NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CASE STUDY OF SOFTWARE DEVELOPMENT IN THE DOD

by

Amy Hsu and Robert Patterson

September 2020

Thesis Advisor: Glenn R. Cook
Second Reader: Thomas J. Housel

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE CASE STUDY OF SOFTWARE DEVELOPMENT IN THE DOD	**5. FUNDING NUMBERS**
6. AUTHOR(S) Amy Hsu and Robert Patterson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The Department of Defense (DOD) and its components have been pushing to consolidate their information technology infrastructure in order to reduce cost and waste of unused resources and increase the efficiency, effectiveness, and security of the infrastructure. Technology Services Organization (TSO), a Marine Corps unit, recently migrated its software development environment from the Marine Corps Worldwide (MCW) network to the Marine Corps Enterprise Network (MCEN), where software development is restrictive. Now TSO is setting its sights on optimizing its development process and eventually transitioning to DevSecOps and the cloud. This case study explored the software development methodology and environments of similar organizations within the DOD and examined how TSO might improve software development performance metrics and proceed to a DevSecOps environment. Two of the three organizations interviewed employ agile and pseudo-agile methodologies, and the third is in the process of transitioning to DevSecOps. Organizations familiar with agile methodologies are best suited for the transition but will still face challenges. Management and DevSecOps teams can overcome these challenges by focusing on their people, processes, and tools.

14. SUBJECT TERMS software development, code migration, cloud migration, DevSecOps, case study, Technology Services Organization, TSO, Marine Corps Worldwide, MCW, Department of Defense, DOD	**15. NUMBER OF PAGES** 83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

CASE STUDY OF SOFTWARE DEVELOPMENT IN THE DOD

Amy Hsu
Captain, United States Marine Corps
BS, University of California - Davis, 2009

Robert Patterson
Captain, United States Marine Corps
BS, U.S. Naval Academy, 2014

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

and

MASTER OF BUSINESS ADMINISTRATION

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: Glenn R. Cook
Advisor

Thomas J. Housel
Second Reader

Thomas J. Housel
Chair, Department of Information Sciences

Glenn R. Cook
Academic Associate, Graduate School of Defense Management

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Department of Defense (DOD) and its components have been pushing to consolidate their information technology infrastructure in order to reduce cost and waste of unused resources and increase the efficiency, effectiveness, and security of the infrastructure. Technology Services Organization (TSO), a Marine Corps unit, recently migrated its software development environment from the Marine Corps Worldwide (MCW) network to the Marine Corps Enterprise Network (MCEN), where software development is restrictive. Now TSO is setting its sights on optimizing its development process and eventually transitioning to DevSecOps and the cloud. This case study explored the software development methodology and environments of similar organizations within the DOD and examined how TSO might improve software development performance metrics and proceed to a DevSecOps environment. Two of the three organizations interviewed employ agile and pseudo-agile methodologies, and the third is in the process of transitioning to DevSecOps. Organizations familiar with agile methodologies are best suited for the transition but will still face challenges. Management and DevSecOps teams can overcome these challenges by focusing on their people, processes, and tools.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ADTE	Application Development and Test Environment
AI	artificial intelligence
ATO	authority to operate
CaaS	container as a service
CI	continuous integration
CD	continuous delivery
CFD	cumulative flow diagram
DaaS	data as a service
DIB	Defense Innovation Board
DOD	Department of Defense
DMDC	Defense Manpower Data Center
DSO	development, security, operations
ERP	enterprise resource planning
ESD	Enterprise Services Division
FY	fiscal year
HR	human resources
IaaS	infrastructure as a service
IDE	integrated development environment
IS	information systems
ISSE	Information Systems Security Engineer
ISSO	Information Systems Security Officer
ISSM	Information Systems Security Manager
IT	information technology
JEDI	Joint Enterprise Defense Infrastructure
NDAA	National Defense Authorization Act
NAVSUP BSC	Navy Supply Systems Command Business Systems Center
MCFIAS	Marine Corps Financial Integrated Analysis System
MCIPPS	Marine Corps Integrated Pay and Personnel Systems
MCTFS	Marine Corps Total Force Systems
MARFORCYBER	Marine Forces Cyber

MCBOSS	Marine Corps Business Operations Support System
MCEN	Marine Corps Enterprise Network
MCW	Marine Corps Worldwide
MTTR	mean-time-to-recover
MVP	minimum viable product
PaaS	platform as a service
SaaS	software as a service
SABRS	Standard Accounting, Budgeting and Reporting System
SAT	system acceptance testing
SIT	system integration testing
SDLC	software development life cycle
SSMS	SQL Server Management Studio
TSO	Technology Services Organization
UAT	user acceptance testing
USMC	United States Marine Corps
VDI	virtual desktop infrastructure
XP	Extreme Programming

ACKNOWLEDGMENTS

We express our gratitude to the staff of the Department of Information Sciences and the Graduate School of Defense Management. Their expertise and student-first attitude made this a rewarding experience. Thank you to our advisors for the support, patience, and commitment during the unorthodox events of completing this thesis. We would like to thank the interviewees from TSO, NAVSUP BSC, and DMDC for taking the time to help us learn about the software development process in their organization. We would also like to thank our Marine Corps friends, Captains Chew and Hutcheon, for helping us on the topic of DevSecOps.

Rob would like to thank his wife for her support and behind-the-scenes hard work that allowed him to successfully complete the demands of the last two years.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Technology Services Organization (TSO) is a Marine Corps unit equipped with an in-house software development team. According to the orientation given to the authors, TSO is tasked with the mission to provide "development, production and sustainment support of enterprise-level military pay, accounting, personnel and financial management information technology (IT) systems for the Marine Corps, Department of Navy and other Department of Defense components, services and agencies. Utilizing industry standards and innovative technologies, TSO delivers secure, auditable, and proficient capabilities to its stakeholders" (TSO, PowerPoint slides from meeting, February 11, 2020). For TSO to provide quality development, production, and sustainment support, it requires a software development environment with access to an array of tools and resources. Access to these tools and resources can become problematic with ever-increasing cybersecurity threats and corresponding cybersecurity policies and network requirements.

Prior to September 2019, TSO operated within the Marine Corps Worldwide (MCW) network. The MCW network allowed the developers at TSO the leeway to use a wide array of tools and resources, but as a result of the continued efforts to consolidate IT infrastructures as directed by the Secretary of Defense in August of 2010 and in accordance with Marine Forces Cyber (MARFORCYBER) Operations Order 18–0001, TSO was directed to migrate to Marine Corps Enterprise Network (MCEN) NLT 30 September 2019.

The MCEN is the Marine Corps' all-encompassing network composed of "people, processes, logical and physical infrastructure, architecture, topology and Cyberspace Operations" (United States Marine Corps [USMC] 2011, under "What is it?"). Prior to TSO's migration to MCEN, MCEN's policy did not allow for software development and testing, thus presented a major obstacle for TSO's day to day operation. TSO overcame this obstacle by creating an Application Development and Test Environment (ADTE) on a virtual desktop infrastructure (VDI), accessible from MCEN. As TSO worked through the obstacles of migrating to MCEN, it sets its sight forward to evolve with a landscape driven

by the requirements for faster and more secure software, as well as the Department of Defense's (DOD) directive to move into the clouds. This case study intends to analyze the software development environment at TSO and determine the changes necessary for its eventual migration to DevSecOps and the cloud.

B. PROBLEM STATEMENT

The problem is that the DOD is behind the curve on software development. The method for developing and acquiring software is outdated in many DOD organizations. This is a problem because the older development methods are too slow at adapting warfighters' needs while maintaining a high level of security. Metrics from the private sector are showing high performing software developers have a low change lead rate and high deployment frequency. These developers achieve these metrics by using DevOps and DevSecOps. Having a low change lead rate and high deployment frequency are ever more important for the DOD to support its warfighters, and to defend itself from being out cycled by its competitors and adversaries. DOD organizations, such as TSO, cannot achieve these metrics with their current software development method.

C. PURPOSE STATEMENT

The purpose of this research is to conduct a comprehensive examination of options TSO can adopt to improve its software development performance metrics, such as reducing its change lead rate and increasing its deployment frequency, and eventually transition to a DevSecOps environment. It is important for DOD organizations to improve their software development performance metrics to meet the operational demands of today's warfighters. This research will analyze how similar software development units across the Department of Defense develop their software products and the feasibility of TSO to adopt different approaches in order to improve their own development process. Due to the reliance of software in the DOD, this will provide TSO with alternate development methods that improve deployment frequency and functionality while maintaining security. Additionally, this research will look ahead for an optimal way to migrate the existing software development environment to the cloud.

