can be thought of as budgeting for the capacity to execute a certain number of LOC against dynamic requirements, where the funding level reflects how fast the development teams can burn down their backlog (Fox, 2020). Planning and budgeting work of this manner requires a fundamental cultural shift that can only happen via hands-on experimentation and learning by doing (Fox, 2020).

That said, there are many automated productivity, collaboration, and management tools designed to manage agile software engineering workflows. Commonly used ALM software, which are used to develop and manage backlogs, plan and manage iterations/sprints, plan and manage releases, determine and track team velocity, and so forth, are JIRA, VersionOne, Rally, ServiceNow, and PlanView (Mihalache, 2017; DIB, 2019b). These and other ALM technologies automatically generate three commonly used charts to visualize and monitor progress: burn-up charts, burn-down charts, and CFDs (Maddox & Walker, 2021). Burn-down charts show the amount of scheduled work items relative to the time remaining as of a specific date, as shown in Figure 49 (OUSD[A&S], 2020b).

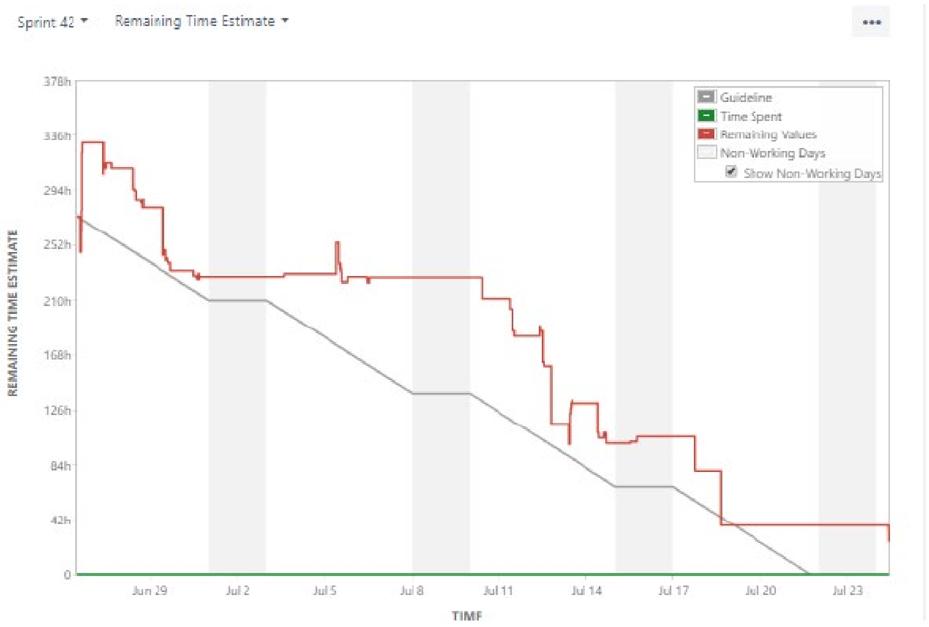


Figure 49. Burn-Down Chart. Source: OUSD(A&S, 2020b).

This specific burn-down chart shows the amount of planned versus actual work done on Sprint 42. Agile development teams use burn-down charts to track their pace of completing work units, typically measured in SPs or hours of work (OUSD[A&S], 2020b). Based on the team's throughput, the burn-down chart is used to estimate the completion date of all scheduled work items. However, because of their unique contexts, the utility of burn-down charts is mostly limited to individual development teams (OUSD[A&S], 2020b). Another tool for visualizing progress is the burn-up chart. Burn-up charts function like EVM charts comparing Earned Value versus Planned Value, for they show the total completed work relative to the total planned work as of a specific date as seen in Figure 50 (OUSD[A&S], 2020b):

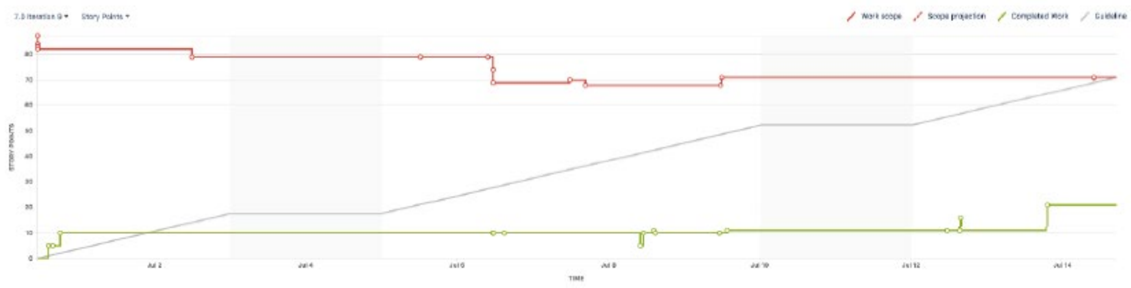


Figure 50. Burn-Up Chart. Source: OUSD(A&S, 2020b).

Like burn-down charts, burn-up charts measure work items typically in SPs or hours of work (OUSD[A&S], 2020b). Unlike burn-down charts, however, burn-up charts are used to track the rate of progress and estimate completion dates over several iterations/sprints (OUSD[A&S], 2020b). For instance, in agile environments practicing the Scrum framework, burn-up charts are commonly used to show progress planned versus actual progress over an entire release (OUSD[A&S], 2020b). Assuming the Scrum team's release backlog contained 500 SPs and its average velocity is 100 SP/sprint, then by maintaining this pace of work, the team could estimate to require about five sprints to complete all assigned work (OUSD[A&S], 2020b). That said, because of the emergent nature of software development requirements, it should be noted that longer-term planning can be highly uncertain (OUSD[A&S], 2020b). As a result, while burn-up charts provide a point-in-time progress assessment, managers should assess longer-term progress not on one but many data points (OUSD[A&S], 2020b). Moreover, because burn-up charts are based on a specific team's throughput in a unique context, their utility is limited to individual development teams (OUSD[A&S], 2020b).

One of the best tools to capture the health of a software development process is the CFD, as shown in Figure 51.

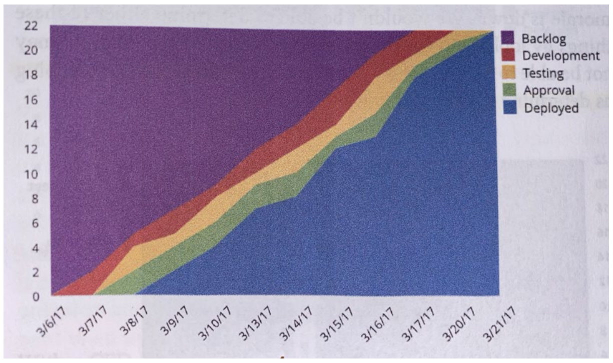


Figure 51. Cumulative Flow Diagram. Source: Norton (2020).

Given a software development process, CFDs plot the total quantity of work in each state on the y-axis, time on the x-axis, and capture lots of process performance information (Norton, 2020). The CFD's top line represents the work item arrival subprocess, typically measured in new backlog stories, and the bottom line represents the work item departure subprocess, usually defined as code deployment to production (Norton, 2020). Generally, a healthy software development process is indicated by relatively thin, parallel lines representing synchronized work arrival and work completion rates, as shown in Figure 52 (Norton, 2020):

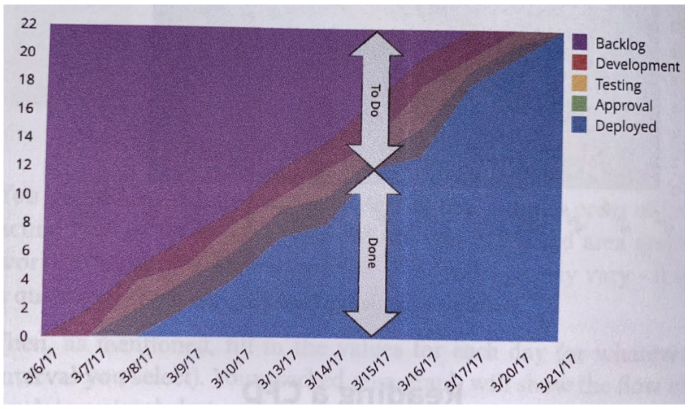


Figure 52. Determining Remaining Versus Completed Work in a CFD. Source: Norton (2020).

When viewed along the y-axis, the vertical distance between the arrival line and the departure line is work that has arrived but not yet departed the software development workflow—the instantaneous size of the queue (Reinertsen, 2009). Thus, the height of CFD bands, each of which represents a software development phase, can be used to determine WIP levels at any point in time as shown in Figure 53 (Norton, 2020):

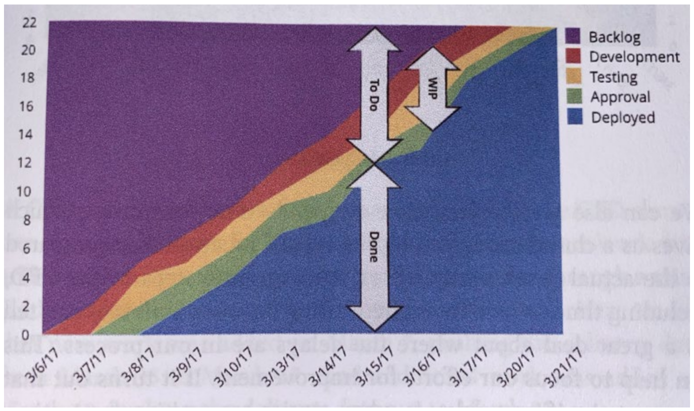


Figure 53. Determining WIP Levels in a CFD. Source: Norton (2020).

Additionally, the horizontal distance between the arrival and departure lines indicates the processing time for a work item (Reinertsen, 2009). Thus, when viewing a process or subprocess along the x-axis, CFDs also show the lead time and cycle time, respectively, for completing stories, as shown in Figure 54 (Norton, 2020).

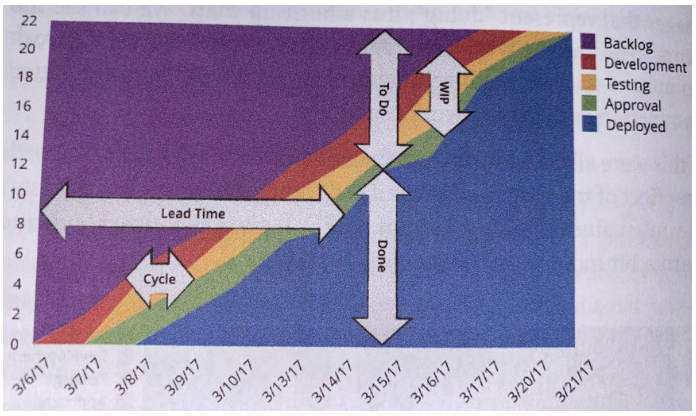


Figure 54. Determining Cycle Time and Lead Time in a CFD. Source: Norton (2020).

By showing differences in lead time and cycle time, CFDs provide context clues for how long each subprocess takes relative to the overall software development process, which is critical to detect and manage bottlenecks via targeted intervention (Norton, 2020). Furthermore, CFDs alert one to scope changes, specifically through a rise (addition) or fall (removal) in the arrival line as shown in Figure 55 (Norton, 2020):

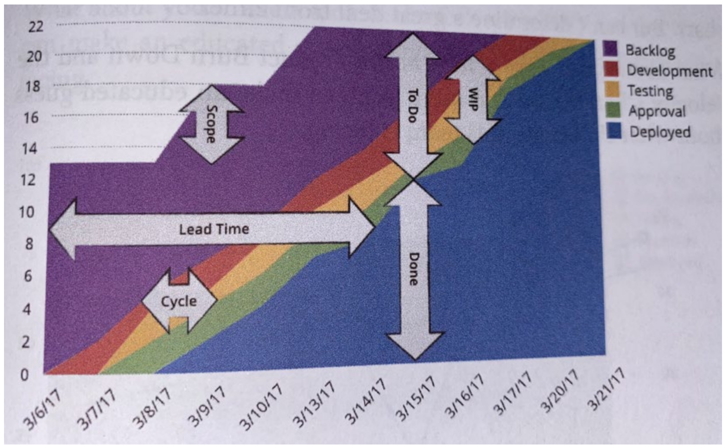


Figure 55. Determining Scope Changes in a CFD. Source: Norton (2020).

Relatively speaking, CFDs are far more robust than burn-down, burn-up, and other commonly used charts in agile environments. For instance, the burn-down chart in Figure 56 indicates a potential lapse in progress, but it is not clear what the team's issue could be (Norton, 2020).

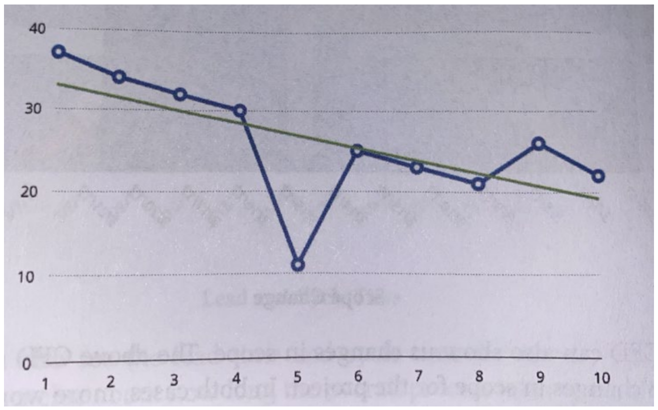


Figure 56. Burn-Down Chart Deviation. Source: Norton (2020).

Similarly, the velocity chart in Figure 57, a chart that simply shows the velocities of prior iteration/sprints, shows no apparent progress in Iteration/Sprint 6 (Norton, 2020). Again, however, it is not clear what the team's issue could be:

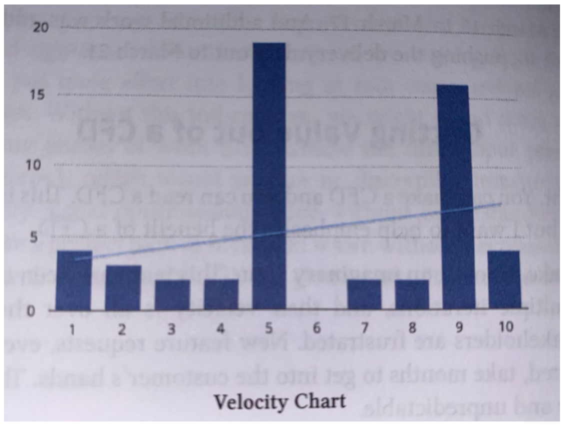


Figure 57. Velocity Chart Deviation. Source: Norton (2020).

However, as shown by the CFD in Figure 58, the team gained new scope in Iteration/Sprint 6, which can help explain their dip in throughput (Norton, 2020).

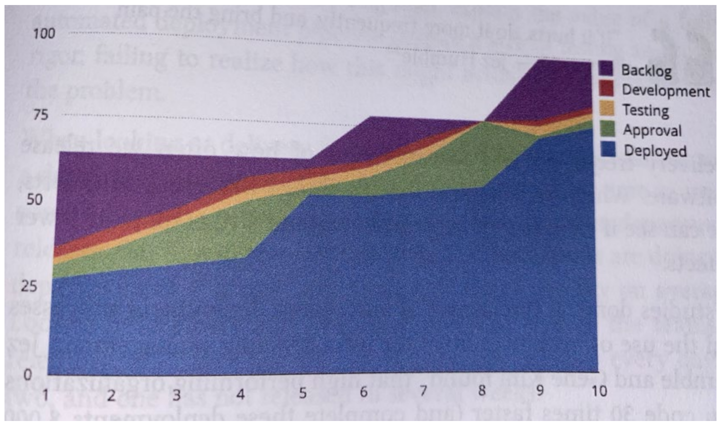


Figure 58. CFD Showing Two Scope Additions. Source: Norton (2020).

CFDs can also be used to determine demand and capacity, for the slope of the arrival line indicates the level of demand feeding into the queue, whereas the slope of the departure line indicates the capacity of the process emptying the queue (Reinertsen, 2009). Thus, when a CFD band widens too quickly, the arrival rate of work items (i.e., the demand) at that subprocess exceeds the departure rate of work items, or capacity (Reinertsen, 2009). To avoid delays, one would need to focus on completing current tasks before starting new ones. On the other hand, when a CFD band narrows too quickly, capacity exceeds demand at the affected subprocess, such that one might consider reallocating it (Reinertsen, 2009). Overall, given their versatility and holistic view of software development activities, CFDs are one of the most effective tools to manage work in agile environments (Reinertsen, 2009).

According to agile best practices, once software has been released to users, the system of work should always be to identify the total number of escaped defects, the rate of escaped defects, as well as the rate of work units (i.e., stories) delivered in a software development process (Norton, 2020). Fortunately, burn-down charts, burn-up charts, and CFDs each offer unique insights to track the rate of work units delivered. As for escaped defects, these are defects that somehow escaped preproduction processes and became detected in production environments, resulting in degraded quality for users (OUSD[A&S], 2020b). Each time a defect is found in production, a ticket (i.e., work item) must be manually created to track and fix the defect using ALM tools. Once escaped defects are assigned to tickets, however, ALM tools automatically track the total number and rate of escaped defects over set time periods. Thus, by enabling the use of burn-down charts, burn-up charts, and CFDs, as well as the monitoring of the total quantity and trends of escaped defects, ALM tools facilitate the adoption of agile best practices (Norton, 2020). Excessive quantities and/or poor trends of escaped defects indicate deficiencies in the software development process, so DON SWPs are advised to monitor these metrics throughout the post-MVCR phase, for defects should ideally be contained through automated testing and other preproduction testing activities (OUSD[A&S], 2020b).