D. RESEARCH QUESTIONS

This thesis seeks to answer the following questions:

- How could TSO go from a multitude of approaches to DevSecOps?
- How will the transition to DevSecOps impact the software developers at TSO to continue to develop critical software securely and efficiently?
- How do similar software development activities within the Department of Defense develop software and might the Marine Corps consider similar approaches?
- What is the optimal way to migrate the existing software development environment to the cloud, and continue to allow access to the requisite software development tools?

E. RESEARCH METHODS

This research is a qualitative case study relying on a review of select DOD organizations' software development methods through interviews and site visits. First, this research will begin with a review of the literature about software development and cloud computing, to include migration methods that might a provide a framework for the interview questions. Using information from the literature, we will develop and modify our interview questions to gather perspectives and best practices amongst the organizations interviewed. Interviews will be conducted with developers and managers that are involved with the day-to-day tasks of development.

The authors expected to conduct site visits and interact with the developers face to face in order to complete a robust analysis and provide worthwhile recommendations to TSO. However, only one site visit was conducted at TSO before COVID-19 unexpectedly halted all travel. The authors interviewed the remaining two organizations through multiple meetings conducted via phone calls and/or videoconference, in addition to email correspondence.

F. PROPOSED DATA, OBSERVATION, AND ANALYSIS METHODS

Data will be limited to available government publications on software development and cybersecurity prior to interviews and site visits with various DOD organizations that conduct software development. Interviews and site visits are intended to provide the authors with an accurate visualization of the development environment and processes employed by each of the visited organizations. Information on the development environment and processes including tools, resources, policies, and regulations will be collected and analyzed against existing literature and current industry practice.

G. POTENTIAL BENEFITS AND LIMITATIONS

This research is beneficial in providing an analysis on the differences in current software development environments employed by DOD organizations, with a focus on the processes and tools used at these organizations. Access to DOD organizations' internal cybersecurity policies, processes, and regulations may be a limitation to the research. The findings from the case studies will form recommendations aimed to improve TSO's software development performance metrics and the rate it can develop secure software.

H. ORGANIZATION OF THE THESIS

Chapter II is a literature review of topics relevant to this case study. The literature review covers the following topics: the state of DOD's software development capabilities, DOD modernization, software development methodologies, organic development, software development tools, software development performance metrics, cloud migration, and software development in the cloud environment.

Chapter III contains the research methods and a summary of organizations that were interviewed. The organizations interviewed were: Marine Corps Technology Services Organization (TSO), Navy Supply Systems Command Business Systems Center (NAVSUP BSC), and Defense Manpower Data Center (DMDC).

Chapter IV contains the summarized interview responses from the three organizations, an analysis of similarities and differences amongst the organizations, and a

discussion of how TSO can adopt DevSecOps and improve its development performance metrics in order to develop secure software, faster.

Chapter V provides a summary of the research, conclusions, and recommendations for TSO to go forward with their software development optimization and migration of their development environment to the cloud. The chapter concludes with recommendations for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LITERATURE REVIEW

A. INTRODUCTION

Software is everywhere in the Department of Defense (DOD). In today's environment, software is the mission-critical item in many of our systems, both weapons and business alike (Defense Innovation Board [DIB], 2019). It is no longer only an enabler of the Department's hardware components. Currently, America's adversaries are developing and deploying their software-enabled capabilities faster and more efficiently than us. The DOD must be able to adapt, respond, develop, and protect our most vital software-defined capabilities better than its adversaries or else secede its military advantages.

This chapter describes a few of the many software methodologies, tools and metrics, the advantages of organic development, and the benefits of cloud migration that could advance the DOD's ability to develop and deploy superior and more secure software.

B. STATE OF DOD'S SOFTWARE DEVELOPMENT CAPABILITIES

The DOD is heavily reliant on software and as the landscape of warfare changes, so must the DOD's ability to rapidly code, refine, deploy and monitor all software, but especially the most mission critical. Software is vitally important for the DOD's combat and mission systems, as well as an essential component to the department's business and enterprise activities that ensure it can effectively function (DIB, 2019). All the systems must work in harmony in order to ensure transactions are made that enable the acquisition of updated weapons and equipment, validate that personnel are paid and arrive to a theatre on time, as well as keep all of the records and personal data safe.

The software landscape has changed drastically, and the DOD must recognize the urgency to adapt to that change. The current procurement process treats software programs like hardware programs but that is no longer acceptable as it cannot produce the timely delivery of much-needed software capabilities (DIB, 2019). A study conducted by the Defense Innovation Board (2019), titled the *Software Is Never Done Refactoring the Acquisition Code for Competitive Advantage* decomposed the Department's software and

systems into three broad operational categories in order to highlight that not all systems are the same and they must be optimized accordingly. The board (2019) categorized them as Enterprise Systems, Business Systems and Combat Systems. Enterprise Systems are DOD-level systems that are very large-scale and must maintain large amounts of records and interface with multiple other systems (DIB, 2019). They include email systems, accounting systems and travel and human resource (HR) systems (DIB, 2019). Business systems are essentially the same but are differentiated by scale. Business systems operate at the service level (DIB, 2019). They need to be interoperable with other DOD systems, but each service can customize them to fit their needs. Examples include logistics systems, software development environments, or HR and financial systems (DIB, 2019). The third is the combat systems. While slightly different from the first two, based on the reliance of enterprise and business systems software, attacks on those systems can cripple the department or service component's combat systems as well (DIB, 2019).

Due to the close integration of all the systems and the reliance on software, Department and the Services must modernize its software development practices. In recent years, leadership has focused on these efforts and is transitioning to more adaptive and faster software development acquisition and development methods. As threats evolve, so must the Department's ability to respond to them. There is evidence that China expects to be the world leader in Artificial Intelligence (AI) by 2030 and is very concerned with cybersecurity and focused on becoming the world leader in both (Webster et al., 2017). These indicate the focus on software by the Nation's most capable adversaries and emphasizes the importance of improving the DOD's software capabilities as the foundation of other critical developments like machine learning and more intensive computing initiatives.

C. DOD MODERNIZATION

Ellen Lord, the Under Secretary of Defense for Acquisition and Sustainment said that software is the thread that runs through all our programs and that the DOD must shift its approach towards software development (Kelman, 2018). Agile and DevSecOps facilitate getting software to the fleet quickly and securely. Ellen Lord said,

> I believe we are at an inflection point in terms of doing things differently. We are pivoting from the traditional waterfall software development methodology to agile and DevOps. So, we are coding every day, testing every night. (Kelman, 2018)

The DOD has investigated the best software development practices for years with little improvement in the practices used by the DOD. However, in recent years, there has been a strong focus on modernization. The Fiscal Year (FY) 2018 National Defense Authorization Act (NDAA) directed the Secretary of Defense to task the Defense Innovation Board "to undertake a study on streamlining software development and acquisition regulations" (DIB, 2019). Most recently, the Department launched a joint program with the Under Secretary of Defense (Acquisition and Sustainment), the DOD CIO, the Air Force, DISA and the Services called the DOD Enterprise DevSecOps Initiative (Air Force, n.d.). As the lead on this initiative, the Air Force launched multiple software factories in Colorado, Boston (called Kessel Run), and throughout the United States in order to provide tools, platforms, and services to revolutionize software development within the DOD.

The DOD will benefit from the ability to develop software quickly and internally since software powers all our systems. It is not only a vital part of our business systems, but it is ingrained in our command and control, aircraft, and weapons. While developing software is not a core competency of the military, the functions that IT and software provide are critical to the DOD's core competencies. Software is a key enabler to maintain the United States military's competitive advantage over our adversaries, whether it is the software-intensive F-35 or less intensive manpower applications.

Software is the foundation for DOD's competitive advantage and that gap is shrinking as countries like Russia and China pour money into the quests for technological advantages. The United States military's technological advantages define its competitive advantage throughout the world. Figure 1 demonstrates the rise of software in DOD's aircraft. This same graph could apply to the business and enterprise systems. The internal development of software and utilizing partnerships with industry, like Kessel Run, will maintain this competitive advantage. The IS infrastructure flexibility being the ability to make existing, new and packaged software applications come together successfully (Byrd

et al., 2004). Software provides the instructions to applications that input, process, store and control the activities of the information systems (IS).

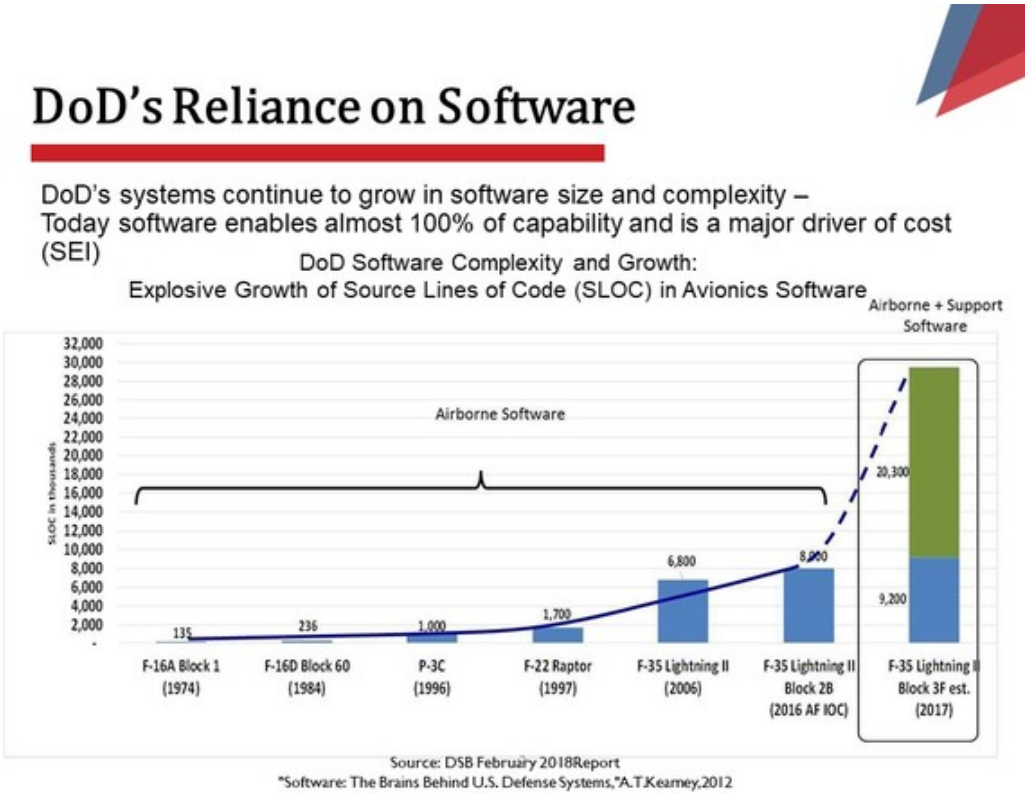

Figure 1. DOD Line of Code Progression. Source: DOD (2018c).

SOFTWARE DEVELOPMENT METHODOLOGIES

There is a multitude of different software development methodologies. The DOD recognizes that the waterfall method is not compatible with all software. It is a suitable method for acquiring large programs, like ships and vehicles, that have a defined criterion from the beginning that is unlikely to change. As far back as the 1980s, the DOD has initiated research projects into how to adjust software acquisitions, as well as implement better software development methods. Until recently, these studies lacked action. Industry adopted agile approaches many years ago and companies like Amazon, Facebook, and Netflix have reaped the benefits of fewer defects, faster delivery and lower costs (Rigby et al., 2018). A study conducted by the Standish Group of IT projects between 2013 and 2017

found that Agile projects were about twice as likely to succeed and one-third as likely to fail as Waterfall projects (Mersino, 2018).

1. Waterfall

The Waterfall methodology is the traditional software development methodology that follows a sequential process (Mahalakshmi & Sundararajan, 2013). It was adopted from the traditional hardware strategies adopted in the 1970s (Ragunath et al., 2010). Ironically, the waterfall process in software development was an idea from William Royce about how it is a flawed software development process (Ragunath et al., 2010). Generically, it is a sequential five-step process. The five steps are plan, build, test, review, and deploy.

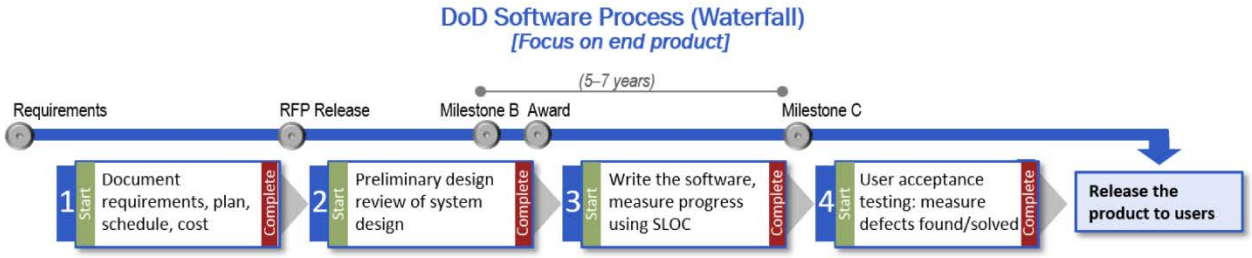

Figure 2. Waterfall Process. Source: Pace (2019).

The planning phase begins with requirements gathering. When the requirements are unlikely to change and clearly defined, the waterfall method is preferred. It is fairly resource light and easier to implement due to its sequential nature (Mahalakshmi & Sundararajan, 2013).

Due to the linear sequentially, the requirements phase must be robust in order to account for all the necessary features since changes are not accepted in the waterfall process (Mahalakshmi & Sundararajan, 2013). Some testing happens after each phase; however, the robust testing does not happen until the fourth phase since working software is not produced until late in the life cycle (Ragunath et al., 2010). This is problematic since costly and repairable errors that happen early are not caught until near the end (Mahalakshmi & Sundararajan, 2013). It also adds a high amount of risk and uncertainty (Ragunath et al., 2010). As a project progresses without testing and with possible bugs, it makes it more

difficult and costlier to find and fix since it was programmed potentially months before it was identified. Not to mention, if not all the defects are found, users are left with a faulty product and must endure long cycle time to receive an updated, coherent, and capable product. Sequential phases provided little room for customer feedback, which is the advertised improvement of agile methodologies.

2. Agile

In 2001, a group of 17 software developers collaborated in Utah to write the Agile Manifesto (Beck et al., 2001). It defined four values and 12 principles that encompassed a new way of developing software (Beck et al., 2001). It focused on the customer's needs as well as the ability to accept changes to requirements and delivering working software (Beck et al., 2001). It attempted to create a new development method that replaced the waterfall method's shortcomings. The values of the Agile Manifesto are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan (Beck et al., 2001)

The Agile Methodology relies on an iterative approach that delivers usable software to the customer to provide feedback in order to incrementally deploy the application (Beck et al., 2001). It deploys completed code in iterations then applies customer feedback for the next iteration. In Agile, communication with the customer is key as they provide often and frequent feedback to the developers (Beck et al., 2001). It is a more responsive development methodology that allows the organization to maintain relevance in a constantly changing IT environment. In conjunction with the four values, these initial twelve principles started the Agile movement (Beck et al., 2001):

1. [The] 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.

The Agile values and principles can be applied to many different organizations. They are appropriate guiding lights for general customer interactions and all business; however, their focus is on software. To that end, it is important to note that the Agile method is not the ideal method for software in all cases. Every software organization must evaluate themselves and understand their abilities in order to determine the right software development methodology for their service or application and their organization.

3. Extreme Programming

Extreme programming (XP) is one of the most popular agile methods. XP aims to produce higher quality software and a higher quality of life for the development team (Agile Alliance, n.d.). It focuses on the roles of customer, manager and programmer and outlines responsibilities for each of them (Anderson et al., 2001). The values of simplicity, communication, feedback, and courage are the foundation for the way the "XP Teams" conduct their work.

a. The XP Team

The XP team consists of the customer, the programmer, and the manager. The customer describes the user stories, the priority of work, and determines the criteria for acceptance. The preference is that the customer is on-site so that issues are resolved quickly and with less ambiguity. The customer can be one individual or a team of people. It is just

critical that they have an in-depth knowledge of the future system and its expected functionalities.

The programmers analyze, design, test, program, and integrate the system (Anderson et al., 2001). The XP programmers' job is to transform the user stories into one coherent, functional program that meets the customers' business needs and values (Anderson et al., 2001). A user story is essentially a requirement written in natural language of what the customer expects the system to do. The programmers write the code in pairs from the same machine in order to develop more code of better quality. This also allows them to switch off when one programmer gets tired and ensures that two people understand the code for a specific story.

The manager's function is to eliminate distractions for the programmers and very focused on the management of the project (Anderson, 2001). This person facilitates formal engagements between the customer and programmers in order to provide status updates, bring in new user stories, and manage relationships between the programmers working together, and the teams. This role is vitally important for the maintenance of the computers, the updates to the systems, and effectively designing the workspace. Ultimately, the manager coordinates activities, reports results, and always removes obstacles (Anderson, 2001).

b. How XP Works

XP relies heavily on customer involvement. The customer writes the user stories that define business value and then the programmer builds that. In a user story, the customer explains the features and functionality they want the system to accomplish. One of the key tenants of XP is communication so the process is a little more sophisticated. The customer defines value, then the programmer estimates the cost of the work which allows the customer to determine the appropriate action before the programmer begins coding that story (Anderson et al., 2001). The customers and programmers are highly dependent on each other.