As for managing schedule, agile development teams often believe they must stabilize their velocity (i.e., iteration/sprint throughput) before using it as a basis to forecast completion dates of their assigned work (Norton, 2020). However, because software development workflows are stochastic processes, their schedule outcomes are probabilistically distributed (Reinertsen, 2009). Thus, regardless of whether an agile team's velocity has stabilized, their commitment to point estimate completion dates is an inherently flawed approach (Norton, 2020). To account for variability when estimating completion dates, and thereby improve the quality of conversations with stakeholders regarding their schedule commitments, agile teams should use Monte Carlo simulation methods in their velocity-based forecasting techniques (Norton, 2020). One publicly available automated tool, the Throughput Forecaster, enables agile teams to probabilistically forecast their completion date (Norton, 2020). To use it, agile teams specify low and high bounds for the number of remaining stories, level of complexity, low and high bounds for split rate, and their iteration/sprint length. The tool then simulates 500 trials to complete all required work, outputting a histogram of likely schedule outcomes as shown in Figure 59 (Magennis, 2017):

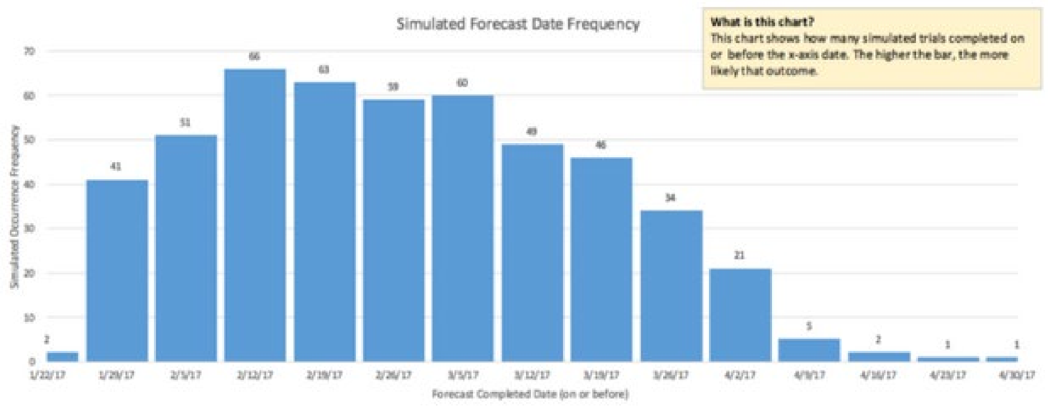


Figure 59. Monte Carlo Simulated Schedule Forecast. Source: Magennis (2017).

Split rate accounts for growth in scope (work units) during development (Norton, 2020). For instance, a split rate of 1.00 for the low bound indicates no split, and a split rate of 1.18 for the high bound indicates that for every 50 work units started, 59 work units will result (Norton, 2020). Based on Figure 59, the team is most likely to complete all required work on or before February 12 (Magennis, 2017). However, the team is also now much more aware of its distribution of schedule outcomes. Thus, by using Monte Carlo simulation methods to account for work units remaining, work unit growth rate, and velocity, agile software development teams can much more precisely determine the level of schedule risk and estimate their completion dates.

Overall, the Agile Manifesto demanded a mindset shift and increased capacity to continually maximize focus on individuals and interactions, working software, customer collaboration, and responding to change—all new ways of software engineering work. Yet new engineering practices are enabled and sustained by the appropriately designed management tools, processes, and governance practices, and the agile software engineering movement has created a wide variety of tools and techniques to manage progress and performance in increasingly uncertain, complex, and fast-paced Information

Age business environments (Hayes et al., 2014). Because it is principles-based, the agile movement has few standardized technical or managerial practices. That said, to enable and sustain agile ways of work, DON SWP programs must modify and/or substitute the management tools and methods that rely upon detailed, long-range planning; fixed, large-batch requirements; one predetermined delivery date; and minimal end user observation and feedback. The next section includes a discussion of the interactions between EVM and agile software engineering.

C. AGILE–EVM INTERACTIONS

This section includes an examination of the interactions between EVM and agile software engineering, an analysis of how and where they conflict, and recommendations of alternative techniques to manage performance of software-intensive development projects that practice agile software engineering. EVM is a project control system that integrates a project's work scope, schedule, and cost parameters to enable progress tracking and forecasting, trend analysis, and timely detection and resolution of potential performance issues (Department of Defense [DoD], 2019). To implement EVM, the PM must first define and organize all a project or program's technical work tasks in a hierarchal WBS; then, they must aggregate work packages and planning packages derived from the WBS to create a Performance Measurement Baseline (PMB), a comprehensive time-phased budget used to measure the accomplishment of all authorized work. The PMB creates the foundation for applying EVM techniques, so projects or programs that have a well-defined and well-managed PMB—including its source of work tasks and overall scope, the WBS—can generate timely, accurate, and useful insight regarding project progress and performance status (Dibert & Velez, 2006). By measuring progress "according to the amount of work, or investment, already done, relative to the amount still to do" (Goldratt, 1997, p. 73), EVM uses many metrics to provide early warning indicators of potential project issues.

Common EVM metrics include the Budgeted Cost of Work Scheduled or Planned Value, the Budgeted Cost of Work Performed or Earned Value, the Actual Cost of Work Performed or Actual Cost, the Schedule Performance Index, the Cost Performance Index, the Schedule Variance, and the Cost Variance (DoD, 2019). Other common EVM metrics are Budget at Complete, which is the total cost and schedule parameters authorized for the project; Estimate to Complete, which captures the remaining work and its costs on the project; and Estimate at Completion, which is the is the sum of Earned Value and Estimate to Complete and is used to predict the project's total cost and total duration upon completion (Winterowd, 2013).

EVM makes two key assumptions: (a) the technical work content is known in advance at a level of detail necessary to estimate and build a WBS, and (b) the system requirements are reasonably stable (DoD, 2019). Thus, EVM applies when projects are designed with a fixed scope of requirements and pre-allocated cost and schedule resources (DoD, 2019). Given such project conditions, leveraging EVM techniques to track progress of planned work, monitor variances, and forecast end results helps the project accomplish its predetermined scope through efficient use of the resources allotted to it (DoD, 2019). On the other hand, enabling agile software engineering to be practiced requires structuring a project into several mini-projects, each of which spans an iteration/ sprint, assumes not predetermined but emergent work scope, has relatively fixed costs, and represents a microcosm of the SDLC (Ching, 2015). Consequently, unless EVM is made compatible, implementing it to govern projects designed with a fundamentally different set of requirements assumptions, planning process, and resource constraints results in distorted progress and performance metrics.

For instance, in the early 2000s, the F-22 System Program Office (SPO) implemented EVM to manage iterative development of the fighter aircraft's integrated avionics system software (Dibert & Velez, 2006). To cope with the extreme complexity, uncertainty, and volatility of the engineering work, its PMs practiced rolling wave planning (i.e., routinely re-baselining the PMB to absorb and account for frequent requirements changes; Dibert & Velez, 2006). However, as the tempo of software development activity increased, the ability to maintain the integrity of the PMB decreased, and rolling wave planning eroded the program office's confidence in the EVM data and the PMB used to generate it (Dibert & Velez, 2006). Additionally, the PMs noted that in comparison to hardware development activities, software development tends to propagate change effects at a higher rate; its work process and work product are abstract; its design process has far fewer standardized methods, components, or structures; and it is much more difficult to determine when software development tasks are completed (Dibert & Velez, 2006). Overall, practicing iterative development increased the SPO's capacity to implement emergent avionic system requirements (Dibert & Velez, 2006). However, the combined high content of software; the inherently uncertain, complex, and emergent nature of software development; and the accelerated pace of design changes escalated the potential cost, schedule, and performance problems reported by EVM to a level that exceeded both the developer's and the government's ability to contain, despite their augmenting EVM with iterative PMB management (Dibert & Velez, 2006). Given the overwhelming level of noise, it was unclear whether EVM or the contractor's implementation of EVM was the problem.

After independently evaluating this F-22 software-intensive program's implementation of EVMS standards, researchers noted that the developer inadequately controlled LOE activities (Dibert & Velez, 2006). According to the DoD EVMS Interpretation Guide, LOE is "work defined as having no practical measurable output or product that can be discretely planned and objectively measured at the work package level" (DoD, 2019, p. 84). EVMS Guideline 12 requires identifying, segregating, and minimizing all LOE work because "objective measurement of [LOE] activity is impracticable and provides little, if any, visibility into actual performance; therefore, [LOE] use must be minimized" (National Defense Industrial Association [NDIA], 2018, p. 23). Instead, EVMS standards require project work tasks to be planned, estimated, and managed using work packages (WPs), which are "the point at which work is planned, progress is measured, and earned value is computed" (DoD, 2019, p. 89).

On the other hand, as previously noted, stories are the fundamental unit of work in agile environments; and agile practitioners use an abstracted measure of relative difficulty, SPs, to estimate the team effort required to complete each story in upcoming iterations/sprints (Rawsthorne, 2006). Thus, agile teams use SPs for relative sizing to plan and manage their capacity, not as an absolute measure of the cost and schedule resources necessary to implement each story. Moreover, instead of discretely planning and objectively measuring the tasks to complete each story, agile projects are structured into several, fixed-time iterations to provide the capacity to complete stories of emergent scope. Therefore, the most basic unit of work in agile environments—stories—are planned and managed as an LOE. But since EVMS Guideline 12 requires minimizing LOE use, whereas practicing agile relies upon LOE use, the methods by which EVM and agile plan work tasks are incompatible.

Additionally, within EVM, credit for software engineering work is earned in exchange for piecewise completion of CSCIs in accordance with the WBS and schedule (Hayes et al., 2020). But given that a CSCI is simply an aggregation of software that requires configuration management (Thayer, 2003), implementing a CSCI doesn't necessarily result in useful software functionality for the customer or end user. On the other hand, credit for agile software engineering work is earned in exchange for realizing stories, each of which represents a software system feature that is defined through customer or end user feedback (Hayes et al., 2020). Consequently, EVM appraises software engineering work based on implementing CSCIs derived from the WBS, whereas agile appraises software engineering work based on implementing stories derived from customer or end user engagements. Thus, the second manner by which EVM and agile conflict is their valuation method for software engineering work.

The third conflict between EVM and agile is due to the lack of a suitable WBS model by which to implement EVM in agile environments. The current available WBS standard within the DoD, MIL-STD-881F, assumes localized software in all system hardware modules and was designed for capital-intensive, weapon system acquisition programs that must successfully pass stage-gate milestone reviews (DoD, 2022). Consequently, MIL-STD-881F assumes long-range planning, detailed specification, and large-batch documentation deliverables to meet oversight requirements, and it supports designing and managing the WBS in a manner that makes engineering changes progressively costlier as the program and its projects mature (DoD, 2022). The current WBS standard does not prohibit practicing agile software engineering (DoD, 2022). But while the standard is 308 pages long, it offers little guidance or consideration for iterative engineering and/or digital technologies (DoD, 2022). Following this standard for the sake of implementing EVM poses several challenges for agile.

First, the WBS standard assumes a project design ill-suited for agile development. While the WBS standard assumes predetermined scope and uncertain cost and schedule project parameters, an agile project assumes uncertain scope and sequences fixed batches of cost and schedule resources to enable an iterative development process. This is a vastly different level of certainty regarding cost, schedule, and scope parameters, both at the beginning of and throughout a project. Secondly, while the WBS standard assumes one monolithic project deliverable and predetermined product acceptance criteria, agile projects are structured into mini-projects to enable incremental delivery of the best presentable product to the customer, whatever that may be. Finally, the WBS standard assumes a top-down system engineering approach, whereas agile environments are intended to enable practicing both top-down and bottom-up system design as needed. Thus, the current DoD WBS standard assumes a uniform system design process that is incompatible with agile software engineering.

While no WBS model suited for agile yet exists, the DoD also lacks a software-centric WBS standard in general (Winterowd, 2013). Crucially, the purpose of practicing agile software engineering is to enable a DoD software program to rapidly and iteratively deliver software capability to the user. However, using a WBS to conduct heavyweight, hardware-oriented planning reduces software programs' capacity to gracefully absorb, create, and deliver digital technology design changes, ultimately compromising the purpose of adopting agile methods in the first place (Winterowd, 2013). While beyond the scope of this research, it is possible that a software-centric WBS standard should start with new lexicon, for the very name *Work Breakdown Structure* requires a technology development paradigm anchored on top-down design and preparing work for a monolithic solution (Winterowd, 2013). Developing something like a Software Work Structure (SWS) standard that is designed for software system engineering could provide a technical work management framework that is aligned to the unique attributes of software and complexity of software development (Winterowd, 2013). Moreover, a new SWS standard could enable structuring, planning, and managing software engineering work in a way that enables EVM to be practiced while remaining robust to frequent change.

EVMS standards require rigorous change management controls—including documenting traceability, where trade-offs occurred, and evaluation of program impacts—to preserve the integrity of the EVM data generated from the PMB (Dibert &

Velez, 2006). As noted, however, dynamically controlling the PMB becomes untenable in software development project environments of persistent requirements uncertainty, complexity, and rapid change (Dibert & Velez, 2006). That said, improving the PMB's robustness in such environments should be done by utilizing an SWS, not by overlaying rolling wave planning to continually re-baseline the PMB. An SWS could resolve the underlying problem. However, until such an SWS exists, there is no useful convention for DoD software programs to organize software engineering work, generate a PMB, and enable effective EVM implementation (OUSD[A&S], 2020a).

The fourth reason why EVM and agile are incompatible is due to their conflicting approaches to maximize value. As previously noted, the DoD instituted EVMS standards to improve cost controls of Cold War–era major defense projects (Abba, 2017). In such cases, the design of the capitally intensive asset and the design's value were fixed and known in advance. Thus, in 1967, the DoD established EVM system standards to help monitor and control cost and schedule performance on such technology development projects (Abba, 2017). To maximize the amount of fixed value, EVM requires the contractor to routinely disclose cost and schedule performance, incentivizing them to minimize the total amounts of actual resources consumed (DoD, 2019).

On the other hand, agile development projects assume neither the design nor the design's value in advance. Instead, the design is iteratively developed, and the design's value is incrementally realized by regularly demonstrating and/or delivering the product to the customer or end user for feedback. Of course, in doing so, agile projects do not forgo managing cost and schedule resources. But instead of long-range planning and cost estimating a WBS, agile projects are structured into a series of fixed-time iterations/sprints, creating a process that enables iterative development and incremental delivery of the most valuable product possible with all remaining resources. Interestingly, recent research showed that DoD software development projects practicing agile methods saved at least 15% in total labor costs and 20% in total schedule compared to those that used waterfall (Patel, 2021). Thus, implementing EVM may not even be necessary to control cost and schedule growth, for practicing agile alone will contain cost and schedule risks and likely even result in efficiencies. In the end, however, the agile approach to maximizing value is fundamentally different.

To inform and enable the behaviors that maximize value, the Agile Manifesto drives software engineering environments toward a culture of continually developing and delivering high-priority software to the user. However, EVM drives long-range planning and heavyweight governance to preserve value, which inevitably constrains the ability to develop and manifest an agile culture. This is not necessarily a flaw with EVM. But asking the developer to maximize value by conducting long-range internal planning and conforming to said plan undermines the process of maximizing value by iteratively developing and incrementally building the most useful product possible through continuous market feedback. One cannot assume that value is both predetermined and fixed while continually revalidating value in the eyes of the customer or end user. As a result, implementing EVM in agile environments invariably creates ambiguity.

Thus, there are four ways in which agile and EVM conflict: (a) agile plans work tasks as an LOE, whereas EVM requires minimizing LOE by discretely planning all work; (b) agile appraises work based on stories derived from customer needs, whereas EVM appraises work based on CSCIs derived from the WBS; (c) agile has no convention for structuring, planning, and managing all work tasks, whereas EVM captures all scope in a standardized WBS; and (d) agile incentivizes maximizing value through an iterative, incremental process, whereas EVM incentivizes maximizing value by controlling and minimizing resource consumption.

While the National Defense Industrial Association (NDIA) has published a guide that attempts to integrate agile software engineering planning processes and practices with EVM techniques, its motivation to do so was to address the "demand for responsiveness and efficiency" (NDIA, 2019, p. 3) of EVMS systems. However, improving EVM system efficiency is necessary but not sufficient. The purpose of practicing agile software engineering is to enable a DoD software program to rapidly and iteratively deliver software capability to the user. Therefore, any attempt to adapt EVM to agile, and implement both in software acquisition, must enable these intended outcomes.