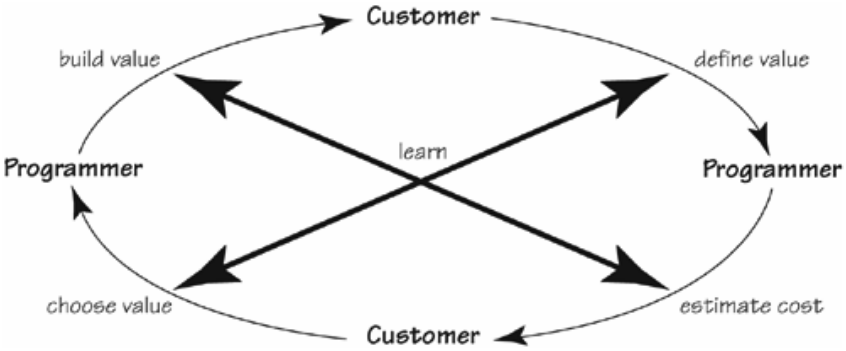

Figure 3. Improved Extreme Programming Cycle. Source: Anderson et al. (2000).

XP functions on two sprints to deliver a minimum viable product. After the above analysis of the entire project, the customer determines the most valuable features to be developed first. The programmers decompose the chosen stories into tasks and develop test cases in order to determine completion. The stories consist of two weeks, so the programmer has control over the scope and that timeline allows the programmer to make accurate estimates of completion (Anderson et al., 2000). This is in line with the principles of communication, simplicity, and feedback.

By incorporating testing into the process early it ensures the programmers remember the work they did and can quickly fix any problems. This provides immediate feedback and serves as a progress report. The important measure is not how many stories are complete but rather how many are functioning and passed testing. At the end of the two-week iterations, functioning code is released in order to give the customer a useful subset of the overall product (Anderson et al., 2000). This is assisted by continuous integration. As code passes testing, it is integrated early and often in order to avoid a large integration of all the programmer's code, but also to ensure everyone is working on the most recent code.

c. Principles

For practitioners looking to implement XP, the rules are simply stated in twelve rules (Beck, 1999).

1. The planning game: At the start of each iteration, customers, managers, and developers meet to flesh out, estimate, and prioritize requirements for the next release. The requirements are called “user stories” and are captured on “story cards” in a language understandable by all parties.
2. Small releases: An initial version of the system is put into production after the first few iterations. Subsequently, working versions are put into production anywhere from every few days to every few weeks.
3. Metaphor: Customers, managers, and developers construct a metaphor, or set of metaphors after which to model the system.
4. Simple design: Developers are urged to keep design as simple as possible, “say everything once and only once.”
5. Tests: Developers work test-first; that is, they write acceptance tests for their code before they write the code itself. Customers write functional tests for each iteration and at the end of each iteration, all tests should run.
6. Refactoring: As developers work, the design should be evolved to keep it as simple as possible.
7. Pair programming: Two developers sitting at the same machine write all code.
8. Continuous integration: Developers integrate new code into the system as often as possible. All functional tests must still pass after integration or the new code is discarded.
9. Collective ownership: The code is owned by all developers, and they may make changes anywhere in the code at any time they feel necessary.
10. On-site customer: A customer always works with the development team to answer questions, perform acceptance tests, and ensure that development is progressing as expected.
11. 40-hour weeks: Requirements should be selected for each iteration such that developers do not need to put in overtime.

12. Open workspace: Developers work in a common workspace set up with individual workstations around the periphery and common development machines in the center.

4. Scrum

Scrum is another agile framework within which people address complex adaptive problems, while productively and creatively delivering products of the highest possible value (Schwaber & Sutherland, 2017). First described in 1996, even before the Agile Manifesto, it adopts its name from Rugby. Scrum is based on empiricism, or that knowledge comes from experience and making decisions based on what is known (Schwaber & Sutherland, 2017).

Three pillars lay the foundation for the Scrum teams. The three pillars are transparency, inspection, and adaptation. Transparency is ensuring all observers share a common understanding throughout the project that includes using a common language and understanding the end state (Schwaber & Sutherland, 2017). Inspection is frequent and deliberate in order to detect undesirable variances (Schwaber & Sutherland, 2017). Adaption is quickly adjusting errors in order to decrease the chance of further deviating from the intended outcome. Scrum team members are encouraged to live by the five Scrum values of commitment, courage, focus, openness, and respect.

a. Scrum Artifacts

In order to make sense of the team composition and how the Scrum works, it is important to understand some of the “Scrum” specific terms.

- Product Backlog: An ordered list of everything that is known to be needed in the product. The requirements that are initially known and best understood become the earliest project developments. It evolves as the project evolves. The product backlog lists all the features, functions, requirements, enhancements and fixes that must be made for future releases. Multiple teams work on one product backlog.

- Sprint Backlog: Are subsets of the Product Backlog that are selected for a specific Sprint. It includes a plan to deliver the product increment and the work necessary to deliver a finished increment.

- Sprint: This is the heart of the Scrum. It is a one-month or less period that a releasable product increment is created. Sprints are continuous and are composed of an ecosystem of Sprint specific tasks, like sprint planning, Daily Scrums, Sprint reviews and the sprint retrospective. During the Sprint, no changes are made that might affect the sprint goal. Changes are generally saved for a later sprint.

- Daily Scrum: A 15-minute meeting at the beginning of every day that outlines the Development Team plans for the next 24-hours and highlights what was accomplished the day before (Schwaber & Sutherland, 2017).

b. The Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. These are cross-functional teams that are self-sufficient. The Product Owner is responsible for managing the Product Backlog and prioritizing the items within it (Schwaber & Southerland, 2017). The Development Team is composed of the professionals who deliver the work at the end of each Sprint. The Development Team self-organizes into teams to accomplish the coding, testing, and business analysis (Schwaber & Sutherland, 2017). A team size between four and eight members is preferred. The Scrum Masters are responsible for the success of all the other teams by coordinating actions and increasing the efficiency, effectiveness, and productivity of the other two teams.

c. How Scrum Works

During the Daily Scrum, the Development Team reviews progress and determines its goal for the day (Schwaber & Sutherland, 2017). All the requirements and features start in the Product Backlog until Sprint Planning. During Sprint Planning, which is a maximum of eight hours, the Scrum Team determines the functionality that will result from the upcoming Sprint and how they are going to accomplish the work. As specific features are

selected for the upcoming iteration, they are placed in the Sprint Backlog and assigned to a team. Then during the Sprint, the Development Team completes its incremental contribution to the final product.

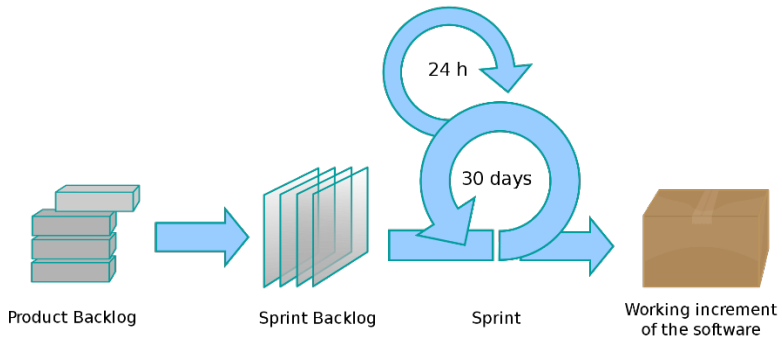

Figure 4. Scrum Process Graphic. Source: Scrum Process (2009).

At the end of a Sprint, a Sprint Review is held to inspect the increment and adjust the Product Backlog (Schwaber & Sutherland, 2017). This informal meeting is to discuss lessons learned, readjust the target delivery dates if necessary, and review the way forward. The outcome is an adjusted Product Backlog to prepare for the next Sprint Planning meeting/Sprint (Schwaber & Sutherland, 2017). The Sprint Review is boxed by a maximum duration of four hours. A Sprint Retrospective occurs in order to inspect intra-team dynamics, such as relationships, process, and tools in order to improve for next time. This is a formal event that lasts no longer than three hours (Schwaber & Sutherland, 2017).

Scrum is very timeline oriented. All the meetings have maximum duration times. Even once the Sprint begins, its duration is fixed and cannot be shortened or lengthened (Schwaber & Sutherland, 2017). Scrum strives to create regularity and to minimize the need for meetings (Schwaber & Sutherland, 2017). When the meetings have reached the end of their utility and the intent of the meeting is achieved it should end so that there is no waste in the process (Schwaber & Sutherland, 2017).