Agile is based on incremental delivery of to-be-determined scope scheduled into fixed time boxes, whereas EVM measures efficient completion of a large, fixed set of predetermined work packages (Park, 2010). Agile project teams commit to delivering requested capabilities in fixed time intervals, whereas EVM measures project teams by their completing specific tasks by a specified time (Park, 2010). Agile recognizes value when capabilities are delivered to the user, whereas EVM recognizes value for completing planned tasks (Park, 2010). Within agile, the user iteratively defines value, whereas EVM captures no user feedback (Park, 2010). Thus, for EVM to objectively measure project progress in an agile environment, the interaction of agile and EVM must support undetermined work scope, use recurring time intervals (i.e., iterations/sprints) that represent a microcosm of the SDLC, enable frequent delivery of working software, and maintain alignment to the customer's definition of value through routine feedback.

With respect to work planning and management, agile environments typically use a backlog to identify and prioritize software development work, and they use a roadmap to show upcoming iterations/sprints, software release events, and longer-term product goals (OUSD[A&S], 2020a). Both agile engineering's backlog and roadmap are designed for frequent change. According to the DoD's agile and EVM desk guide, the PMB may be developed using agile planning techniques, but it must capture all work scope to meet the intent of EVM, whether using a WBS or a WBS substitute (OUSD[A&S], 2020a). Thus, the backlog could potentially be used to develop and maintain a PMB, which in turn could be used to calculate EVM metrics.

However, because software system engineering requirements are emergent (i.e., new software system requirements become apparent only as their system modules interface and/or through customer interaction; Pelrine, 2011) the work contained in the backlog will frequently change and/or be reprioritized. To manage and implement emergent requirements effectively, agile practitioners treat the software development process as a queuing system, the backlog as a queue, and plan near-term development work in, at most, 2–4–week iterations/sprints to ensure teams have the latest knowledge of requirements and their relative priority. Thus, while developing a PMB via long-range initial planning and forecasting total project parameters is appropriate for waterfall software engineering, agile environments should develop the PMB using a shorter, less uncertain time horizon (Hayes et al., 2014). To the extent that the initial backlog enables planning, the PMB should be developed based on the software development work planned for the first few iterations/sprints (Hayes et al., 2014). Before the first PMB's period elapses, a second PMB would be developed based on the state of the backlog at the time the second PMB is created, a third PMB would be developed based on the second PMB before the latter reaches its term, and so on for the entire software development project.

If the backlog is substituted for the WBS and is used to iteratively develop a set of PMBs that span the entire project, the interaction of agile and EVM could support flexible work scope; a steady cadence of software delivery; and routine feedback on customer needs, priorities, and product use. However, without timely access to cost estimates for stories and actual cost data as each one is implemented, then integrating work scope, schedule, and cost parameters to enable objective progress measuring will become impossible. EVM metrics such as Actual Cost of Work Performed and Estimate at Completion fundamentally require cost data. Furthermore, in comparison with EVM in waterfall software development projects, implementing EVM in agile environments would ironically require more frequent generation and reporting of cost data. However, it is not likely that requiring the developer to frequently generate and report cost data will help the overall goal in practicing agile software engineering, which is to enable a DoD software program to rapidly and iteratively deliver software capability to the user. It is far more likely that levying cost reporting requirements would work against this goal. Given that fundamental aspects of EVM are irreconcilable with agile, this begs the question of whether any EVM-like techniques should be applied within agile software engineering.

EVM enables PMs to measure progress of planned work, assess the value of completed work products, identify cost and schedule performance trends, and forecast total cost and schedule at completion (Rawsthorne, 2006). But, regardless of the EVM data, managers really want to know two things: (a) How much value does the product currently provide? and (b) What percentage of work is done relative to the remaining resources necessary to accomplish the business objectives (Rawsthorne, 2006)? Additionally, in environments of persistent technological and market uncertainty, complexity, and frequent change, managers know that merely tracking to a plan is not sufficient to manage risk. Thoughtless allegiance to the original plan could assure failure if market conditions, mission parameters, or fundamental assumptions made while building the plan have changed. Thus, to mitigate risks of this work-to-plan trap, managers also have a third question: How accurate is the development project’s current vector (Hayes et al., 2014)? Once the agile movement began, the software engineering industry quickly developed management methods and tools to calculate the cumulative value and percentage completed of each work product (Rawsthorne, 2006). Furthermore, seeking to establish and maintain product and market fit through early and frequent feedback is arguably why the agile movement began in the first place (Rawsthorne, 2006).

Interestingly, once the agile movement began, some Scrum practitioners repurposed EVM’s formulas for use in agile software engineering environments using the Scrum framework. After making EVM’s formulas compatible with Scrum’s planning process, measuring activity with SPs, and conducting trend analysis using velocity, these software engineers created AgileEVM, a set of EVM metrics that enables Scrum software development projects to track schedule and cost performance of a product release using a burn-down approach (Sulaiman et al., 2006). AgileEVM can be used to track cost and schedule performance for a Scrum team working on one product release comprised of several sprints, for the Scrum team’s SP estimates and highest-priority stories are not likely to substantially change (Hayes et al., 2014). In these circumstances, a PMB is created for just one product release using stories in the backlog, enabling progress and performance management using AgileEVM. However, AgileEVM is not an intended substitute for large-scale EVMS implementation to manage development projects that span multiple cross-functional teams in a variety of engineering disciplines, such as DoD major defense acquisition programs (Hayes et al., 2014).

Ultimately, AgileEVM was met with software engineering “industry reluctance to employ” (Winterowd, 2013, p. 79). AgileEVM’s use of cost metrics, and the resultant administrative burden these cost controls impose on developers, is likely the most significant factor contributing to the tool’s poor software engineering industry adoption, despite Sulaiman et al. (2007) defending their use. In any case, this research notes that AgileEVM, a modified version of EVM that adapted all formulas, terms, and definitions to match the work planning and management processes of an agile environment, has been minimally adopted by agile software engineering practitioners. Thus, implementing AgileEVM to manage SWP programs is not advisable, for forcing agile software practitioners to conform to AgileEVM metrics may create the same conflicting incentives attributable to EVM.

As noted in Chapter III, one method that was developed to track progress in agile projects is EBV (Rawsthorne, 2006). Unlike AgileEVM, EBV avoids biased use of one-dimensional units such as dollars or time. Like AgileEVM, however, EBV is only intended to be used for one software release comprised of several iterations/sprints (Rawsthorne, 2006). Notwithstanding the limitations of AgileEVM and EBV, it is worth noting that both techniques assess project progress based on implementation of working software to meet dynamic demand, not the on-time, on-budget completion of activities per internal plans such as EVM. In the end, however, all development project plans are merely proxies for their intended product (Perri, 2018). Thus, whether traditional EVM, AgileEVM, or EBV are used to track and assess project progress, project management progress metrics must always rely on proxy metrics to assess value, for the true value of the product is not realized until its users benefit from operational use (Hayes et al., 2014). That said, agile principles and values always prioritize delivering software capability over exclusively internal activities (Beck et al., 2001). Thus, when it comes to tracking and assessing progress, agile principles and values favor proxy metrics that measure product-oriented work over proxy metrics that measure the process of work (Hayes et al., 2014).

For example, agile software engineering favors output metrics driven by SPs and the demonstration of features to customers over process metrics such as Earned Value, Schedule Performance Index, or Estimate at Completion, because only the former proxy metrics help the team understand what it is building and will deliver to the customer (Hayes et al., 2014). Moreover, instead of using both product and process metrics to measure progress, agile principles and values also call for maximizing the value of work not done (i.e., minimizing waste; Beck et al., 2001). Thus, as soon as teams are developing and demonstrating software on a regular cadence, agile practitioners forgo the necessary evil of earning value on a design document or tracking progress using EBV, for process proxy metrics will have served their purpose and become non–value added overhead (Packaged Agile, 2020).

But while agile principles and values favor product- over process-oriented proxy metrics, all proxy metrics must be utilized with great care and only temporarily. To cope with increasing complexity, large organizations tend to benchmark against proxy metrics to get the results they want (Bezos, 2017). Thus, over time, the proxy metric inevitably becomes equated with success. However, the purpose of a proxy is to create a means to the end of better serving customers—not to serve the proxy itself, as may unintentionally happen through bureaucratic inertia (Bezos, 2017). As Goodhart's Law indicates, when a measure becomes a target, it ceases to be a good measure because people unconsciously work to make metrics show a positive result despite problems underneath the hood (Packaged Agile, 2020). To mitigate the unintended consequences of proxy metrics, some agile software engineering practitioners have suggested that program performance be measured via two types of metrics, BV and diagnostics; that only one BV metric should be used at any time; and that all other software development performance metrics should be treated as temporary diagnostics to enable improved capacity for BV delivery (Hartmann & Dymond, 2006). Based on such a strategy, as well as a concerted effort to maintain alignment with the Agile Manifesto, it may be beneficial for software acquisition programs using the SWP to use a phased approach to progress metrics.

As noted, SWP programs are required to accomplish MVCR within 1 year of funding development work (OUSD[A&S], 2020c). Prior to deploying the MVCR, however, SWP programs are required to first build and demonstrate an MVP (i.e., a minimally engineered product that establishes a feedback loop to iteratively inform design decisions; Ries, 2011). Naturally, to design, develop, and deliver the most useful Type C custom software in their first release, SWP programs must think of the learning opportunities created by the MVP demonstration in the mid-to-late pre-MVCR phase as critical. To make the most of the opportunity, SWP programs should also demonstrate the MVP in a production-like testing environment. Unlike the MVCR, however, the SWP prescribed no schedule standard for the MVP.

As noted in Chapter III, DON SWP programs should use velocity-based metrics to track productivity, the ratio of created defects/completed work items in a set period (e.g., each sprint/iteration) to track software development process quality, test coverage to track code quality, and EBV to track and assess progress towards building the MVP.

Once the MVP is demonstrated in a production-like testing environment, DON SWP programs should forgo EBV and begin tracking and updating product-oriented progress metrics, such as the number and percentage of accepted user stories/features (OUSD[A&S], 2020b). At that point, EBV will have served its purpose, so economic progress measures at that point should graduate from process-oriented to product-oriented proxy metrics, whereby completed user stories/features are demonstrated at the end of every iteration/sprint in a production-like testing environment. Ultimately, this ensures that DON SWP programs deliver the highest-priority capability in the MVCR.

Once SWP programs deploy their Type C custom software to operations, their first VA cycle begins (OUSD[A&S], 2020c). The SWP requires VAs to be conducted at least annually, and the metric(s) used in the VA are tailored to the software system, mission needs, and customer priorities (OUSD[A&S]), 2020c). Because each VA is based on actual product use, not merely the demonstrations of completed user stories/ features, VA metrics should replace product-oriented metrics to measure and assess value as soon as DON SWP programs operationalize their software systems.

Moreover, to adopt agile best practices, VA metrics should serve as the only measures of tracking and steering progress to deliver value, whereas all other metrics—velocity-based metrics, quality metrics to track and reduce the quantity and/or rate of escaped defects to production, and so forth—should be treated as temporary diagnostic measures and used for no other reason than to enhance the ability to accomplish goals of the current VA cycle. In this manner, the SWP program's performance and value are assessed based directly on external stakeholder feedback; each subsequent VA can be tailored based on product, mission, and/or customer needs; a whole slew of diagnostic metrics can be tailored and used to enhance capability delivery; and most importantly, elevating and treating VA metrics as the North Star and subordinating all other measures to them improves the SWP program's ability to drive focused, sustained efforts towards the metrics that matter most.

Thus, SWP programs should refrain from using EVM as the basis for managing progress and measuring value, as there are several ways in which agile methods and EVM are fundamentally incompatible. Instead, SWP programs should manage progress and measure value using a phased metrics approach: (a) measure and manage progress using EBV to accomplish the MVP, (b) manage progress and measure value using product-oriented metrics to accomplish the MVCR, and (c) manage progress and measure value using VA metrics for as long as the Type C custom software is in use. Furthermore, to align to agile best practices and the vision of the Agile Manifesto, all other metrics should be treated as diagnostic measures that are only temporarily used to drive improvements towards these economic measures.

While EVM may be considered the standard for acquisition program performance, this standard should only apply to development projects where the design to be implemented and the value of said design are known with sufficient certainty to create a WBS. Most likely, these development project circumstances will be true for mature, hardware-intensive development projects. That said, because the design and value of the design are highly uncertain in cutting-edge, software-intensive acquisitions, governing their performance using EVM would be inappropriate. In such development project circumstances, the ways in which agile methods and EVM plan work tasks, measure value, define and manage scope, as well incentivize behaviors to maximize value are incompatible. Consequently, managing agile environments with EVM may reduce the utility of both the software engineering methodology and management framework. The proposed phased approach to managing progress and measuring value, tailored to SWP programs, may be a more effective program assessment framework than EVM.

That said, the incompatibility of agile methods and EVM do not suggest that EVM is an ineffective tool for acquisition program governance. Fundamentally, the extreme uncertainty inherent to digital development project environments is what drives agile software engineering and EVM's incompatibilities. EVM should continue to be used on capitally intensive, reasonably well-defined development projects using cost-reimbursable or incentive-type contract line items. Naturally, given such acquisition circumstances, these programs will be able to create a WBS; establish and integrate project work scope, schedule, and cost parameters; establish a PMB; and then proceed to manage progress and measure value using EVM as the standard tool for program governance (DoD, 2019). When implemented according to current EVMS standards, EVM will help effectively detect and contain cost risks, and there is no other tool as comprehensive as EVM to use in such acquisition circumstances. Since the Cost/ Schedule Control Systems Criteria were implemented in 1967, EVM has been consistently applied any time managing the risk of major systems acquisition cost growth resided with the government (Fleming & Koppelman, 1997), and EVM should remain the standard management tool for such capitally intensive cost and incentive-type contracts.

As for Type C custom software development contracts using the SWP, as noted, despite beginning over 20 years ago, the agile software engineering movement has still not converged upon standardized metrics and management tools (Maddox & Walker, 2021). Thus, a standard for program assessment, equivalent to EVM, is neither available for commercial software development projects nor for DoD/DON SWP programs. That said, there are many commonly used and best practice metrics to manage progress and measure value in agile environments. To adopt them, SWP programs should use a phased approach by using EBV to accomplish the MVP and product-oriented metrics to accomplish their MVCR and then treat VA metrics as their metrics that matter most for as long as their custom software is in operational use. While this proposed, phased approach was not intended as a new standard, it may get the job done because it was designed using existing software engineering industry methods and with the SWP's purpose in mind: to enable rapid and iterative delivery of software capability to the user.

D. ADVISORY REPORTS

This section includes an evaluation of the recommendations of advisory reports to reform DoD software acquisition and/or streamline acquisition practice. In 1987, the DSB Task Force on Military Software recommended implementing standard software development project metrics to "help ensure that costs and schedules are being met and that complete products will be delivered" (Brooks et al., 1987, p. 32). Specifically, Recommendation 20 suggested using (a) program size, (b) software complexity, (c) personnel experience, (d) testing progress, and (e) incremental-release content (Brooks et al., 1987).

In 2000, the DSB Task Force on Defense Software recommended establishing metrics and measuring techniques for software quality and completeness, the use and reporting of which would be enforced through contractual provisions (Hansen & Nesbit, 2000). Additionally, it reported that the ineffective use of metrics prevented major defense software-intensive programs from assessing software development project health and progress (Hansen & Nesbit, 2000). To account for the unique nature of software on major defense software-intensive programs, the Task Force on Defense Software recommended supplementing, not replacing, existing management practices with the following core metrics:

- **Progress:** planned value, earned value, cost performance index, schedule performance index, to complete performance index, aggregate milestone slippage against plan, and segment completion against plan
- **Staffing:** key vacancies and turnover
- **Requirements:** percentage implemented in design, percentage implemented in test, and percent change over time
- **Quality:** number of open defects, number of closed defects, age of defects, number of planned tests, number of conducted tests, and number of passed tests
- **Product Stability:** percent of baselined products inspected and total amount of corrective effort on baselined product (Hansen & Nesbit, 2000).

In 2018, the DSB noted that the classic acquisition metrics are cost, schedule, and performance and that the classic acquisition phases are development, production, and sustainment; however, modern software is in continuous development (OUSD[R&E], 2018). Therefore, designing and managing modern software development projects with these traditional defense acquisition management practices "creates a misalignment between the DoD's processes and the reality of contemporary industry practices" (OUSD[R&E], 2018, p. 21). Furthermore, while the DSB's 2018 report acknowledged that each software-intensive acquisition requires a program-appropriate management framework, it also recommended the following agile-oriented charts and/or metrics to estimate delivery status in the Missile Defense Agency's software-intensive acquisition programs:

- **Sprint Burndown Chart:** tracks work completion throughout a sprint
- **Epic and Release Burndown Chart:** tracks development progress over a larger body of work than a sprint
- **Velocity:** the average amount of work items a team completes during a sprint

- **Control Chart:** tracks the total time from initiating to completing work on individual issues
- **CFD:** shows whether a team's workflow is consistent; identifies shortages and bottlenecks (OUSD[R&E], 2018).

Additionally, the DSB's 2018 report acknowledged that while there may be short-term costs in transitioning to agile software development, such as establishing the IT ecosystem and providing acquisition staff education and training, the net costs of software acquisition programs "can be expected to decrease after adopting iterative development" (OUSD[R&E], 2018, p. 25). Thus, in its 2018 report, the DSB recommended treating the SDLC as an indefinite development process and highlighted some delivery-oriented tools and methods to manage software development projects. Additionally, while it suggested to expect long-term cost savings based on commercial industry's experience, it did not recommend specifically tracking costs (OUSD[R&E], 2018).