5. DevSecOps

In recent years, DevSecOps is one of the most highly adopted software development methodologies, a truncation of development, security, and operations. The term first appeared in a blog by a Gartner vice president, Neil MacDonald. He stated that the root cause of most of the downtime in the system is due to breakdowns in communication and process errors between the development, operations, and security teams (MacDonald, 2012). The overarching idea behind this method is to balance the need for speed and agility with the need to protect critical assets, applications, and services (MacDonald, 2012). Security tends to pull in the opposite direction of speed and agility so DevSecOps seeks to build it into every stage and incorporate it into the framework itself (Winder, 2018). There is not one single definition of DevOps; however, the authors of *What is DevOps?* from Amazon Web Services (n.d.) defined it as

> DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.

DevSecOps is one of the more challenging methodologies to implement due to the significant cultural shift it requires. DevSecOps is not simply adopting a different method, as in adopting scrum from waterfall. It involves members from the development security, and operations teams to create one team with a focus on security early and often. This new dynamic adds complexity to the transition due to the new interpersonal interactions between the members. Generally, friction occurs with DevSecOps because the security team often says "No," due to security concerns, the operations team strives for stability and are reluctant to make changes, and the development team desires to code fast and deploy frequently to satisfy customer needs (Carter, 2017). The culture must transition to one that is characterized by a high degree of collaboration across roles and that is focused on business objectives instead of departmental objectives (Coyne & Sharma, 2015). A primary focus must be on the social engineering aspect and creating teams that will work well together (Coyne & Sharma, 2015).

DevSecOps is incorporating security into the software development life cycle from the requirements gathering phase. This type of DevOps requires secure coding practice and security testing from the beginning of the life cycle in order to build it into the application. A prevalent bad practice is "testing in security" by adjusting the code if it fails a specific test (H. Pace, PowerPoint slides, 2019). DevSecOps makes security the responsibility of the development team and the operations team, not just the security team.

DevSecOps attempts to cover for the shortfalls of the traditional and well-established methodologies. According to Francis Raynaud, a leader of DevSecOps and the founder of the DevSecCon, the traditional methods treat security like "a bolt on" at the end when the product is close to delivery (Carter, 2017). DevSecOps is about teaching coders to code securely instead of merging security into already developed code (Carter, 2017). As new systems are developed, security is often misunderstood or ignored for more high-profile requirements like system availability or the correctness of software systems (Cois, 2014). In a DevSecOps software development life cycle, security is a fundamental requirement from the beginning (Cois, 2014).

a. How DevSecOps Works

Organizations that implement DevSecOps are relying on individuals to create cross-functional teams that work closely together to develop their product and removing silos. DevSecOps creates a shift towards "collaboration between development, quality assurance, and operations" (Ebert et al., 2016). Amazon Web Services (n.d.) described how DevSecOps work as:

> Under a DevOps model, development and operations teams are no longer "siloed." Sometimes, these two teams are merged into a single team where the engineers work across the entire application life cycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

The DevSecOps software life cycle is comprised of nine steps as opposed to the traditional five steps of requirements gathering, design, coding, testing, and implementation. The DevSecOps approach is phased between plan, develop, build, test, release, deliver, deploy, operate, and monitor (DOD, 2019). This is through a fully

automated process or semi-automated process in order to allow continuous integration and continuous delivery (CI/CD) (DOD, 2019). Continuous integration is the practice of merging the developers' code into the main branch or a central repository after which build and tests are run, whereas continuous delivery is ensuring the code is ready for release at any time (Pittet, n.d.). These are not linear steps but rather in parallel or simultaneously accomplished through the build process. DevSecOps strives to test and integrate the software early and often. With more frequent releases, DevSecOps is adaptable and can fluidly adjust based on customer feedback between iterations as necessary.

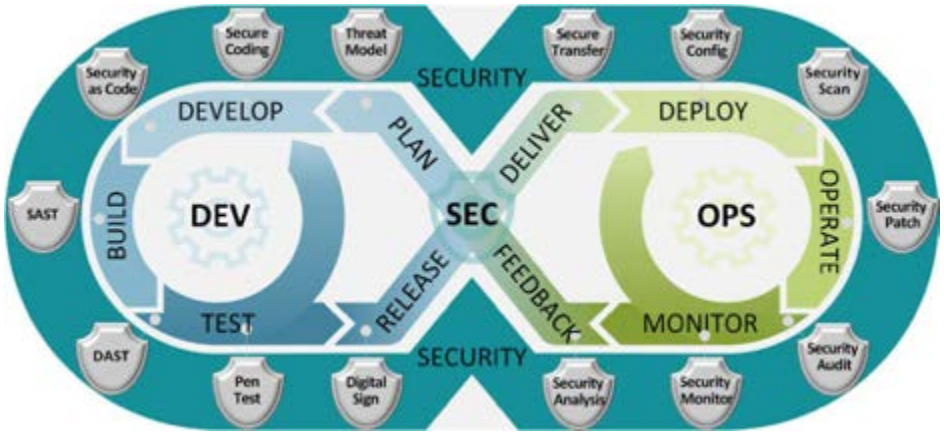

Figure 5. DevSecOps Process. Source: DOD (2019).

DevSecOps relies on heavily automated techniques to implement, maintain, and monitor the status of the project. Automation during the build and release stages is important as it minimizes human interaction with the software that might cause avoidable errors (Cois, 2014). In contrast to the traditional, siloed environments of the past, DevSecOps leverages CI/CD. Continuous integration (CI) is the process of building and testing software continuously and each time a change is made (Cois, 2014). Continuous deployment (CD) is an uninterrupted process that deploys live software to the production environment (Cois, 2014). Instead of developers releasing software on a schedule or during assigned product release dates, the software is now deployed continuously as part of the process.

One of the goals of DevSecOps is to react quickly and make changes more rapidly, which amplifies the customer feedback loop (Coyne & Sharma, 2015). When all the systems involved in DevSecOps work in concert with each other it provides the stakeholders and customers continuous, real-time information on the status of the project (Cois, 2014). To support this communication, DevSecOps releases a minimum viable product (MVP) which is the first point at which the code can do useful work and when feedback is gathered in order to support the refinement of certain features (DIB, 2019). An example of an MVP might be 25% of the fully completed application that contains 80% of the features. This provides the customer with a functioning application to request updates or approve the current design. Coyne and Sharma (2015) identified the feedback loop as one of the principles of DevSecOps and in order to successfully react, organizations must have a responsive feedback mechanism and then learn rapidly in order to improve the next iteration. This idea is consistent with the Agile Manifesto principle of more information, more often will lead to better project outcomes (Cois, 2014).

The release of the MVP is the first time the new feature will interact with the current environment. In the traditional method by waiting until the end to implement the complete package of software, there are fewer options to reiterate and deploy an improved package. This problem is enhanced when security is bolted on at the end instead of thoughtfully incorporated throughout the life cycle. DevSecOps combats this through the "shift-left" principle.

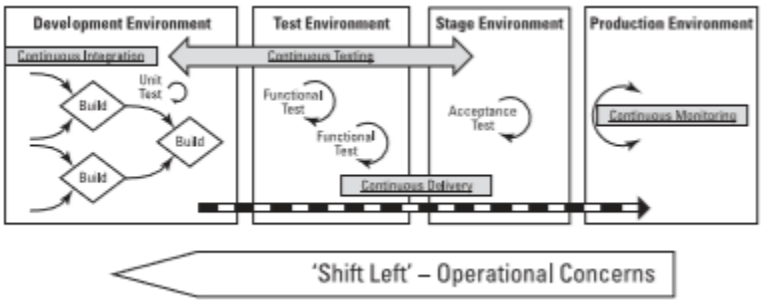

Figure 6. Shift-left Mentality. Source: DevOps for Dummies (2015).

One DevSecOps principle is to develop and test against production-like systems. This is the "shift-left mentality" which moves operations earlier in the development life cycle (Coyne & Sharma, 2015). For development and quality assurance, the goal of "shift-left" is to test early and often through automation in order to identify issues before deployment. By shifting left, the application is tested in a similar environment and the delivery process is validated upfront (Coyne & Sharma, 2015). For the operations staff, the shift-left principle allows them to observe how the environment supports the application and fine tune the environment to better support it. Continuous deployment supports the shift-left principle by eliminating surprises for the operations team by providing predictable, low-risk releases of validated highly-quality software (Cois, 2014). Containers and microservices facilitate the "shift-left" mantra by providing more reliable code through smaller scale iterations and a consistent software environment throughout the SDLC.

b. DevSecOps Tools

DevSecOps provides solutions for many of the shortcomings of traditional software development approaches but it requires specific procedures in order to be successful. Due to the error-prone nature of manual processes in addition to the waste and delayed responses, an organization must commit to the automation required in DevSecOps (IBM, 2013). Automation enables the CI/CD pipeline that provides DevSecOps it's speed and agility.