Around this time, however, the congressionally commissioned Advisory Panel on Streamlining and Codifying Acquisition Regulations—the Section 809 Panel—specifically recommended exempting EVM and EVM system requirements for software-intensive acquisition programs using an agile engineering approach (Section 809 Panel, 2018). Currently, DoD policy mandates EVM implementation for all cost- and incentive-type contracts valued at $20 million or more, and it requires the contractor to have a certified EVM system for cost- and incentive-type contracts valued at $100 million or more (Section 809 Panel, 2018).

According to Section 809 Panel Recommendation 19, however, this policy conflicts with agile software engineering, for agile projects require maximum flexibility to adjust scope as software development progresses and the product is iteratively built, whereas EVM requires projects to plan scope in a WBS, create a PMB to begin and govern development, then carefully control scope changes throughout the project (Section 809 Panel, 2018). Therefore, given the inherently customer-driven and dynamic approach to identifying and planning software engineering requirements in agile environments, implementing a static, "batch oriented EVM system has limited value" (Section 809 Panel, 2018, p. 153).

To provide relief from EVM and EVMS requirements, and thereby enable more effective adoption of agile, Section 809 Panel Recommendation 19 issued the following recommendations: (a) the Executive Branch should waive EVM/EVM system requirements for software development or integration contracts at any dollar value when agile methodologies are used; (b) the Executive Branch should allow the Program Executive Officer (PEO) to approve appropriate project monitoring and control methods for agile software development or integration programs; (c) the PEO should ensure agile software development or integration programs, at a minimum, track schedule accomplishment versus plan, cost accomplishment versus plan, and estimate to complete metrics; (d) and the Executive Branch should revise Defense Federal Acquisition Regulation Supplement (DFARS) 234.201, DoDI 5000.02 Table 8, and Office of Management & Budget (OMB) Circular A-11 to reflect the previously mentioned recommendations (Section 809 Panel, 2018).

As of November 2022, DFARS 234.201 does not authorize exemptions for programs using agile methodologies (Defense Federal Acquisition Regulation Supplement [DFARS], 2022), and OMB Circular A-11 notes that "EVM and agile development are complementary and can be used on the same project" (Office of Management and Budget, 2022, p. 15 of "Capital Programming Guide"). However, since the latest version of DoDI 5000.02 does not mention EVM or EVM system requirements (OUSD[A&S], 2022a), the Section 809 Panel's recommended changes to this policy are not applicable. Overall, to improve capacity to change scope, Section 809 Panel Recommendation 19 has called for relieving agile software programs from EVM/EVMS requirements. However, Recommendation 19's proposal to use, at a minimum, planned versus actual cost, planned versus actual schedule, and estimate to complete metrics undermines its findings. Without further guidance, these proposed metrics are ambiguous and could otherwise confused with EVM's Actual Cost of Work Performed, Budgeted Cost of Work Scheduled, Budgeted Cost of Work Performed, and Estimate to Complete metrics. Thus, Section 809 Panel Recommendation 19's findings may perpetuate, rather than relieve, the implementation of EVM and/or EVMS on software acquisition programs practicing agile methods.

In 2019, the DIB SWAP study contained a specific supplement entitled *Metrics for Software Development* (DIB, 2019a). This supplement recommended not using source LOC and programmer productivity metrics because, while they are readily measurable, they aren't "necessarily predictive of cost, schedule, or performance" (DIB, 2019a, p. S82). Instead, it proposed 14 metrics to track software acquisition program performance and drive improvement in cost, schedule, and performance, as shown in Table 8:

Table 8. Metrics for Software Development. Adapted from DIB (2019a).

Metric Type	Metric
Deployment Rate	• time from program launch to deployment of simplest useful functionality • time to field high priority functionality; find and fix security issue • time from code committed to code in use
Response Rate	• time required for regression testing; cybersecurity audit/penetration testing • time required to restore service after outage
Code Quality	• automated test coverage of code • number of bugs caught in testing versus field use • change failure rate (e.g., required rollback) • percentage of code available for DOD to inspect/rebuild
Functionality	• number/percentage of functions implemented • usage and user satisfaction
Program Management, Assessment, and Estimation	• complexity metrics • development plan/environment metrics
Progam Progress	• software development-based Nunn–McCurdy thresholds

Moreover, the supplement established standards for these proposed metrics based on the type of software and computing infrastructure involved, provided rationale to justify their use, and included guidance to effectively use and/or tailor them (DIB, 2019a). For instance, instead of cost-based Nunn–McCurdy thresholds that limit unit and/or total program cost growth, which may not make sense for continuously developed software programs, *Metrics for Software Development* recommended establishing intervention thresholds based on the number and rate of code commits, number of commenters on pull requests, number of pull request mergers, average and standard deviation of the number of commits per month, and so forth (DIB, 2019a). That way, management attention is based on deviations in software development activity, and because these metrics are typically automatically captured, the burden of implementing this metric is minimal for both engineers and managers (DIB, 2019a).

Overall, *Metrics for Software Development* incorporates several best practice metrics from the DevOps movement—such as time from code committed to code in production, change failure rate, and so forth—to motivate software development speed with discipline (Kim et al., 2021); and it offers a robust yet lightweight management toolkit for software acquisition programs. As such, the software acquisition management framework proposed in *Metrics for Software Development* has significant utility for SWP programs.

E. MANAGEMENT PRINCIPLES AND PATTERNS

As noted, managers initially resisted agile software development methodologies until they became accustomed to designing projects around flexible scope, minimizing batch sizes, and regularly reprioritizing development tasks. Thus, agile environments required learning and/or developing new management paradigms, for realizing that questions such as "Which features must be deferred if we run into unanticipated problems?" were more effective than "How late will we deliver?" only came with time and hands-on experience (Hayes et al., 2014, p. 11).

Moreover, because agile software development methods are intended to enhance organizational capacity to absorb and/or create changes, managers also needed to shift from static to dynamic business goals (Reinertsen, 2009). This shift in goal-setting practices is one of the reasons OKRs have been so effective in the high-tech industry: OKRs are designed to regularly set and iterate against dynamic business goals in software companies that practice agile engineering (Wodtke, 2016). On the other hand, precisely due to their intended dynamism, neither agile nor OKR methodologies have standardized tools and techniques, whether technical or managerial. Naturally, as digital technology continues to evolve, digital engineering and management know-how will have to evolve, too. To make the most of the know-how highlighted in this project, this section discusses some fundamental digital design and management theories intended to provide the know-why. There are always several principles and patterns to consider in planning and managing software acquisitions.

First, managers must be aware that software architecture and organizational structure are intimately related. In 1968, computer scientist Melvin Conway (1968) famously observed that, "Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations" (p. 31). This principle, now known as Conway's Law, essentially states that there is a symmetrical relationship between the design of the IT organization and the design of its tech stack, for better or worse (Conway, 1968). In other words, the organization chart and its software system architecture mimic each other, for the system's structure reinforces the organizational structure, and the organization's structure reinforces the system's structure (Norton, 2020). One cannot be changed without appropriately changing the other; they must evolve together (Norton, 2020).

Consequently, in planning and staffing software development projects, managers must maintain an abstract awareness of organizational dynamics and how they inform both organizational and tech stack design. Furthermore, because metrics and management practices often induce behaviors which require changing the tech stack, managers must be mindful in how organizational structures and development team topologies either enable or inhibit the optimal flow of software engineering work. Specifically, Conway's Law implies that software delivery teams must be separated from those who support software delivery teams (e.g., contracting, finance, and other acquisition staff members), for only the former directly change the codebase. Thus, DON SWP programs should seek to organize software delivery teams in the most optimal way to facilitate rapid and iterative software delivery, especially throughout the post-MVCR phase. As for the MVCR, given the complex communication channels involved in obtaining ATO and other necessary accreditations, Conway's Law also explains why deploying a new software system for the first time can take so long within the DoD, as shown in Figure 60:

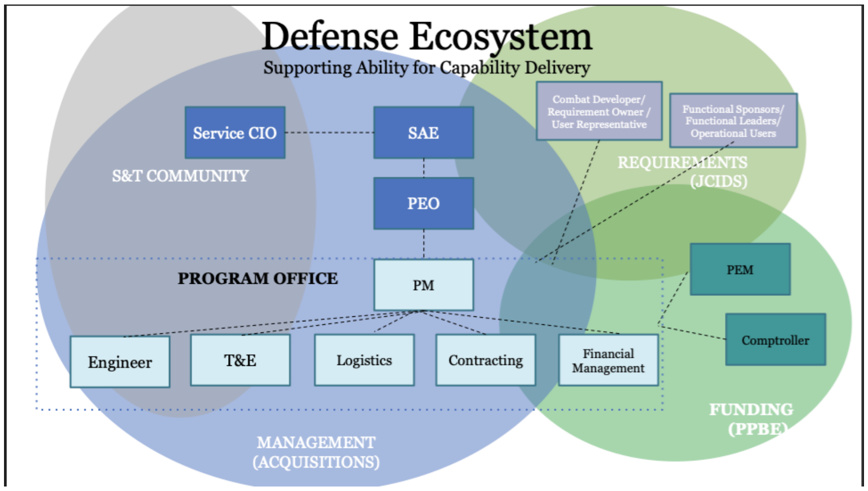

Defense Ecosystem
Supporting Ability for Capability Delivery
Service CIO
SAE
PEO
PM
S&T COMMUNITY
Combat Developer/ Requirement Owner / User Representative
Functional Sponsors/ Functional Leaders/ Operational Users
REQUIREMENTS (JCIDS)
PROGRAM OFFICE
PEM
Comptroller
Engineer
T&E
Logistics
Contracting
Financial Management
FUNDING (PPBE)
MANAGEMENT (ACQUISITIONS)

Figure 60. The DoD Software Acquisition Ecosystem. Source: DIB (2019b).

Thus, how the organization chart helps or hurts software capability delivery is critical. As for optimizing software architecture, new software programs are generally designed utilizing a monolithic software architecture, meaning that all software shares the same logical structure, is compiled together, and is deployed through a single process (Hering, 2018). The simplicity of monolithic software architectures works very well in the early life of new software programs, despite their tightly coupled code base. However, as software programs evolve to incorporate new functionalities, their logical structure and deployment process inevitably become increasingly complex—including the communication structures of the teams that do and/or manage the software development work. As software programs mature, they must evolve from a monolithic to a service-oriented architecture (SOA), a software architecture explicitly designed to enable the independent design and deployment of domain-specific services (Hering, 2018). Given the DoD's adoption of cloud-native software technology, SWP programs are equipped with the tools to independently design and deploy new software functionality rather quickly, such that transitioning to an SOA has become much more feasible. However, as long as the organizational structure and/or software architecture are inadequately designed, SWP programs' ability to create and deliver value quickly, through as many economical channels as possible, will be constrained. Clearly, maintaining a monolithic IT ecosystem in the long-term is self-limiting.

Put simply, Conway's Law states that systems reflect the organizational structure in which they were built (Hering, 2018). Therefore, to make Conway's Law work for them, managers must create the organizational structure that they'd like to have reflected in the system architecture (Hering, 2018). For containerized software technology, the most effective organizational models are those where the application container is fully owned by one, balanced product team (Hering, 2018). In such environments, product teams can quickly develop, deploy, manage, and continually improve their own application. If the applications are relatively small, then one team can optimally own multiple (Hering, 2018). However, if the application container is too large for one team, then it's likely too large in general and should be broken down further (Hering, 2018). Intuitively, managers understand that both organizational structure and system architecture must be optimized; however, they may not be aware that the two are intimately related in technology organizations. In the end, managers must make Conway's Law work for them by continually refining organizational and system architecture to facilitate the mission outcomes necessary.

Second, while agile software engineering accelerates feedback cycles to inform design decisions, it's important to understand why such fast, actionable feedback is psychologically important. For example, Reinertsen (2009) taught that people are generally wired to make attributions of causality when there is a short, elapsed time between cause and effect. For instance, if we push a button and a light quickly goes on, we subconsciously assume that our pushing the button made the light go on. That said, we do not make this association if the switch takes 5 seconds. Furthermore, when people see patterns in the consequences of their actions, they become motivated to gain even more control, such that fast feedback loops become regenerative. Instead of seeming to drift about in a vast, monolithic system, they discover a unique subdomain where they have clear autonomy and can exercise it. Unfortunately, victims often remain victims because they assume they lack control over outcomes, so they may have a steering wheel but can remain reluctant to turn it. On the other hand, when people learn that they can steer the car, they start steering the car and move towards better outcomes. Thus, fast feedback enables people to gain a sense of control, and it motivates purposive action that reinforces their sense of control.

Naturally, the most effective agile software engineering metrics are simple, relevant, and leading performance indicators—they inform clear, purposeful business decisions (Reinertsen, 2009). That said, to foster intrinsic motivation, each team should define the metrics that govern their progress and performance, for, "Psychologists have found that participation in defining goals always results in greater commitment to these goals by those who have to carry them out" (Reinertsen, 1997, p. 176). Thus, to leverage software engineering best practices while tapping into creative knowledge workers' intrinsic motivation for autonomy, mastery, and purpose (Hering, 2018), the best metrics are simple, relevant, based on leading indicators, and chosen by the team to be measured by them. These same principles apply to leveraging the OKR framework in SWP programs' VA process—ideally, the SWP program should define its own success in each VA/OKR cycle.

Third, managers should exercise caution when adding developers to ongoing development projects, for doing so often increases complexity far more than it scales productivity. In 1987, computer scientist Frederick Brooks observed that IBM's adding new programmers to a software development team did not immediately increase the capacity of said team (Hering, 2018). Rather, instead of aiding a troubled project by making it go faster, adding more people only delayed the project further: "The bearing of a child takes nine months, no matter how many women are assigned" (Hering, 2018, p. 217). This principle is now known as Brook's Law (Hering, 2018). Generally, scaling in creative knowledge work such as software engineering increases complexity significantly more than in manufacturing work, because far more parties need to communicate, information needs to be disseminated more widely, and common context needs to be created across more communication boundaries (Hering, 2018). Whether by increased context switching, cognitive overload, or some other cause, the cost of this additional complexity can be quite significant (Hering, 2018). Thus, simply adding people to linearly scale productivity, as if people were production inputs, is ineffective and often backfires. A far more effective, sustainable way to accelerate software development performance is to continually reduce underlying work system complexity by practicing DevOps (Hering, 2018).

Fourth, to improve process performance, managers should manage queues, not capacity. Within software development, each work item is generally characterized as follows: (a) highly uncertain arrival time, (b) unique tasks, (c) highly uncertain duration, and (d) unique delay costs (Reinertsen, 2009). Every software development work item invariably has highly uncertain, unique capacity requirements. Thus, attempting to monitor and control capacity is ill-advised because it leads to a game of whack-a-mole (Reinertsen, 2009). That said, because software development workflows are queuing systems, managers can exercise queuing discipline to manage their performance effectively (Reinertsen, 2009).

One of the most famous principles of queuing theory, Little's Law, proved that, on average, the more work there is in a stable queuing system, the longer it will take to complete each unit of work (Reinertsen, 2009). Furthermore, it proved the following relationship between three process performance parameters:

$L = \lambda * W$

L: Average number of work items inside the queuing system (Queue Size/WIP)

λ: Average number of work items completed in a set period (Throughput)

W: Average time a work item spends in the queuing system (Lead or Cycle Time)

(Reinertsen, 2009).

Thus, when certain conditions are met, Little's Law can be used to provide insight into and fine-tune the software development process performance, even when only two of three process parameters are known (Diaz et al., 2017). For example, if work items spend an average of 30 days in the process and average Throughput is about 1 work item/5 days, then the average Queue Size/WIP = 1 work item/5 days * 30 days = 6 work items (Diaz et al., 2017). Little's Law applies to both an entire process and its subprocesses (Reinertsen, 2009). Thus, there are many ways to apply Little's Law to the queuing system shown in Figure 61 (Reinertsen, 2009):

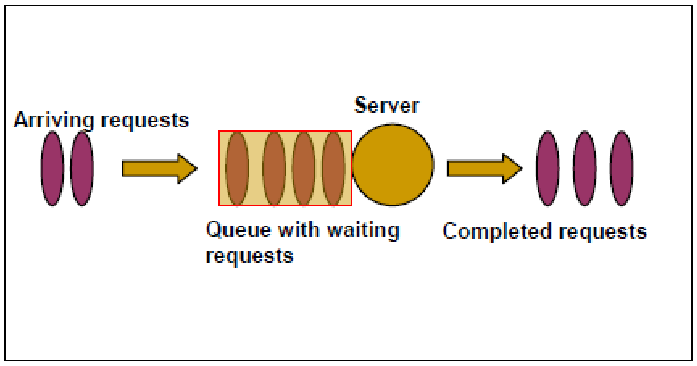


Figure 61. Generic Model of a Process Workflow. Source: Liu (2006).

As for SWP programs, each SWP program's software development process can be thought of as a queuing system: requests are work items (e.g., stories, defects, and so forth); servers are software developers; arrival rate, or demand, is the average rate at which work items enter the workflow; departure rate, or capacity, is the average rate at which work items exit the workflow; Queue Size/WIP is the average number of work items in the workflow; completed requests are implemented work items that resulted in deployed code; and Throughput is the average number of work items completed in a set period (Liu, 2006). Additionally, depending on where one defines the starting point, Lead Time and Cycle Time are the average time a work item spends in the software development process or specific subprocess(es), respectively (Liu, 2006). Typically, the starting point for Lead Time is defined as the date a new customer request is identified; the starting point for Cycle Time is defined as the date the work to complete said request begins; and the finishing points for both Lead Time and Cycle Time are defined as the date the product is delivered to the customer, as shown in Figure 62 (DeGrandis, 2017):

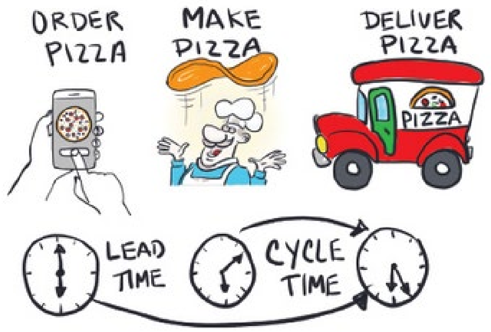
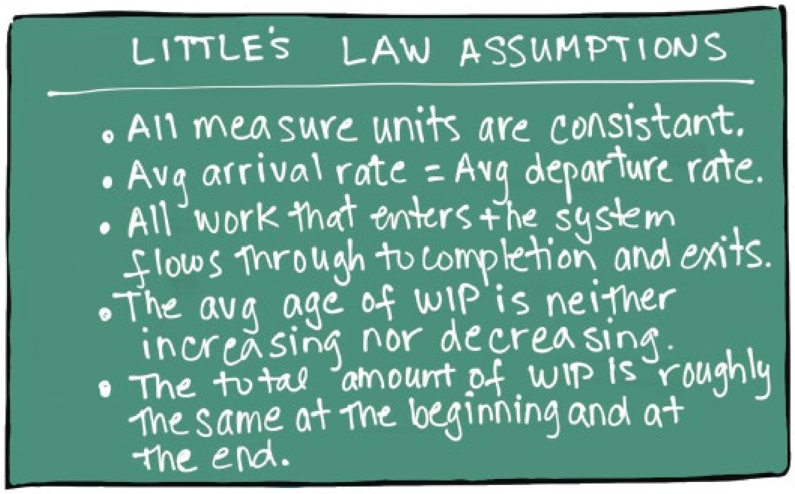


Figure 62. Cycle Time Versus Lead Time. Source: DeGrandis (2017).

Generally, as the requesters of products and/or product features, customers care about Lead Time, whereas internal teams care about the efficiency of their own subprocesses (i.e., Cycle Time; DeGrandis, 2017). But because managers must understand and optimize both Lead Time and Cycle Time, this is where Little’s Law helps (DeGrandis, 2017). To be applied, Little’s Law makes five assumptions, which are shown in Figure 63:



LITTLE'S LAW ASSUMPTIONS

- All measure units are consistant.
- Avg arrival rate = Avg departure rate.
- All work that enters the system flows through to completion and exits.
- The avg age of WIP is neither increasing nor decreasing.
- The total amount of WIP is roughly the same at the beginning and at the end.

Figure 63. Little’s Law Assumptions. Source: DeGrandis (2017).

Thus, Little’s Law applies only when the queuing system is stable, meaning that average arrival rate approximately equals average departure rate (Reinertsen, 2009). Little’s Law doesn’t account for sharp increases in work item arrival rate; when demand spikes there must already be available capacity to handle the Queue Size/WIP growth (Diaz et al., 2017). Consequently, because it is a law of averages, Little’s Law is not intended to be used for near-term planning (Diaz et al., 2017). That said, to stabilize software development workflows and to keep them as stable as possible, Little's Law implies the need to match average demand and capacity (Reinertsen, 2009). In other words, Little's Law implies to not initiate new projects or even begin new work items until current projects or work items are completed (Inthapichai, 2020). Additionally, to reduce average Lead or Cycle Time, one must increase average Throughput and/or reduce average Queue Size/WIP (Dennis, 2010).

Generally, increasing average Throughput involves significant time, effort, and/or money, for it requires allocating capital to purchase production technology and tooling, creating incremental gains through continuous process improvement initiatives, investing in education and training programs, and so forth (Inthapichai, 2020). On the other hand, reducing Queue Size/WIP can be accomplished by modifying planning policies and procedures, which can immediately affect Lead or Cycle Time at minimal cost (Inthapichai, 2020). Naturally, SWP programs should continually pursue both options. But given the dynamic nature of agile software development workflows, reducing Queue Size/WIP requires some careful considerations.

Within SWP programs, new work item arrival rate (e.g., demand) may not be entirely predictable and/or within local PMO control. Additionally, as noted, each software development work item has highly uncertain, unique capacity requirements (Reinertsen, 2009). Thus, attempting to predict and control both software development demand and capacity is impractical, if not impossible. That said, managers can reduce average Queue Size/WIP by always maintaining some marginal capacity, setting Queue Size/WIP limits, and throttling demand once Queue Size/WIP limits are reached (Kim et al., 2021). For instance, in the Kanban board shown in Figure 64, there are four In Progress work items (DeGrandis, 2017):

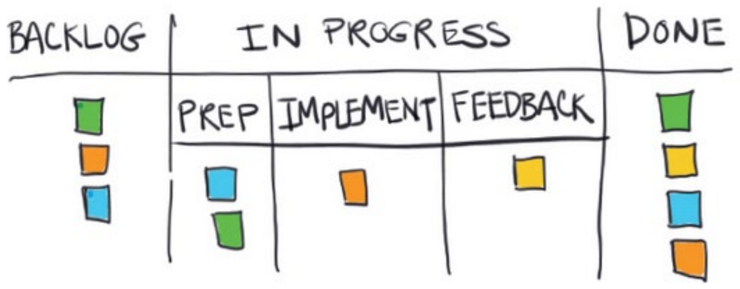


Figure 64. Kanban Board Example. Source: DeGrandis (2017).

In other words, the current Queue Size/WIP for this phase is four work items (DeGrandis, 2017). Assuming the Queue Size/WIP limit is six work items, then once there are six cards within In Progress, no new work items may be started until an In Progress card is completed and moved to done, signaling available capacity to pull a new card into development (DeGrandis, 2017). Thus, work items flow through the software development process based on the Queue Size/WIP limits and pull policies established by managers (DeGrandis, 2017).

When managers set and enforce Queue Size/WIP limits appropriately, they prevent the queuing system from becoming overloaded and enable development teams to optimally focus on completing and deploying current work (DeGrandis, 2017). Thus, managers play a critical role in balancing the flow of work with current demand, as well as enabling software developers to build and maintain flow in practicing their craft, both of which are essential to strong, sustainable software development performance outcomes (DeGrandis, 2017).

There is no formula for Queue Size/WIP limits; managers must continually monitor and assess development team needs, process performance, and so forth, to define, refine, and enforce Queue Size/WIP limits over the life of an SWP program—all of which are actions within PMO purview (Diaz et al., 2017). But while managing Queue Size/WIP limits is entirely within each SWP program's control, the most difficult aspects of reducing average Lead Time and/or Cycle Time through queueing discipline may be cultural.

In manufacturing queuing systems, work items are generally characterized as follows: (a) predictable arrival times, (b) repetitive tasks, (c) homogenous task durations, and (d) homogenous delay costs (Reinertsen, 2009). Because capacity requirements were reasonably predictable and homogenous, managers managed Lead and/or Cycle Time performance by monitoring and controlling capacity (Reinertsen, 2009). Over time, the assumptions regarding the relationship between capacity utilization and processing times have become deep-seated in management thinking and practice (Reinertsen, 2009).

Within software development queuing systems, however, controlling capacity is impractical given the highly uncertain, unique capacity requirements for each work item. Additionally, as noted, the relationship between capacity utilization and processing time (e.g., Lead Time or Cycle Time) in software engineering is nonlinear, as shown in Figure 65 (DeGrandis, 2017).

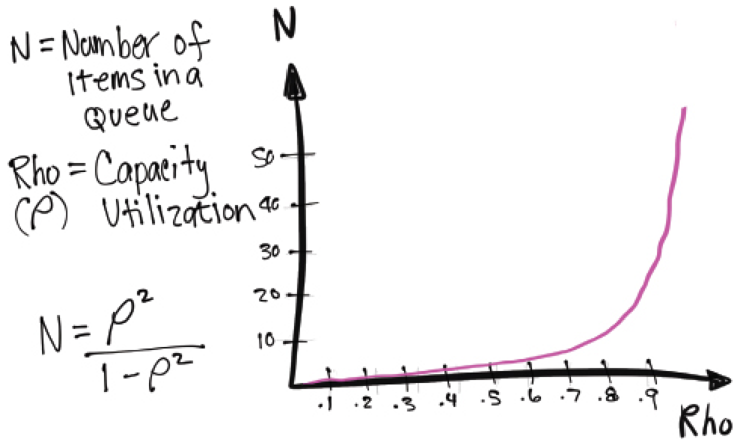


Figure 65. Processing Time Versus Capacity Utilization. Source: DeGrandis (2017).

Therefore, capacity utilization is almost a useless metric for real-time software development Lead or Cycle Time control (Thomke & Reinertsen, 2012). That said, given that Little's Law provides that Queue Size/WIP and Lead/Cycle Time are proportional, managers must shift from using capacity utilization to Queue Size/WIP as their control variable (Reinertsen, 2009). Fortunately, Queue Size/WIP are leading indicators of Lead/ Cycle Time, so monitoring Queue Size/WIP growth enables managers to detect and assess processing time risk quickly enough to intervene (Reinertsen, 2009). On the other hand, Lead/Cycle Time are lagging indicators: one cannot measure Lead or Cycle Time until work items exit the system, and the respective process or subprocess is completed (Reinertsen, 2009).

Additionally, by regularly screening new stories and grooming their Program Backlogs, SWP programs can avoid situations where they simply accumulate work items and hope it all works out during design and development. Excessive software development queues increase processing time, expenses, and risk (Reinertsen, 2009). Additionally, they slow feedback, reduce quality, and decrease motivation (Reinertsen, 2009). On the other hand, well-conditioned queues are the key to enabling and sustaining healthy software development performance (Reinertsen, 2009). Thus, to improve process performance, managers should manage queues, not capacity, primarily by limiting Queue Size/WIP and building a culture of queueing discipline.

Fifth, managers must understand the subtle yet quite significant behavioral impact of metrics. Consider the following quotes: (a) "Tell me how you will measure me, and I will tell you how I will behave" (Goldratt, 1990, p. 26); (b) "What we choose to measure is a window into our values, and into what we value. … Because if you measure something, you're telling people it matters" (Doerr, 2018, p. 220). Clearly, metrics are an incentive. Thus, how metrics are defined and utilized is critical. For instance, a common management aphorism is that "what gets measured gets done." Yet once aware their efforts are being observed and measured, people become motivated to improve—even if that entails the appearance of improvements. For instance, if someone is challenged to lose 5 lb, they could abstain from eating or drinking for several days. The person will quickly lose 5 lb this way, and fasting for this period will not be so bad due to their excitement of attaining the goal. However, what was measured did not get done—the person will have mostly reduced water weight, only to regain it after seemingly accomplishing the goal and ending their fast. This phenomenon is known as the Hawthorne Effect, which states that "that which gets measured will appear to improve" (Norton, 2020, p. 41). In other words, once people become aware they are being measured, they become motivated to attend to improvements, including the appearance of improvements. Thus, what gets measured does not necessarily get done; simply declaring a metric is insufficient. Sixth, metrics must not serve as the end but the means to the end, with clear context as to how they enable reaching a goal. As noted, the purpose of a metric is to drive decision-making towards intended business outcomes. Originally, EVM was intended to inform and enable defense acquisition decision-making. Over time, however, EVM's metrics became equated with defense acquisition program success (Patel, 2021). Consequently, both DoD PMs and defense contractors became incentivized to make EVM metrics look acceptable rather than use EVM to make good business decisions (Packaged Agile, 2020). This phenomenon is known as Goodhart's Law, which states that "when a measure becomes a target, it ceases to be a good measure. ... And the target therefore no longer means what you think it does" (Norton, 2020, p. 42). In other words, when a metric is treated not as the means but as the end, its incentives are distorted, resulting in unintended consequences. Again, the purpose of a metric is to "induce the departments to do what is good for the company as a whole" (Goldratt, 1997, p. 107). Additionally, this research noted that the most effective metrics focus teams on business goals; they focus teams on outcomes, not just outputs; and they define value in the eyes of the customer. Generally, OKRs are effective because they do all these things. Moreover, because objectives are well-defined, qualitative goals, whereas KRs are quantitative success criteria, OKRs do not violate Goodhart's Law—KRs are the means, not the end. Furthermore, because teams usually contribute to the OKR goal-setting process, the Hawthorne Effect is neutralized. Thus, while OKRs are a Silicon Valley management best practice, the strongest reason why SWP programs should use OKRs is because they are a fundamentally sound goal-setting framework. Adopting agile software engineering requires entirely new behaviors, and the most effective means to incentivize the focused, right ones for the mission are by using the OKR goal-setting framework.

Finally, because agile software engineering is principles-based, the process and practices of agile environments are predominantly shaped by the mindset, values, and principles espoused by its practitioners, as shown in Figure 66.

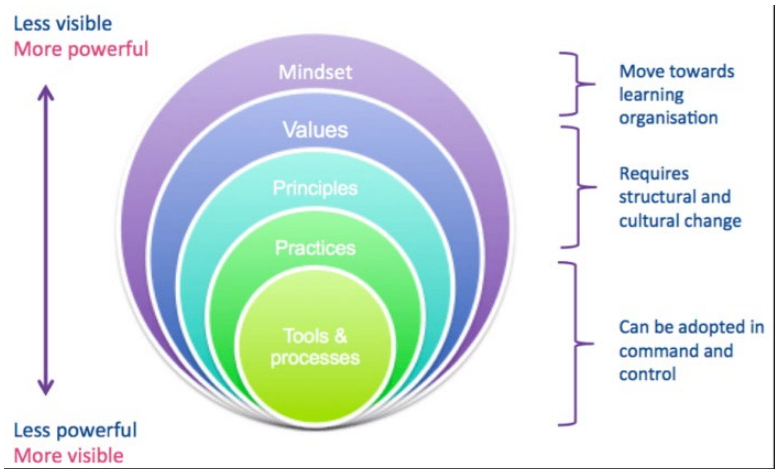

Less visible
More powerful
Mindset
Values
Principles
Practices
Tools & processes
Less powerful
More visible
Move towards learning organisation
Requires structural and cultural change
Can be adopted in command and control

Figure 66. The Cultural Factors of an Agile Environment. Source: Coyne (2020).

As noted, the Agile Manifesto itself did not prescribe a new approach to the SDLC, for its signatories acknowledged that there will always be a wide variety of technical and management practices in fast-paced, creative knowledge work such as software engineering. Instead, the Agile Manifesto envisioned better ways of work through its principles and values, empowering creative knowledge workers to reform existing and/or develop new practices as shown in Figure 67:

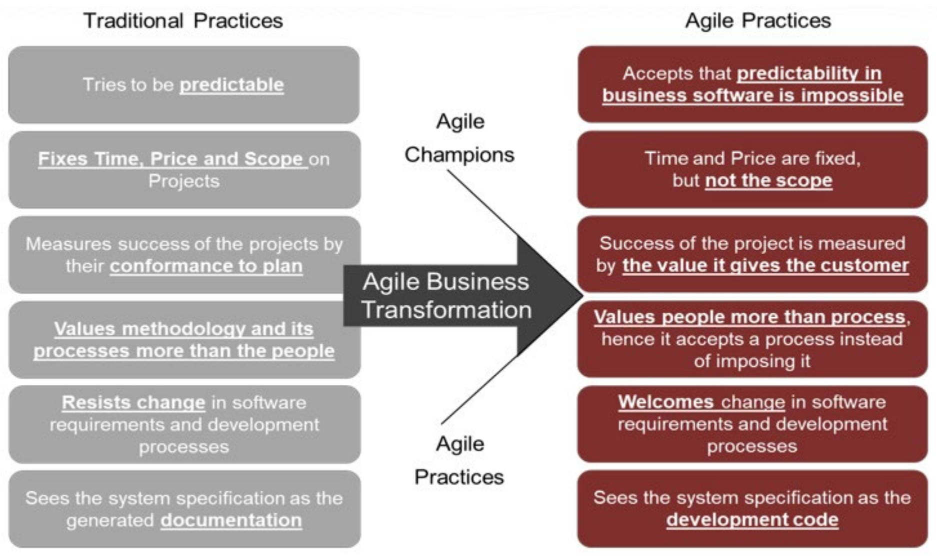
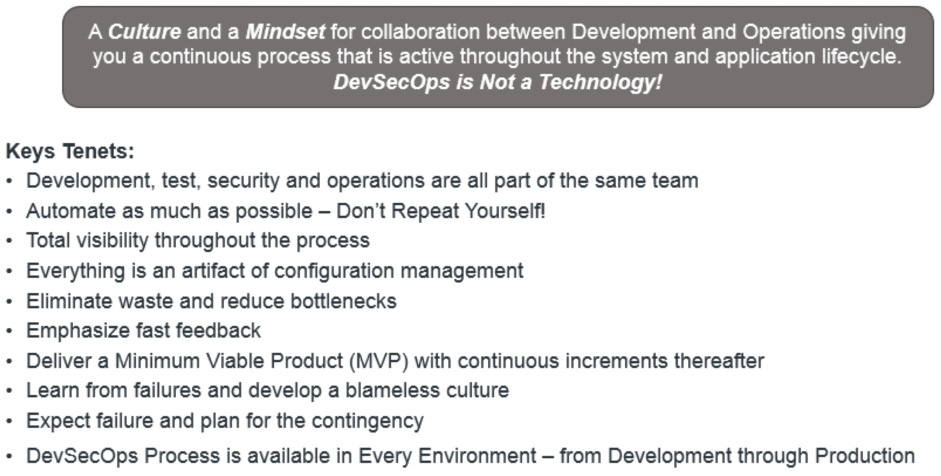

Traditional Practices
Agile Practices
Tries to be predictable
Fixes Time, Price and Scope on Projects
Measures success of the projects by their conformance to plan
Values methodology and its processes more than the people
Resists change in software requirements and development processes
Sees the system specification as the generated documentation
Accepts that predictability in business software is impossible
Time and Price are fixed, but not the scope
Success of the project is measured by the value it gives the customer
Values people more than process, hence it accepts a process instead of imposing it
Welcomes change in software requirements and development processes
Sees the system specification as the development code
Agile Champions
Agile Business Transformation
Agile Practices

Figure 67. Traditional to Agile Practices. Source: Carpenter and Carrigan (2022).