One such tool is the use of containers. Containers solve the issue of getting the software to run reliably between computing environments, for example, from the staging environment into production (Rubens, 2017). A CIO article (2013) explains how containers solve this issue:

> Put simply, a container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. By containerizing the application platform and its dependencies, differences in OS distributions and underlying infrastructure are abstracted away.

Containers provide a consistent environment from testing to final production and deployment (Surianarayanan et al., 2020). Containers are similar to virtualization;

however, virtualization is “heavier” as it contains a guest operating system (OS). Contrasted with virtualization, containers are more lightweight and use fewer resources (Rubens, 2013). Another benefit of containerization is that it allows for greater modularity (Rubens, 2013). An entire application does not need to be run in one container. It can be split into modules that are easier to manage since each module is relatively simple and the changes can be made without rebuilding the entire application (Rubens, 2013). This is called the microservices approach.

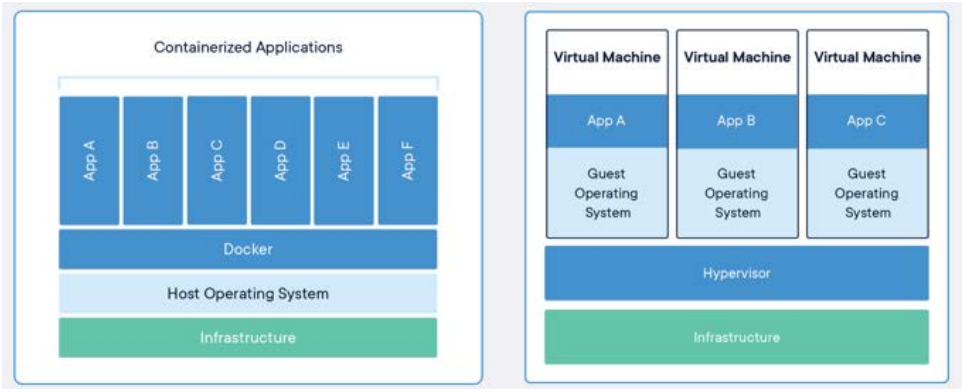

Figure 7. Containers versus virtualization. Source: Docker (2020).

In 2011, Martin Fowler documented the term “microservices (Surianarayanan et al., 2020). According to Fowler, microservices consist of “suites of independently deployable services” organized “around business capability and automated deployment” (Surianarayanan et al., 2020 p. 30). The main goal of the microservices architecture (MSA) is “to achieve easy maintenance, quick software development with frequent deployment, short development cycles, and continuous delivery” (Surianarayanan et al., 2020, pp. 36–37). Further, microservices are small applications that can be deployed, scaled, and tested independently and they are usually organized around a single business activity (Thones, 2015). Microservices are an enabler of DevSecOps.

Deploying software is a modular approach in contrast to a monolithic architecture where all the features are deployed as one component. The main idea is that an application that is partitioned into smaller microservices is easier to build and maintain

(Surianarayanan et al., 2020, p. 33). In the preceding years of the MSA, many applications were growing too large and were unable to be improved easily or at all. Additionally, the more modular approach is necessary for DevSecOps because it allows for smooth automated testing and continuous delivery (Thones, 2015). With the MSA, teams can make updates on the same application but on different functions without coordinating as closely due to the self-deployable nature of MSA. In the traditional monolithic method, although teams work on different functions, they are not all independent and they require close coordination (Surianarayanan et al., 2020, p. 37). A change in a monolithic application cannot go to deployment until all teams are ready due to the tight coupling among modules (Surianarayanan et al., 2020, p. 37). This inhibits automated testing and CI/CD. However, with a modular microservice approach, one team's work on a function does not affect the other functions within the same application. This allows that portion of the application to move throughout the SDLC when it is ready. Another benefit of the MSA is that the application code is more reliable since it is independent and not reliant on another team's input (Surianarayanan et al., 2020, p. 38). This provides less chance for human error or faulty compatibility during code mergers.

The previously discussed containers and container orchestration are one of the key enablers of the MSA. As mentioned, virtual machines are "heavy," so as opposed to using them, containers are a suitable choice since they are "light." (Surianarayanan et al., 2020, pp. 38–39). The container's small size, in addition to the guideline of *one microservice per container*, means that deployment time will take only a few seconds since they both consume fewer resources (Surianarayanan et al., 2020, p. 39-41). Their small size also makes them suitable to be deployed over cloud resources (Surianarayanan et al., 2020, p. 41). Microservices enable cloud adoption using containers since they can both scale up/down easily at the service level to make them more responsive (Surianarayanan et al., 2020, p. 56). Combined with event-driven computing, the idea that resources need to scale based upon an event, potentially an influx of customers on a site, the cloud and microservices combination can quickly provision and adjust the appropriate amount of resources (Surianarayanan et al., 2020, p. 56). The independent scaling of individual microservices also make it appropriate for developing web applications (Surianarayanan et

al., 2020, p. 56). This combination of microservices and containers provide an appropriate style for developing agile and secure applications with DevSecOps (Surianarayanan et al., 2020).

MSA is not a panacea for all software development organizations and must still be investigated in order to determine if it is the right approach. While it can enhance an organization's ability to transition to DSO, there are also drawbacks that limit its utility. By nature of employing microservices, it breaks down a larger application into smaller components, which adds complexity in the form of managing many different microservices (Surianarayanan et al., 2020, p. 45). The ease of building a less complex monolithic application into smaller pieces inherently adds more "pieces" one must maintain. Although they are easier to update and deploy, the developers must be cognizant of the services' ability to interact with all the other possible configurations and services (Surianarayanan et al., 2020, p. 45). These unforeseen and untested configurations add vulnerabilities. Although, the code is more reliable due to fewer developers working together on it, when the entire application comes together it may create opportunities to penetrate the system (Surianarayanan et al., 2020, p. 45).

There are trade-offs to employing a microservice architecture, especially for older organizations that must refactor their legacy applications. However, there is an increasing demand to become agile and to release continuously and deploy frequently. Due to some of the drawbacks mentioned previously, implementing MSA adds significant overhead and operational complexity at a service level, vice developer level. At the developer level, it is advantageous since there is increased freedom and ability to deploy independent of other teams (Surianarayanan et al., 2020, p. 37). Developers can also mix languages and frameworks, however, when an interconnection is necessary, the problem becomes complex and expensive to integrate one service with another (Surianarayanan et al., 2020, p. 58). It involves significant overhead and planning. MSA involves a high-speed network since the service will have strong boundaries between services. Monolithic applications are tightly coupled so the performance will be better in low bandwidth, high latency networks (Surianarayanan et al., 2020, p. 57). A specific concern to an organization that relies on databases, is that they must be decentralized databases in order to effectively employ

microservices due to the independence of the services and the strong boundaries between them (Surianarayanan et al., 2020, p. 57). It will take time to update all the databases unlike in monolithic applications where a change in a database is reflected across the application immediately (Surianarayanan et al., 2020, p. 57). Due to the nature of these problems, the expertise of management is important. The authors of the book, *Essentials of Microservice Architecture* recommend a DevOps culture due to the new skills and tools needed that involve reliance on automation and collaboration between the developers and security and operations personnel.

D. SOFTWARE DEVELOPMENT TOOLS

This section will provide a brief overview of the basic content creation tools used in software development.

- Integrated development environment (IDE) is the content creation tool developers spend most of their time with. It is a software application that consists of a suite of different functions such as source code editor, debugger, and compiler. Some IDEs are catered to specific programming languages. Examples of IDEs include Visual Studio, Eclipse, and IntelliJ IDEA.

- Version control tool is used to track and document changes to the source code. It is a must for developers working in teams to ensure developers are working on the same version of the code. Commonly used version control software includes Git, GitLab SCM, and Apache Subversion.

- Configuration management tool is like version control except it is used to track and document software builds along with changes to the development environment. It is used to ensure consistent deployment. Automated configuration management is an essential tool to form a CI/CD pipeline for DevOps and DevSecOps. Examples include Jenkins, CFEngine, and Ansible.

- Vulnerability scanners and code analyzers are tools used to test codes for potential and known issues. Some of these tools are used on static codes while others are used while the codes are being executed (dynamic). Examples of vulnerability scanners and code analyzers include Fortify, FindBugs, and SonarQube.