Thus, agile teams progressively moved away from waterfall approaches and converged upon concurrent software engineering practices, progressively smaller batch sizes, and so forth, all independently. Moreover, while the DevOps movement—and the DevSecOps movement in security-focused organizations such as the DoD—was primarily driven by technological breakthroughs that enabled integrating the IT ecosystem, its key tenets are nonetheless driven by the Agile Manifesto as shown in Figure 68:



A ***Culture*** and a ***Mindset*** for collaboration between Development and Operations giving you a continuous process that is active throughout the system and application lifecycle. ***DevSecOps is Not a Technology!***

Keys Tenets:

- Development, test, security and operations are all part of the same team
- Automate as much as possible – Don't Repeat Yourself!
- Total visibility throughout the process
- Everything is an artifact of configuration management
- Eliminate waste and reduce bottlenecks
- Emphasize fast feedback
- Deliver a Minimum Viable Product (MVP) with continuous increments thereafter
- Learn from failures and develop a blameless culture
- Expect failure and plan for the contingency
- DevSecOps Process is available in Every Environment – from Development through Production

Figure 68. Key Tenets of DevSecOps Practices. Source: Carpenter and Carrigan (2022).

Thus, culture is the most critical determinant of successfully practicing agile methods. Used the wrong way, the Agile Manifesto could just as easily be used to reinforce dogmatic attitudes. When used with the right intentions to intentionally develop culture, however, the Agile Manifesto is the most powerful tool in any DON SWP program's toolbox. To make the most of this tool, DON SWP programs should continually design and refine their process and practices with the Agile Manifesto's values and principles in mind.

F. SUMMARY

This chapter included a discussion of several methods and tools to manage software-intensive acquisition programs, all of which are tailored to manage the emergent requirements, iterative development and incremental delivery processes, and continuous user feedback practices of agile environments. To use these methods and tools effectively, managers need to adopt fundamentally new ways of thinking about development project progress and performance. Agile environments do not conduct long-range, detailed planning; they do not comprehensively estimate and pre-allocate all necessary resource requirements; and both the design and value of the design remain emergent. Thus, program managers cannot rely upon plan-driven oversight methods and tools that were designed to maximize conformance and efficiency of all planned technical work.

Within agile environments, short-term planning is conducted only to the extent necessary to initiate the first software development cycle; estimation and budgeting are performed just-in-time for the sake of allocating iteration capacity; and both the design and value of the design are continually validated and refined throughout the development project. By rapidly and iteratively delivering software and capturing continuous feedback, agile environments are fundamentally structured around a process that enables building the most valuable product possible through all remaining development cycles of a project. Therefore, to monitor and manage progress and performance in agile environments effectively, program managers need to adopt methods and tools that span not necessarily the plan, but the software development process. As noted, development project plans are merely a proxy for the intended product, whereas agile environments iteratively develop and incrementally expose the emergent product.

Accordingly, the agile movement has created a wide variety of management tools and methods to capture the health of the software development process and to assess the value of the product as it is being built. Fortunately, there are many commercially available ALM and software engineering tools that automatically generate progress and performance information. The most common charts used to monitor software development productivity and progress are burn-down charts, burn-up charts, and CFDs. Additionally, agile environments commonly track the rate and count of escaped defects to production as well as the level of test coverage to detect and contain quality issues, continuously build quality into the development process, and ensure the codebase is architected in a manner that enables automated testing to the greatest extent practicable. To reliably forecast and commit to delivery dates, the agile movement has also created forecasting tools that enable probabilistically determining schedule outcomes based on their rate of implementation.

This chapter also included an analysis of the interactions between EVM and agile software engineering, showing that they fundamentally conflict in four ways. Agile environments plan work tasks as an LOE; they continually validate the value of work through customer engagement; they do not comprehensively identify, estimate, and organize all work using a WBS; and they maximize value by building the best product possible using all remaining development cycles. While many have attempted to truncate planning horizons and adapt EVM to agile planning processes and practices, agile environments and EVM fundamentally incentivize maximizing value in a manner that is irreconcilable. Therefore, SWP programs should not use EVM to monitor acquisition program progress and manage performance.

The researcher also evaluated the metrics and performance management–related recommendations of all prior DoD software acquisition studies, which gradually evolved from assuming one large-batch custom software delivery to an iterative, incremental process that explicitly acknowledges that software development is never done. The metrics recommended in the DIB SWAP study's *Metrics for Software Development* supplement are aligned to leading commercial software development practices and establish several useful performance standards for DoD SWP programs. In evaluating recommendations to streamline acquisition, the researcher also noted that the Section 809 Panel proposed exempting EVM and EVMS requirements for all software-intensive acquisition programs that use agile methods, but it also proposed EVM-like planned versus actual cost and schedule metrics that could potentially undermine its findings.

Finally, to make the most of the software engineering management know-how highlighted in this project, this chapter included a discussion of several principles and patterns to provide some of the supporting know-why. Because organizational structure and software architecture are intimately related, SWP programs must continually ensure that they organize themselves and design their tech stack in a manner that enables, rather than constrains, optimal flow of valuable software. Additionally, instead of seeking to linearly scale productivity by adding more developers, managers should continually improve software development performance by progressively building a more resilient and responsive system of technology work through DevOps and by balancing average demand and capacity via strictly enforced Queue Size/WIP limits. In doing so, managers have a direct role in creating a culture that values finishing and delivering current work before committing to a new workload. Additionally, because metrics function as incentives, they produce the strongest intended effects when they clearly define a means to accomplish desired goals and are designed, even if in part, by the teams that will be measured by them.

Most importantly, the agile movement has few standardized technical or managerial practices, so how SWP programs internalize the Agile Manifesto and leverage it to co-create a high-performance culture of empowerment, continuous learning, and continuous improvement is a critical determinant of performance. For instance, when misused, SWP programs could opportunistically cite select Agile Manifesto phrases such as "maximizing the amount of work not done" (Beck et al., 2001, para. 10) to avoid existing governance methods and tools altogether and continually seek shortcuts. On the other hand, a truly agile culture may set aside time to regularly refactor existing governance, risk, and compliance processes and practices, treating it as no less essential than reducing technical debt in the codebase to improve engineering performance. Overall, DON SWP program managers can lead by example by both enabling developers to continuously deliver high-quality software to operations, as well as motivating everyone in the agile environment to continually build a highly agile, secure, and reliable system of technology work.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

This chapter includes a synopsis of the research project's findings, limitations with respect to its conclusions, and final recommendations to potentially advance the management practices of DON SWP custom application programs where necessary.

A. SYNOPSIS

The software engineering process—the continuous process of planning, designing, developing, integrating, testing, deploying, and operating software—has become the most critical means of creating and delivering value in the Information Age economy (Kersten, 2018). Despite its industrial preeminence, however, the profession of software engineering is ironically still a very young field (Forsgren & Kersten, 2018).

Unlike the manufacturing domain, for which managers have developed highly refined performance management methods, metrics, and operational data collection tools over the past century, the software engineering industry lacks clear consensus on how to measure the software engineering process (Forsgen & Kersten, 2018). Furthermore, the most effective software practices are fundamentally context-dependent and temporary, as new breakthroughs in digital technology will inevitably necessitate new ways of software practice. As such, few standardized software practices exist, and just as state-of-the-art digital technology evolves, the software practices that enable effective digital technology adoption must also continually evolve.

Given that the DON primarily buys, not builds, software, this is predominantly a managerial challenge. The DON already recognized that software acquisition is a dynamic, indefinite process, and it implemented the DoD SWP to enable rapid and iterative delivery of custom software. However, the SWP merely provides the policy framework to acquire custom software. To effectively develop and deliver cutting-edge custom software capability, each DON SWP program still needs to experientially learn and adopt the management principles, methods, and tools suited to modern software practice. Moreover, overcoming this challenge requires DON SWP programs to integrate management beliefs, methods, and tools that, for the most part, neither the DoD nor the

DON created or influenced. Unfortunately, this is a very heavy lift for each DON SWP program.

The explicit intent of the DoD SWP is to facilitate rapid and iterative delivery of software capability the user, so it clearly directs what DON SWP programs should do. But because software engineering is creative knowledge work and the modern management principles, methods, and tools to practice it have predominantly originated outside the defense industry, this research questioned why and how DON SWP programs could successfully adopt an agile approach. Specifically, to enable DON SWP programs to adopt agile methods as effectively as possible, the researcher investigated the currently most effective agile metrics; management tools; and software engineering and digital product development practices. Additionally, to reconcile these commercially driven management practices with defense acquisition program governance, the researcher also evaluated the utility of EVM and the recommendations of prior DoD software acquisition reform studies.

In the end, development project plans have always functioned as a proxy for the intended product (Perri, 2018). Thus, prior to the agile movement, both DoD and commercial industry developed and delivered software via waterfall projects, and they used proxy metrics to measure conformance and efficiency against the development project plan. Generally, these proxy metrics were backward facing.

In agile environments, however, the product is iteratively developed and incrementally delivered and refined through continuous customer and/or end user feedback. Thus, to measure and steer progress and performance, agile environments tend to use forward facing, not backward facing, metrics. Furthermore, instead of proxy metrics, agile environments monitor and manage performance using product-based metrics that measure the functional and nonfunctional attributes of the emergent product, as well as process-based metrics that capture the health of the end-to-end software development process.

Overall, based on the conclusions of this research, DON SWP programs should rapidly and iteratively deliver cutting-edge custom software by using a combination of project, product, and process metrics and a phased approach to measure their progress and performance.

B. RESEARCH FINDINGS

The primary goal of this research was to propose a management framework for DON SWP custom application programs by answering the following four questions:

1. What metrics should the DON use to assess agile/incremental program performance?
2. What are the leading management tools, monitoring and control methods, and practices to track and review progress and performance of a software acquisition program?
3. Should EVM be replaced or augmented as the standard for program performance?
4. What are the metrics being recommended by the DSB, DIB Software Acquisition and Practices Study, and Section 809 Panel?

To establish the scope of the software acquisition management framework, the research project made three assumptions: (1) an approved DON SWP custom application program is beginning its Execution Phase; (2) the DON SWP program is acquiring custom application software running on commercial hardware/operating system platform (i.e., Type C software); and (3) the DON SWP program's software development, assurance, deployment, and operations activities are primarily performed by contractors. Now, the research project's questions are answered in order.

(1) What metrics should the DON use to assess agile/incremental program performance?

DON SWP programs should implement a phased approach using balanced metrics to assess their performance. When software development activities begin, the DON SWP program should use EBV to track and assess progress towards the MVP, velocity-based metrics to track team productivity, and automated test coverage to track quality of the codebase.

Once MVP is built and its features are demonstrated to intended users in a testing environment, the DON SWP should use product-based metrics to track and assess progress against features planned for the MVCR, velocity-based metrics to track team productivity, automated test coverage to track quality of the codebase, and the rate and count of defects to track quality of the software development process. To the greatest extent practicable, the DON SWP should demonstrate newly completed features to customers and/or users in production-like testing environments at the end of each iteration/sprint, noting and incorporating feedback on new functionality to ensure that the MVCR delivers the highest priority software capability to operations.

Once the MVCR is deployed, the DON SWP should use VA metrics to track and assess value in each VA cycle, the four DORA metrics—Lead Time for Changes, Deployment Frequency, Change Failure Rate, and MTTR—to holistically track and assess software delivery performance health, automated test coverage to track quality of the codebase, and the rate and count of escaped defects to production to track and assess quality of the software development process.

To drive and sustain mission focus, DON SWP programs should always treat value-based metrics as the most important measures of program progress and performance. Thus, EBV progress metrics in the pre-MVP phase, product-based metrics in the pre-MVCR phase, and VA metrics in the post-MVCR phase should always be prioritized over all other progress and performance indicators. DON SWP programs are advised to use special titles or phrases (e.g., North Star metric, One Metric That Matters, and so forth; Ries, 2017) to create cultural norms around this practice.

Finally, to implement clear, consistent performance standards based on software acquisition best practices, DON SWP programs should use the metric standards for Type C software established in the DIB SWAP study's *Metrics for Software Development* supplement (DIB, 2019a).

(2) What are the leading tools, monitoring and control methods, and management practices to track and review software acquisition program progress and performance?

Instead of comparing planned versus actual work activities and resource expenditures to conduct program governance, agile environments use automated ALMs tools to visualize and dynamically track the flow of work in the software development process. The most common agile charts used to do this are burn-down charts, burn-up charts, and CFDs. DON SWP programs should use burn-down charts to track productivity, burn-up charts to track progress, and CFDs to understand and manage the health of the software development process.

Due to the stochastic nature of the software development process, DON SWP programs must also understand the statistical distribution of potential schedule outcomes to reliably forecast and make delivery date commitments. To do so, the DON SWP should use the Throughput Forecaster or other Monte Carlo simulation-based agile forecasting tools to probabilistically estimate their schedule outcomes.

Next, because software development work items have highly uncertain, unique capacity requirements, managers should not attempt to monitor and control capacity utilization, for it is highly impractical. Instead, managers should monitor and control queues. Given that software development is an indefinite process, it is a queueing system to which Little's Law applies. Little's Law proved that, on average, the more work there is in a stable queuing system, the longer it will take to complete each unit of work. Thus, to match the average flow of demand with that of capacity, Little's Law implies that DON SWP programs should establish a culture of not initiating new work items until a current one is completed—hence the attitude to stop starting and start finishing within the high-tech industry (Kim, 2019). Additionally, because Little's Law proved that Queue Size/WIP and Lead or Cycle Time are proportional, DON SWP programs should establish Queue Size/WIP limits that temporarily throttle demand once they're reached. There are no set formulas for determining these limits. Each DON SWP program will have to continually set, monitor, and recalibrate its Queue Size/WIP limits to maintain optimal software development flow throughout the life of the DON SWP program.

Finally, to manage longer-term software acquisition outcomes, DON SWP programs should use the Flow Framework,® for it provides a MECE governance structure to holistically monitor and manage all functional, nonfunctional, quality, and noncritical yet important software development requirements that impact DON SWP outcomes. Without a comprehensive tool such as the Flow Framework,® DON SWP programs will neither be able to plan software architecture upgrades, technical debt reduction, and so forth, nor trade off between capability development and internal process improvement work effectively.

(3) Should EVM be replaced or augmented as the standard for program performance?

The software engineering industry has not yet converged upon the most effective performance metrics for iterative, incremental development (i.e., agile methodologies; Maddox & Walker, 2021). Thus, a standardized tool for managing cost and schedule performance in agile environments, which could potentially replace EVM as the standard for acquisition program governance, is not available. Nevertheless, DON SWP programs should refrain from using EVM, for agile software engineering methods and EVM fundamentally conflict in four ways: (a) agile environments plan all work tasks as a LOE, whereas EVM requires minimizing LOE use; (b) agile environments valuate work product based on stories derived through customer engagement, whereas EVM valuates work product based on CSCIs derived from the WBS; (c) agile environments have no universal framework to identify and organize all technical work content, whereas EVM uses a standardized WBS to delineate and control all work scope; and (d) agile environments motivate building and maximizing value via an iterative, incremental process, whereas EVM motivates maximizing value by minimizing resource consumption en route to a predetermined design.

These four conflicts do not suggest that EVM itself is an ineffective acquisition program management tool; rather, they are attributable to the extreme uncertainties of the design and value of the design in cutting-edge software development projects which make EVM implementations intractable. Therefore, DON SWP programs should not use EVM because it is not feasible. Fortunately, recent research has shown that simply practicing agile methods has enabled DoD software development programs to save at least 15% in total labor costs and 20% in total schedule compared to those practicing waterfall methods (Patel, 2021). As a result, DON SWP programs may not even need to implement EVM to control cost and schedule growth. To track schedule performance using alternative methods, DON SWP programs should use Monte Carlo simulation-based forecasting tools noted in this project to probabilistically estimate and manage schedule performance. As for tracking cost performance for its own sake, development effort (i.e., labor) is generally the largest cost element associated with developing custom software (Nichols et al., 2022). Thus, development labor is the best variable by which

DON SWP programs can estimate cost (Nichols et al., 2022). Since agile environments plan all work tasks as an LOE, DON SWP programs can estimate their cost performance by calculating total development labor expenses multiplied by elapsed time duration (Nichols et al., 2022).