E. PERFORMANCE METRICS OF SOFTWARE DEVELOPMENT

There are different aspects of software measurement. The two main classifications of software measurement are product and process (Misra & Omorodion, 2011). The goal of these metrics is "identification and measurement of the essential parameters that affect software development" (Misra & Omorodion, 2011) so better decisions can be made. These measurements attempt to measure performance through different frameworks and indicators, quantitatively and qualitatively. Broadly speaking, projects are often measured by profitability through their Return on Investment (ROI). A profitability metric, such as ROI, is not always the best measurement of performance, especially in a non-profit organization such as the DOD. The cost and profitability of software are perhaps not the best indicators for software development in the DOD. A better way to measure the performance of software development would be through efficiency or productivity.

Different development methodologies use different metrics to measure productivity. There is no set list of standard metrics, therefore the metrics used in DOD's *Agile Metrics Guide* will be covered here. For Agile, these metrics are (DOD, 2019c):

- Story points measure the complexity of a story. This unit of measurement is the building block that allows a team the ability to estimate how much effort is required and how much work can be completed within a given sprint or release. This unit of measurement varies from team to team.

- Velocity measures the amount of work a team completes in each sprint or release. This measurement can be represented by units such as Story Points, hours, etc.

- Velocity variance is the standard deviation from average velocity. It shows the difference from the average.
- Velocity predictability is the difference between planned and completed velocity, or the difference between planned and completed story points.
- Story completion rate is the number of stories completed in each sprint or release.
- Spring burndown chart is used to estimate the work completed daily, usually in hours. The chart can be used to generate an estimated completion date based on the current work pace.
- Release burnup is a chart that measures the amount of work completed based on the total amount of work planned for a given release. This chart is used to estimate whether the team is on track.
- Cumulative flow diagram (CFD) depicts the flow of work through a process by keeping a cumulative count of the number of items at each step in the process.

For DevSecOps, metrics are used to provide insight into the delivery pipeline. These metrics include (DOD, 2019c):

- Mean time to recovery (MTTR) measures how quickly a system or solution can be restored to functional use after a critical failure.
- Deployment frequency provides information on the cadence of deployments in terms of time elapsed between deployments.
- Lead time measures how long it takes to deliver a required solution.
- Change fail rate measures the percentage of changes to production that fail.

These performance metrics are here to help managers make better decisions that will enable faster software delivery. Managers should also note that "there is no tradeoff between improving performance and achieving higher levels of stability and quality. Rather, high performers do better at *all* these measures" (Forsgren et al., 2018).

F. ORGANIC DEVELOPMENT

Literature shows that mass customization and time-to-market are enablers of sustained competitive advantage (Byrd, 2001). Mass customization is delivering a customer's need in a cost-effective way (Byrd, 2001). It is a dynamic process in that it does not require the same sequence for every deployment and is customer focused. It uses "pre-engineered modules" and configures them however is necessary for the specific project or customer. In software, mass customization is analogous to incremental and iterative development that relies on customer feedback. An aspect of agile development is common code repositories that are available for use on other projects. When and how they must be used is not defined but access to these repositories saves developers from writing the same code twice.

In software development, time is a critical piece of the puzzle, whether the project is on schedule is critical but equally important is whether what is being developed is still relevant and what the warfighter needs. In software development, time is also critical in that it must fill a capability or security gap quickly before an adversary can exploit it. Byrd describes time as time-to-market and delivery performance. Time-to-market refers to the time between requirements gathering and a minimum viable product. Delivery performance is the ability to deliver the product faster than competitors or in the military's case, faster than adversaries can exploit the vulnerability and its relevancy. Mass customization is important within time-to-market, especially for industries with customized products, like DOD (Byrd, 2001). Shorter product development and delivery provide a strategic advantage (Byrd, 2001).

Developing software organically will enhance our ability to produce mass customized software and more closely control the product delivery time with the goal of sustaining our competitive advantage. Internal development may also decrease the time-

to-market as these software factories, like Kessel Run, can iteratively develop and deploy rapidly, instead of relying solely on contractors and the software acquisition process. The ultimate benefit is that internal software development is an enabler of the military's core competencies that create a sustained competitive advantage (Byrd, 2001).

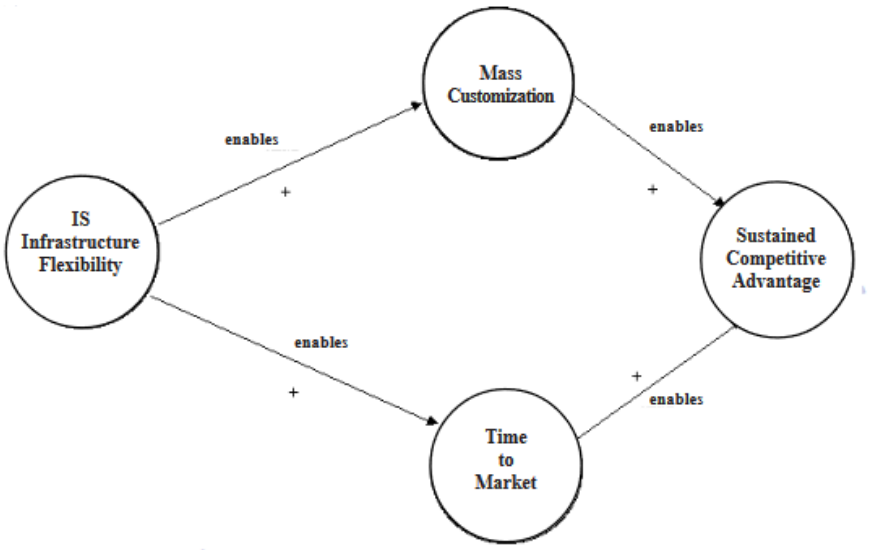

Figure 8. Sustained Competitive Advantage Map. Source: Byrd (2001).

The DOD's software acquisition process is antiquated and is unable to maintain pace with the current software demands. With the right level of authorization and coding talent in the Defense Department and working side by side with the DOD, the warfighter will receive software updates within minutes to days instead of months or years. DOD hardened software factories provide tremendous cost and time savings. A McKinsey study that reviewed 5,400 IT projects found that 66 percent were over budget, 33 percent experienced a schedule overrun, and 17 percent did not meet the expected benefits (Bloch et al., 2012). This can no longer be the status quo for the DOD as our adversaries will exploit our vulnerabilities.

Agile methodologies, like DevOps, provide an opportunity to improve the DOD's software, as well as prevent program cancellation. The ability to develop the software internally using an agile approach will allow the user to interact with a functional product

to provide feedback that will improve the quality. This interaction also provides real-time status and metrics of software completion since there is a live version. Agile developers use a myriad of tools that add security, flexibility and efficiencies into the process.

G. CLOUD MIGRATION

1. Policies and Directives

In alignment with the federal government's IT infrastructure consolidation efforts, the DOD Cloud Strategy, signed off by the Deputy Secretary of Defense in December 2018, aims to use the cloud to further reduce the amount of data centers, increase security posture through integrated Defensive Cyber Operations, and provide rapidly deployable common services (Department of Defense [DOD], 2018a). In July 2018, the DOD released the final Request for Proposal for an enterprise-wide cloud called Joint Enterprise Defense Infrastructure (JEDI) Cloud (DOD, 2018b). JEDI is a $10 billion cloud contract that would overhaul DOD's IT infrastructure over a period of 10 years, if all contract options are exercised (DOD, 2019b). The solicitation called for Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) for both classified and unclassified systems (DOD, 2018b). In October 2019, the JEDI contract was awarded to Microsoft, however, this decision is being challenged by Amazon Web Services due to alleged political bias and intervention (DOD, 2019b; Dastin, 2020).

2. Cloud Characteristics and Benefits

The drivers behind cloud migration are fueled by the essential characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service (Mell & Grance, 2011).

- On-demand self-service allows customers quick access to computing resources they need without having to go through human channels.
- Broadband network access allows customers access to their contents through a variety of computing platforms such as mobile phone, tablets, laptops, and workstations.

- Resource pooling is a way for the provider to combine their computing resources to serve multiple customers thereby managing their resources more efficiently.

- Rapid elasticity allows swift allocation of resources based on customer demands. This allocation process is often automatic and appears seamless from the customer's perspective.

- Measured service is similar to utilities monitoring. The customer's resource usage such as storage, processing, or bandwidth, can be monitored, controlled, and reported. The customer would only be billed based on their usage.

These characteristics offer organizations the benefits of efficiency, agility, and innovation (DOD, 2012). Organizations can cut the cost and time of building and maintaining their own data centers and instead use those resources to focus on their businesses (Hochstein et al., 2011). As the machine learning and AI fields continue to grow, cloud computing also enables organizations to conduct big data analytics by accessing and paying for large computing resources on an as-needed basis. The DOD plans on using the characteristics of cloud computing for objectives such as enabling exponential growth, enabling AI and data transparency, and extending tactical support for warfighters at the edge (DOD, 2018a). Although not mentioned in the article, the cloud enables software development through a flexible, cost-efficient, and collaborative environment.