Overall, EVM can neither be replaced nor augmented as the standard management framework for defense acquisition program performance. EVM should continue to be used to contain and control cost growth, particularly in capital-intensive weapon system acquisition programs where the risk of cost growth resides with the government. That said, EVM is incompatible with DON SWP programs, and EVM techniques are not necessary to monitor their cost and schedule performance. DON SWP programs should manage performance using the agile metrics, tools, and methods highlighted in this project and utilize ad hoc techniques to monitor cost and schedule performance only as needed.

(4) What are the metrics being recommended by the Defense Science Board, Defense Innovation Board Software Acquisition and Practices Study, and Section 809 Panel?

In 2018, DSB issued a software acquisition report acknowledging that designing and managing DoD software acquisition programs with waterfall-oriented and acquisition life cycle–based metrics created “a misalignment between the DoD’s processes and the reality of contemporary industry practices” (OUSD[R&E], 2018, p. 21). Instead, the DSB’s 2018 report recommended customizing the governance framework for each software acquisition program, and it highlighted commonly used agile metrics such as velocity to track productivity of individual software development teams, as well as agile management tools such as iteration/sprint burn-down charts and CFDs to track the health of the software development process (OUSD[R&E], 2018). The 2018 DSB report also suggested that adopting agile methods would lead to long-term cost savings, and none of its highlighted metrics and management methods included tracking costs (OUSD[R&E], 2018).

To provide the flexibility necessary for iterative, incremental development, the Section 809 Panel (2018) recommended updating DFARS 234.201 and OMB Circular A-

11 to exempt all DoD custom software development and integration contracts practicing agile methods from EVM and EVMS requirements, regardless of contract type or total dollar value. Instead of EVM/EVMS, the Section 809 Panel (2018) recommended authorizing PEOs to approve the appropriate project monitoring and control methods for such contracts. However, the Section 809 Panel (2018) also recommended using, at a minimum, planned versus actual schedule, planned versus actual cost, and estimate to complete metrics without providing non-EVM techniques to implement them. As a result, the Section 809 Panel's (2018) recommended metrics are ambiguous and potentially undermine its overall findings regarding the incompatibility of agile methods and EVM/ EVMS. Indeed, EVM and EVMS requirements should be waived for all DoD custom software development and integration contracts practicing agile methods, because neither EVM nor EVMS can be reliably implemented in cutting-edge software development projects where both the design and value of the design are extremely uncertain. But to avoid the use of EVM techniques altogether, DON SWP programs should use Monte Carlo simulation techniques and software development labor costs multiplied by total elapsed time to, respectively, track and forecast schedule and cost performance on an ad hoc basis.

Finally, in 2019, the DIB released its SWAP study, the DoD's most comprehensive software acquisition reform effort to date. The DIB SWAP study explicitly assumes that software development is a continuous process, and its *Metrics for Software Development* supplement proposed 14 state-of-the-art metrics to manage software acquisition programs and drive improvement in cost, schedule, and performance. Crucially, the *Metrics for Software Development* (DIB, 2019a) supplement also proposed several performance standards based on the type of software and computing infrastructure utilized. To align to current best practices, DON SWP custom application programs running on commercial computing infrastructure and operating systems (i.e., Type C software) should adopt the metrics and performance standards shown in Table 9:

Table 9. Metrics for DON SWP Custom Application Programs. Adapted from DIB (2019a).

Metric Type	Metric	Performance Standard
Deployment Rate	• time from program launch to deployment of simplest useful functionality • time to field high priority functionality; find and fix security issue • time from code committed to code in use	• ≤ 6 months • ≤ 3 months; ≤ 1 week • ≤ 1 day
Response Rate	• time required for regression testing; cybersecurity audit/penetration testing • time required to restore service after outage	• ≤ 1 day; ≤ 1 month • ≤ 1 day
Code Quality	• automated test coverage of code • number of bugs caught in testing versus field use • change failure rate (e.g., required rollback) • percentage of code available for DOD to inspect/rebuild	• > 90% • > 75% • ≤ 10% • 100%
Functionality	• number/percentage of functions implemented • usage and user satisfaction	• 70% • N/A
Program Management, Assessment, and Estimation	• complexity metrics • development plan/environment metrics	• N/A • N/A
Progam Progress	• software development-based Nunn–McCurdy thresholds	• 1.5X

The time required to restore service after an outage is the same as MTTR.

The number of bugs caught in testing versus field use is the same as the number of escaped defects to production.

By adopting these metrics and performance standards to the greatest extent practicable, DON SWP custom application programs will maximize their abilities to make clear, consistent software engineering decisions; rapidly adopt commercially driven digital technology innovation; continuously improve cost, schedule, and performance; and continuously deliver high-priority software capability at the speed of relevance.

C. RECOMMENDATIONS

Based on the findings of this research, there are several ways the DoD, DON, and/or DON SWP programs could potentially improve their software acquisition practices. First, to effectively manage uncertainty in agile environments, motivate superior performance, and reinforce cultural norms around agile methods, DON SWP programs should formulate and assess their objective goals using the OKR goal-setting framework. As shown in Chapter III, integrating OKRs into the VA process only requires slight adjustments to the VA template. Furthermore, as several leading high-tech companies have shown, OKRs and agile software engineering complement each other by focusing teams on mission outcomes and continually driving improvements and growth (Wodtke, 2016).

Second, the AAF guidance for the DoD SWP should be updated to clarify when its required metrics are applicable. Presently, SWP programs are required to track and semiannually report the following 12 metrics as soon as they enter the DoD SWP: (a) Average Lead Time for ATO, (b) Continuous ATO In-Place, (c) Mean Time to Resolve Experienced Cyber Incident or CVE, (d) Mean Time to Detect Cyber Incident, (e) Average Deployment Frequency, (f) Average Cycle Time, (g) Average Lead Time for Change, (h) Minimum Lead Time for Change, (i) Maximum Lead Time for Change, (j) Change Fail Rate, (k) MTTR, and (l) VA Rating (OUSD[A&S], n.d.-g).

However, metrics e–k are inspired by the benchmark DORA metrics used to assess technology organizations, which DORA has specifically defined as software delivery performance metrics in its literature (Forsgren et al., 2018). Naturally, software delivery performance metrics do not apply until after SWP programs have initially delivered their software to operations. Furthermore, the AAF guidance for the DoD SWP should be updated to clarify that VA Rating reporting is not required until after SWP programs have completed their first VA cycle. Given that SWP programs have up to one year after MVCR to complete their first VA, this could potentially take up to two years from the date software development activities are funded. Thus, tracking and reporting VA Ratings at the very onset of SWP programs is impractical.

Third, the DON SWP should be updated to rescind its mandatory performance metrics. Presently, DON SWP programs are required to track the following software delivery performance metrics: "(1) Average Deployment Frequency; (2) Average and Minimum/Maximum Lead Time to commit code to production; (3) Average Cycle Time; (4) Change Failure Rate" (ASN[RD&A]), 2022, pp. 6–7). However, because these metrics overlap with metrics e–j that are already required by the DoD SWP, the DON SWP's metrics are redundant.

Fourth, the DoD SWP guidance should be updated to clarify that the DORA metrics—Lead Time for Changes, Deployment Frequency, Change Failure Rate, and

MTTR—are intended to be used holistically, not individually. As noted in Chapter II, the purpose of the DORA metrics is to unify the goals and incentives of software developers (i.e., Dev), who are charged with delivering new features, and IT operations engineers (i.e., Ops), who support existing service offerings and infrastructure (Kim et al., 2021). The first two DORA metrics, Lead Time for Changes and Deployment Frequency, provide insight into the velocity of the software development process and how responsive it is to users' evolving needs, whereas the last two DORA metrics, Change Failure Rate and MTTR, indicate how stable the provided services and responsive the technology organization are to production incidents (Forsgren et al., 2018). By measuring these four metrics together and widely radiating current performance levels, the DORA metrics eliminate the false choice between velocity and stability in the software delivery process, specifically by motivating a run what you build mentality (Kim et al., 2021). When used together and visibly measured, the DORA metrics also rally the entire technology organization around continuous improvement of its culture, architecture, and technical practices, as envisioned by the DevOps movement (Kim et al., 2021). Clearly, to facilitate these behavior outcomes, the DORA metrics must be used holistically, but the DoD SWP does not indicate that these software delivery performance metrics must be tracked and assessed together.

Fifth, DON SWP programs should consider incorporating the Flow Framework® to adopt a MECE management framework to measure and manage all types of software development work: features, defects, risks, and debt (Kersten, 2018). Presently, the DoD SWP only alludes to the importance of planning work to reduce technical debt and/or upgrade software architecture. However, the DoD SWP does not provide metrics to track nonfunctional requirements and/or work items to refactor the codebase. Moreover, neither the DoD SWP nor the DON SWP provide management methods to trade off between capability development and internal process improvement work. Without such management tools and methods, DON SWP programs will be ill-equipped to responsibly manage longer-term software acquisition outcomes. The SWP was issued with the intent to facilitate continuous software acquisition and rapid and iterative delivery of software capability to the user. To do so effectively and sustainably, DON SWP programs should adopt the software engineering framework that was specifically designed for continuous software product development, the Flow Framework®.

Six, DON SWP programs should consider implementing release management strategies to help optimally manage risk via smaller, more frequent, and/or more controlled timing of feature releases. For example, instead of deploying new features to their entire userbase, DON SWP programs may replicate their production environment, direct all traffic to one server via a load balancer, deploy new features to the other server, and then can incrementally shift user sub-groups to the newer software version via a technique called canary deployments (Harrison & Lively, 2019). Canary deployments reduce risk by adopting a phased approach to software deployment and enabling rapid rollback if initial user sub-groups report critical errors for new features. Additionally, by decoupling the software deployment process from software releases, DON SWP programs can implement dark launching, whereby new features are deployed to real production environments and released to targeted user sub-groups via feature flags (Harrison & Lively, 2019). Dark launching substantially reduces risk by remotely enabling or disabling new features almost immediately without having to redeploy code. These are just some release management strategies that can enable DON SWP programs to avoid large-scale software deployment failures, such as a bad MVCR. Simply put, not every software deployment must result in software release, and not every software release must be made to the entire userbase.

Finally, the AAF guidance for the DoD SWP should be updated to clarify define and distinguish between a program manager, product manager, and product owner with respect to planning and executing SWP programs. Within the commercial software engineering industry, a product manager is typically a member of the development organization primarily responsible for specifying, testing, shipping, and optimizing a product within specified budget, schedule, and performance constraints (Moore, 2014), whereas a product owner is a team-level PMO leader within the Scrum agile software development framework (Perri, 2018). Thus, product managers tend to focus more on creating and managing a product roadmap to meet strategic business goals, whereas product owners tend to focus more on creating and managing a product backlog in direct coordination with developers. However, according to the AAF, the product owner is a member of the operational or requirements organization (OUSD[A&S], n.d.-h), works with the PMO to develop and manage the product roadmap (OUSD[A&S], n.d.-d), and works with the PMO to develop and manage the program backlogs (OUSD[A&S], n.d.-d). Consequently, the AAF guidance defines the product owner as a key external stakeholder for each SWP program, not as a team-level PMO leader ala the Scrum framework. This is a significant cultural difference and may be very confusing to Scrum and other agile practitioners partnering with the DoD. Given that commercial product managers are much more externally focused than commercial product owners, the AAF guidance for the DoD SWP should redefine the product owner role as a product manager to both reduce ambiguity and better align SWP programs to commercial software industry practices.

D. LIMITATIONS

This research project and its conclusions have some limitations. First, the researcher did not investigate whether new software estimation methodologies have been developed specifically for agile project estimation. Accordingly, the results of this research are not necessarily applicable for estimating a software acquisition program. Second, the results of this research are limited to the acquisition of custom software running on commercial hardware/operating systems under the DON's implementation of the DoD SWP (i.e., Type C software as defined by the DIB SWAP study). The results of this research are not necessarily applicable to DON embedded system software development programs or to other DoD component SWP programs. Third, because the DON SWP was issued in April 2022, there were limited opportunities to research the management practices of current Type C DON SWP programs. To strengthen the findings of this research project, and potentially generate greater benefit for the DON, the researcher recommends conducting a case study on a current Type C DON SWP program so that the metrics and management practices recommended herein can be tested, validated, and potentially refined. Finally, while not necessarily a limitation, all identification of commercial companies, and the specific technologies developed by such commercial companies, throughout this research project occurred strictly for academic purposes. No DoD or DON endorsement of these private entities or their technologies is intended whatsoever.

E. SUMMARY

The DoD helped invent the digital computer and the Internet, and it influenced the early development of the software engineering profession (Mahoney, 1990). However, the DoD neither created nor influenced the agile software engineering movement, whereas since the agile movement started, software has become the most critical element of Information Age technology and technology development processes (DIB, 2019b). Just as the most successful businesses deliberately adapted their operations and management practices around software development, each DON SWP program must now lead digital transformation (i.e., increase its PMO's adoption of software technology and modern software practices) by adapting agile principles, values, and practices throughout all aspects of project management, digital NPD, and software acquisition program governance to exploit the full potential of the SWP. Naturally, given that the SWP's vision is continuous software acquisition, no research could provide all the answers. Fortunately, however, agility is already ingrained in the military's DNA, and the researcher aimed to highlight how and why DON SWP programs can begin leveraging modern software engineering management practices to potentially increase their business agility effectively.

APPENDIX. MISCELLANEOUS

The Flow Framework® is a framework created by Mik Kersten, CEO of Tasktop Technologies Incorporated (“Tasktop”). The Flow Framework® diagrams, images, graphics and other materials referenced herein in relation to the Flow Framework® is protected by copyright laws and may not be copied, modified or distributed without the express written permission of Tasktop. Tasktop® and the Flow Framework®, Flow Efficiency®, Flow Velocity®, Flow Distribution® and Flow Load® are trademarks of Tasktop Technologies Incorporated.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Abba, W. F. (2017, March 1). *The evolution of earned value management*. Defense Acquisition University. https://www.dau.edu/library/defense-atl/blog/Defense-ATandL--March%E2%80%93April-2017-2-The-Evolution-of-Earned-Value-Management

Anderson, R. E. (1993). Can stage-gate systems deliver the goods? *Financial Executive*, *9*(6), 34–38.

Andreessen, M. (2011). Why software is eating the world. *The Wall Street Journal*, C2.

Assistant Secretary of the Navy for Research, Development, and Acquisition. (2022, April 8). *Department of the Navy implementation of the defense acquisition system and the adaptive acquisition framework* (SECNAVINST 5000.2G). Department of the Navy.

Bahcall, S. (2019). *Loonshots: How to nurture the crazy ideas that win wars, cure diseases, and transform industries*. St. Martin's Press.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. https://agilemanifesto.org/

Bell, T. E., & Thayer, T. A. (1976, October). Software requirements: Are they really a problem? In *Proceedings of the 2nd International Conference on Software Engineering* (pp. 61–68).

Bezos, J. (2017, April 17). *2016 letter to shareholders*. Amazon.

Blank, S. (2013). Why the lean start-up changes everything. *Harvard Business Review*, *91*(5), 63–72.

Brooks, F., Basili, V., Boehm, B., Bond, E., & Eastman, N. (1987). *Report of the Defense Science Board Task Force on military software*. Office of the Under Secretary of Defense for Acquisition.

Burns, K. 2017. *Story writing & mapping* [PowerPoint slides]. https://www.slideshare.net/KevinBurns66/story-writing-and-mapping

Carpenter, T., Jr., & Carrigan, C. (2022, March 2). *Let's talk agile webinar: Defense Counterintelligence and Security Agency—Using agile to protect our nation's critical assets*. Defense Acquisition University. https://www.dau.edu/event/Let%27s-Talk-Agile-Webinar-DCSA-Using-Agile-to-Protect-Our-Nations-Critical-Assets

Ching, C. (2015). *Rolling rocks downhill*.

Conway, M. E. (1968). How do committees invent. *Datamation*, *14*(4), 28–31.

Cooper, R. G. (1990). Stage-gate systems: A new tool for managing new products. *Business Horizons*, *33*(3), 44–54.

Coyne, J. (2020). *An agile mindset—Use these simple statements to challenge yourself. https://medium.com/@justincoyne.nemean/an-agile-mindset-use-these-simple-statements-to-challenge-yourself-8bc166ede32*

Defense Federal Acquisition Regulation Supplement, 48 C.F.R. ch. 2 (2022).

Defense Innovation Board. (2019a). *Defense Innovation Board metrics for software development* (SWAP Study Concept Paper, S82). https://media.defense.gov/2019/May/02/2002127284/-1/-1/0/defenseinnovationboardmetricsforsoftwaredevelopment.pdf

Defense Innovation Board. (2019b). *Software is never done: Refactoring the acquisition code for competitive advantage*. https://media.defense.gov/2019/Apr/30/2002124828/-1/-1/0/softwareisneverdone_refactoringtheacquisitioncodeforcompetitiveadvantage_final.swap.report.pdf

Defense Science Board. (2009). *Report of the Defense Science Board Task Force on Department of Defense policies and procedures for the acquisition of information technology*. https://www.hsdl.org/?abstract&did=36935

DeGrandis, D. (2017). *Making work visible: Exposing time theft to optimize work & flow*. IT Revolution.

Dennis, P. (2010). *The remedy: Bringing lean thinking out of the factory to transform the entire organization*. John Wiley and Sons.

Department of Defense. (1985, June 4). *Defense system software development* (DoD-STD-2167). https://quicksearch.dla.mil/Transient/0ECB39822A1A4608B38E21894394BCB0.pdf

Department of Defense. (2019, March 14). *Earned value management system interpretation guide*. https://www.acq.osd.mil/asda/ae/ada/ipm/docs/dod_evmsig_14mar2019.pdf

Department of Defense (2022, May 13). *Work breakdown structures for defense materiel items* (MIL-STD-881F). https://quicksearch.dla.mil/Transient/E4BEE79439224B728BCF48C31F7FA6CE.pdf

Diaz, E., Kumar, S., & Wali, A. (2017). *Clojure: High performance JVM programming*. Packt Publishing.

Dibert, J. C., & Velez, J. C. (2006). *An analysis of earned value management implementation within the F-22 system program office's software development* [Master's thesis, Naval Postgraduate School]. NPS Archive: Calhoun. https://calhoun.nps.edu/handle/10945/34233

Doerr, J. (2018). *Measure what matters: How Google, Bono, and the Gates Foundation rock the world with OKRs*. Penguin.

Efe, P., & Demirörs, O. (2013, September). Applying EVM in a software company: Benefits and difficulties. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 333–340). IEEE.

Fleming, Q. W., & Koppelman, J. M. (1997). Earned value project management. *Cost Engineering*, *39*(2), 13.

Flow Framework. (2022). *What is the flow framework?* https://flowframework.org/about/

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate*. IT Revolution.

Forsgren, N., & Kersten, M. (2018). DevOps metrics. *Communications of the ACM*, *61*(4), 44–48.

Fox, M. R. (2020). IT governance in a DevOps World. *IT Professional*, *22*(5), 54–61.

Gansler, J. S., & Lucyshyn, W. (2013). *Using leading indicators to improve DoD acquisitions* (UMD-AM-13-102). Center for Public Policy and Private Enterprise, University of Maryland School of Public Policy.

Garrison, G. (2022). *Let's talk agile: Operational test for agile software programs—What you need to know about what you need to test*. Defense Acquisition University. https://www.dau.edu/event/Lets-Talk-Agile-Webinar-Operational-Test-for-Agile-Software-Programs-What-You-Need-to-Know-About-What-You-Need-to-Test

Gavrilovic, V. (2013). *Outcomes and outputs*. Renaissance Planning. https://www.citiesthatwork.com/blog/2013/10/outcomes-and-outputs

Goldratt, E. M. (1990). *The haystack syndrome: Sifting information out of the data ocean*. North River Press.

Goldratt, E. M. (1997). *Critical chain.* North River Press.

Hansen, M., & Nesbit, R. F. (2000). *Report of the Defense Science Board Task Force on defense software*. Defense Science Board.

Harrison, D., & Lively, K. (2019). *Achieving DevOps*. Springer Books.

Hartmann, D., & Dymond, R. (2006, July). Appropriate agile measurement: Using metrics and diagnostics to deliver business value. In *AGILE 2006 (AGILE'06)* (pp. 6–134). https://doi.org/10.1109/AGILE.2006.17

Hayes, W., Miller, S., Lapham, M. A., Wrubel, E., & Chick, T. (2014). *Agile metrics: Progress monitoring of agile contractors* (CMU/SEI-2013-TN-029). Carnegie Mellon University Software Engineering Institute. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=77747

Hayes, W., Place, P., Cohen, J., Brown, N., Korzec, K., & Miller, C. (2020). *F-35 JPO software metrics white paper*. Carnegie Mellon University. https://apps.dtic.mil/sti/citations/AD1121518

Hering, M. (2018). *DevOps for the modern enterprise: Winning practices to transform legacy IT organizations*. IT Revolution.

Hodson, E. G. (2016). *Earned value reporting on agile software development programs within the Department of Defense* [Master's thesis, Air Force Institute of Technology]. AFIT Archive: Scholar. https://scholar.afit.edu/etd/397/

Hughes, G. D., & Chafin, D. C. (1998). Turning new product development into a continuous learning process. *IEEE Engineering Management Review*, *26*, 32–45.

Imai, K., Nonaka, I., & Takeuchi, H. (1984). Managing the new product development process: How Japanese companies learn and unlearn. *Harvard Business School*.

Inthapichai, L. (2020). *Little's law applied in agile & knowledge work—Part 1 of 2*. https://medium.com/swlh/littles-law-applied-in-agile-knowledge-work-part-1-81c0c1f217ec

Kahn, K. B. (1996). Interdepartmental integration: A definition with implications for product development performance. *Journal of Product Innovation Management*, *13*(2), 137–151.

Kahneman, D., Sibony, O., & Sunstein, C. R. (2021). *Noise: A flaw in human judgment*. Little, Brown.

Karlström, D., & Runeson, P. (2005). Combining agile methods with stage-gate project management. *IEEE Software*, *22*(3), 43–49.

Karlström, D., & Runeson, P. (2006). Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering*, *11*(2), 203–225.

Kenney, C. (2021). Agile methods for project controls. In *IIE Annual Conference. Proceedings* (pp. 638–643). Institute of Industrial and Systems Engineers. https://www.proquest.com/scholarly-journals/agile-methods-project-controls/docview/2560891376/se-2

Kersten, M. (2018). *Project to product: How to survive and thrive in the age of digital disruption with the flow framework*. IT Revolution.

Kim, G. (2019). *The unicorn project: A novel about developers, digital disruption, and thriving in the age of data*. IT Revolution.

Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software, 29*(6), 18–21.

Lapham, M. A., Miller, S., Adams, L., Brown, N., Hackemack, B., Hammons, C., Levine, L., & Schenker, A. (2011). *Agile methods: Selected DoD management and acquisition concerns* (CMU/SEI-2011-TN-002). https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9769

Liu, H. H. (2006). Applying queuing theory to optimizing the performance of enterprise software applications. In *Int. CMG Conference* (pp. 457–468). https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.3040&rep=rep1&type=pdf

Maddox, M., & Walker, S. (2021, November). Agile software quality metrics. In *2021 IEEE MetroCon* (pp. 1–3). IEEE. https://doi.org/10.1109/MetroCon54219.2021.9666049

Magennis, T. (2017). *TL;DR. Chapter summaries.* Medium. https://medium.com/forecasting-using-data/tl-dr-chapter-summaries-65c7c0ab8962

Mahoney, M. S. (1990). The roots of software engineering. *CWI Quarterly*, *3*(4), 325–334. https://www.princeton.edu/~hos/Mahoney/articles/sweroots/sweroots.htm

Mihalache, A. (2017). Project management tools for agile teams. *Informatica Economica*, *21*(4), 85–93.

Miller, S. (2020). *Virtual learning package 9: Agile and measurement*. Carnegie Mellon University. https://apps.dtic.mil/sti/citations/AD1110339

Moore, G. A. (2014). *Crossing the chasm: Marketing and selling disruptive products to mainstream customers*. Harper Business.

National Defense Industrial Association. (2018). *Earned value management systems EIA-748-D intent guide*. https://www.ndia.org/-/media/sites/ndia/divisions/ipmd/division-guides-and-resources/ndia_ipmd_intent_guide_ver_d_aug282018.ashx

National Defense Industrial Association. (2019). *An industry practice guide for agile on earned value management programs* (Version 1.3) [Handbook]. https://www.ndia.org/-/media/sites/ndia/divisions/ipmd/division-guides-and-resources/ndia_ipmd_agileandevmguide_version_1-3_may302019.ashx

Naur, P., & Randell, B. (1969). *Software engineering: Report of a conference sponsored by the NATO Science Committee* [Paper presentation]. Software Engineering, Garmisch, Germany. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

Nichols, W. R., Yasar, H., Antunes, L., Miller, C. L., & McCarthy, R. (2022). Automated data for DevSecOps programs. In *Proceedings of the 19th Annual Acquisition Research Symposium*, *2*, 163–179. https://dair.nps.edu/handle/123456789/4542

Nicolette, D. (2015). *Software development metrics*. Simon and Schuster.

Norton, D. (2020). *Escape velocity*. OnBelay Consulting.

Office of Management and Budget. (2022). *Circular no. A–11: Preparation, submission, and execution of the budget.* Executive Office of the President. https://www.whitehouse.gov/wp-content/uploads/2018/06/a11.pdf

Office of the Under Secretary for Defense for Acquisition and Sustainment. (n.d.-a). *Capabilities needs statement (CNS)*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/cns/

Office of the Under Secretary for Defense for Acquisition and Sustainment. (n.d.-b). *Define capability needs*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/define-capability-needs/

Office of the Under Secretary for Defense for Acquisition and Sustainment. (n.d.-c). *Execution phase*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/execution-phase/

Office of the Under Secretary for Defense for Acquisition and Sustainment. (n.d.-d). *FAQs*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/faqs/

Office of the Under Secretary of Defense for Acquisition and Sustainment. (n.d.-e). *MVP, MVCR, and deployment frequency*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/mvp-mvcr/

Office of the Under Secretary of Defense for Acquisition and Sustainment. (n.d.-f). *Software acquisition*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/

Office of the Under Secretary of Defense for Acquisition and Sustainment. (n.d.-g). *SWP semi-annual reporting sheet*. Retrieved November 7, 2022, from https://www.milsuite.mil/book/docs/DOC-954816

Office of the Under Secretary of Defense for Acquisition and Sustainment. (n.d.-h). *User agreement (UA)*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/user-agreement/

Office of the Under Secretary of Defense for Acquisition and Sustainment (n.d.-i). *Value assessment2*. Retrieved November 7, 2022, from https://aaf.dau.edu/aaf/software/value-assessment2/

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2019, November 18). *Contracting considerations for agile solutions: Key agile concepts and sample work statement language* [Handbook]. https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/Contracting%20Considerations%20for%20Agile%20Solutions%20v1.0.pdf

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020a, November 17). *Agile and earned value management: A program manager's desk guide* [Handbook]. https://www.acq.osd.mil/asda/ae/ada/ipm/docs/AAP%20Agile%20and%20EVM%20PM%20Desk%20Guide%20Update%20Approved%20for%20Nov%202020_FINAL.pdf

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020b, November 11). *Agile metrics guide: Strategy considerations and sample metrics for agile development solutions* [Handbook]. https://aaf.dau.edu/wp-content/uploads/2022/08/Agile-Metrics-Guide.pdf

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020c, October 2). *Operation of the software acquisition pathway* (DoDI 5000.87). Department of Defense.

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2022a, June 8). *Operation of the adaptive acquisition framework* (DoDI 5000.02). Department of Defense.

Office of the Under Secretary for Defense for Acquisition and Sustainment. (2022b, July 27). *DoD's software acquisition pathway: Digital delivery at the speed of relevance [Brief]*. https://www.dau.edu/Lists/Events/Attachments/617/SWP%20Take%20Three%20-%20Lets%20Talk%20Agile%20-%2027%20Jul%202022b.pdf

Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics. (2015). *Performance of the defense acquisition system*: *2015 annual report*.

Office of the Under Secretary of Defense for Research and Engineering. (2018). *Design and acquisition of software for defense systems*. Defense Science Board. https://apps.dtic.mil/sti/citations/AD1048883

O'Hearn, B. (2022). *Lean, agile, and DevSecOps for AFLCMC/WIU engineering*. Carnegie Mellon University.

Oza, N., & Korkala, M. (2012). Lessons learned in implementing agile software development metrics. In *UK Academy for Information Systems Conference Proceedings* (p. 38). https://aisel.aisnet.org/ukais2012/38/

Özkan, D., & Mishra, A. (2019). Agile project management tools: A brief comparative view. *Cybernetics and Information Technologies*, *19*(4), 17–25.

Packaged Agile. (2020, December 13). *GAO and DoD say earned value management (EVM) makes sense on agile programs*. *But is it a good fit?* https://packagedagile.com/gao-and-dod-say-earned-value-management-evm-makes-sense-on-agile-programs-but-is-it-a-good-fit/

Park, J. M. (2010). *Agile EVM*. Northop Grumman.

Patel, S. (2021). *Dynamic modeling of the effectiveness of software development methods on DoD programs* [Doctoral dissertation, The George Washington University]. GWU Archive: ProQuest. https://www.proquest.com/dissertations-theses/dynamic-modeling-effectiveness-software/docview/2572595494/se-2

Patton, J., & Economy, P. (2014). *User story mapping: Discover the whole story, build the right product*. O'Reilly Media.

Pelrine, J. (2011). On understanding software agility: A social complexity point of view. *Emergence: Complexity & Organization, 13*.

Perri, M. (2018). *Escaping the build trap: How effective product management creates real value*. O'Reilly Media.

Poppendieck, M. (2011). Principles of lean thinking. *IT Management Select*, *18*(2011), 1–7.

Rawsthorne, D. (2006). *Calculating earned business value for an agile project*. CollabNet. https://www.agileleanhouse.com/lib/lib/Organizations/_CollabNet/CalculatingEarnedBusinessValueforAgileProject.pdf

Rawsthorne, D. (2008). *Monitoring scrum projects with AgileEVM and earned business value (EBV) metrics*. CollabNet. https://1library.net/document/y6o7k3gy-monitoring-scrum-projects-agileevm-earned-business-value-metrics.html

Reinertsen, D. G. (1997). *Managing the design factory*. Simon and Schuster.

Reinertsen, D. G. (2009). *The principles of product development flow: Second generation lean product development*. Celeritas.

Reinertsen, D. G. (2011). Towards developing accelerators in half the time. In *Proceedings of the Second International Particle Accelerator Conference* (pp. 1978–1980).

Ries, E. (2011). *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Currency.

Ries, E. (2017). *The startup way: How modern companies use entrepreneurial management to transform culture and drive long-term growth*. Currency.

Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). The secret history of agile innovation. *Harvard Business Review*.

Rossberg, J. (2019). *Agile project management with azure DevOps: Concepts, templates, and metrics*. Apress.

Royce, W. (1970). Managing the development of large software systems. In *Technical Papers of Western Electronic Show and Convention (WesCon)* (pp. 328–338). http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

Schwartz, M. (2020). *The delicate art of bureaucracy: Digital transformation with the monkey, the razor, and the sumo wrestler*. IT Revolution.

Section 809 Panel. (2018). *Report of the Advisory Panel on Streamlining and Codifying Acquisition Regulations: Volume 1 of 3*. https://discover.dtic.mil/section-809-panel/

Singer, P. W., Friedman, A. (2014). *Cybersecurity and cyberwar: What everyone needs to know*. Oxford University Press.

Smith, P. G., Reinertsen, D. G. (1997). *Developing products in half the time: New tools, new rules*. Wiley.

Stretton, A. (2007). A short history of modern project management. *PM World Today*, *9*(10), 1–18.

Sulaiman, T. (2007). *AgileEVM: Measuring cost efficiency across the product life cycle*. InfoQ. https://www.infoq.com/articles/agile-evm/

Sulaiman, T., Barton, B., & Blackburn, T. (2006). AgileEVM—Earned value management in scrum projects. In *AGILE '06: Proceedings of the Conference on AGILE 2006* (pp. 10–16).

Surbiryala, J., & Rong, C. (2019, August). Cloud computing: History and overview. In *2019 IEEE Cloud Summit* (pp. 1–7). IEEE.

Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review*, *64*(1), 137–146.

Tate, D., & Bailey, J. (2022). When is it feasible (or desirable) to use the software acquisition pathway? In *Proceedings of the 19th Annual Acquisition Research Symposium*, *1*, 367–381. https://dair.nps.edu/handle/123456789/4541

Thayer, R. H. (2003). Software engineering glossary. *IEEE Software*, *20*(4), c3.

Thomke, S., & Reinertsen, D. (2012). Six myths of product development. *Harvard Business Review*, *90*(5), 84–94.

Wind, J., & Mahajan, V. (1997). Issues and opportunities in new product development: An introduction to the special issue. *Journal of Marketing Research*, *34*(1), 1–12.

Winterowd, R. (2013). *Agile and EVM for the DoD: A review of the challenges and a new approach to solve them* [Master's thesis, Regis University]. Regis Archive: ePublications. https://epublications.regis.edu/theses/233/

Wodtke, C. R. (2016). *Radical focus: Achieving your most important goals with objectives and key results*. Boxes and Arrows.

Workman, J. P., Jr. (1993). Marketing's limited role in new product development in one computer systems firm. *Journal of Marketing Research*, *30*(4), 405–421.

Wrubel, E., Miller, S., Lapham, M. A., Chick, T. A., Brey, D., Nidiffer, K., Boardman, R., Carlson, R., Crowe, P., Walker, J. C., Matuzic, P., & Molin, C. (2014). *Agile software teams: How they engage with systems engineering on DoD acquisition programs* (CMU/SEI-2014-TN-013). Carnegie Mellon University Software Engineering Institute. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=295943

Yeo, K. T. (1991). Forging new project value chain—Paradigm shift. *Journal of Management in Engineering, 7*(2), 203–212.