3. Cloud Service Models

Cloud service models are the level of service the customers choose that meet their needs. The service models, as defined by National Institute of Standards and Technology (NIST), are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The three service models offer various levels of capability to the consumers (Mell & Grance, 2011).

- IaaS provides the fundamental computing resources to the consumer while allowing the consumer to maintain control over operating systems and applications.
- PaaS includes services provided by IaaS with the addition of operating systems, leaving the control of deployed applications to the consumer.
- SaaS provides all the services included with IaaS and PaaS, leaving the consumer with the decision on the types of applications required.

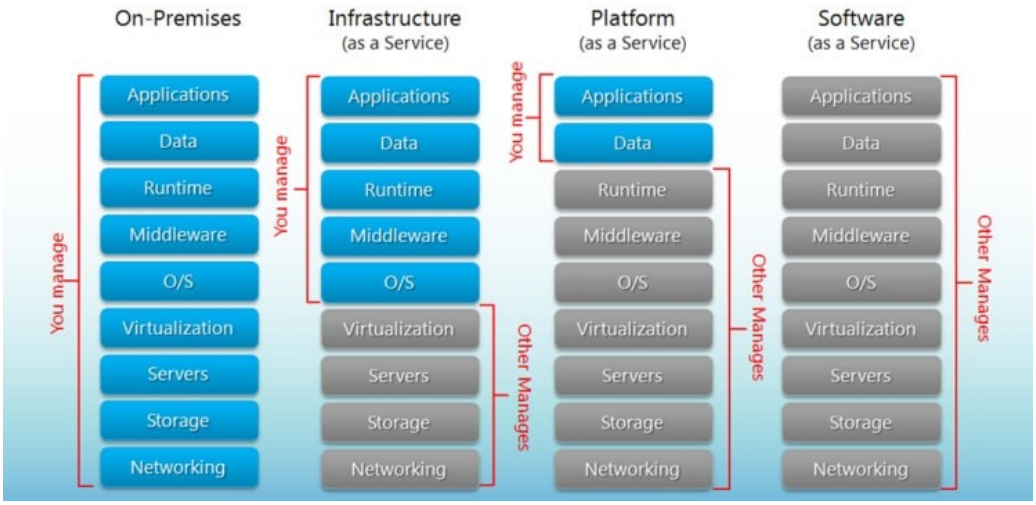

Figure 9. Cloud Service Models. Source: Hou (n.d.).

The DOD is currently focused on IaaS and PaaS due to the flexibility of running various existing applications. Other cloud service models are emerging as cloud computing grows and evolves. One service model, Data as a Service (DaaS), has potential interest for the DOD as an on-demand access to big data for analytic purposes due to its ability to standardize the massive amount of existing data to make DOD information visible and accessible to all authorized users (DOD, 2012). Another up and coming service model of interest is Containers as a Service (CaaS). CaaS would streamline the build/test/deploy pipelines in DevOps since it allows applications to work and run as if built locally, thus eliminating environmental inconsistencies and making testing and debugging easier (IBM,

n.d.). It would be a useful tool for development teams as the DOD transitions to DevSecOps and development in the cloud.

4. Cloud Deployment Models

Cloud deployment models are defined by how the cloud infrastructure is provisioned. Deployment models include private, community, public, and hybrid (Mell & Grance, 2011).

- Private cloud is provisioned for the exclusive use of a single organization. The actual ownership and management of a private cloud may fall on the organization, a provider, or a combination of the two. The physical location of the private cloud could be on or off the organization's premises.

- Community cloud is provisioned for use for a specific set of customers with shared interests such as missions and security requirements. These customers could be from the same or different organizations. The ownership and management of a community cloud may fall on one or more organizations in the community, a provider, or a combination of above. The physical location of the community cloud could be on or off organizations' premises.

- Public cloud is provisioned for use by the general public. The ownership and management of a public cloud can fall on one or more organizations. The physical location of a public cloud is on the premises of the provider.

- Hybrid cloud is a combination of the different deployment models. The different clouds within a hybrid cloud remain as separate entities but are connected to allow for certain data exchange. The ownership, management, and physical location varies depending on individual clouds that make up the hybrid cloud.

DOD's General-Purpose cloud, to be fulfilled by the JEDI contract, is likely to be a private or community cloud used by its service components (DOD, 2018a). Its Fit-for-Purpose clouds, fulfilled by various vendors, are likely to employ multiple deployment models based on different requirements (DOD, 2018a). The completed DOD Enterprise Cloud Environment would be a hybrid cloud encompassing both the General Purpose and Fit-for-Purpose clouds (DOD, 2018a).

5. Cloud Migration Models

The two cloud migration strategies are cloud hosting and cloudification (Mendonca, 2014). Cloud hosting is moving modified legacy applications into the cloud (Mendonca, 2014). Cloudification is either rewrite the legacy applications from scratch or replace the legacy applications with suitable cloud services (Mendonca, 2014). Each strategy contains different solutions depending on the constraints of the legacy applications and the cloud environment. Microsoft Azure narrows cloud hosting solutions down to rehost, refactor, and rearchitect; and cloudification solutions to rebuild and replace (Microsoft Azure, 2019; Moore, 2018).

a. Cloud Hosting Solutions

- Rehost: The legacy application is moved into the cloud without being modified, this is also known as "lift and shift."
- Refactor: The legacy application code is restructured and optimized for the cloud environment without changing its external behavior.
- Rearchitect: The legacy application code is extensively modified to take advantage of the cloud environment.

b. Cloudification Solutions

- Rebuild: The legacy application code is rewritten from scratch with the same specifications and requirements.

- Replace: The legacy application is replaced with a new application with updated requirements.

The migration of a large enterprise system, such as the DOD's, into the cloud is especially challenging due to its wide array of heterogeneous and complex legacy applications and data systems (Mendonca, 2014). One of the many challenges DOD will be facing is deciding which of the above migrating solutions to use for each of its applications based on the characteristics of the applications.

H. SOFTWARE DEVELOPMENT IN THE CLOUD ENVIRONMENT

1. Benefits and Challenges

Traditionally, software development occurs on an offline machine. This machine would require a dedicated physical space, access to power, and personnel to maintain it. It would provide a fixed environment with limited resources, and developers would have to be onsite to access it. Software development in the cloud would eliminate the need for a dedicated physical space and maintenance crew since the development environment would be hosted in the cloud and maintained by the cloud provider. The cloud environment would provide developers the flexibility to customize the development environment, access to virtually unlimited computing resources, and access to the development environment from anywhere with an internet connection (Mall et al., 2017). Other benefits of development in the cloud include increased data reliability, easier group collaboration, and quicker deployment with less probability of failures (Mall et al., 2017).

The top challenges of development in the cloud environment are security risks and giving up control of the environment over to the cloud provider (Al-Rousan, 2015). Security risks of hosting the development environment in the cloud is a trade-off of being able to access the development environment from anywhere with an internet connection. Unauthorized access to the development environment can occur in situations with stolen credentials or by cloud provider employees (Al-Rousan, 2015). Aside from the potential unauthorized access that comes with working with cloud providers is the potential challenge of communication between software developers and the cloud provider (Patidar et al., 2011). The developers will have varied abilities to control the development

environment, including outages, depending on the chosen cloud service and deployment model. Developers will have to communicate their desired changes to the cloud provider for changes they don't have access to.

2. Agile Development in the Cloud

The DOD has been locked into the traditional waterfall method of software development for many years due to its acquisition environment. In recent years, the DOD has taken an interest in the Agile approach to help it rapidly develop and deploy applications. The cloud environment has shown to be complementary to Agile development with many success stories following Salesforce's release of their whitepaper on their transition of moving Agile development into the cloud (Mall et al., 2017). Salesforce's whitepaper credited cloud computing for the elimination of inefficient distribution requirements that can slow the Agile development process down (Salesforce, 2008). The elimination of the delay in distribution enables continuous feedback between developers and customers leading to a significant decrease in days between major releases and an increase in features delivered (Salesforce, 2008; Mahmood & Saeed, 2013). Other complementary factors that cloud computing brings to Agile development include enhanced testing support and transparency (Younas et al., 2016). Enhanced testing support is made possible by many test servers that the cloud can support (Younas et al., 2016). Increased transparency is achieved through capturing of shared data across the development environment by developer services and can be used to help measure and manage the project (Tuli et al., 2014).

I. SUMMARY

While the as-is state of software development in the DOD is concerning, it is also optimistic. The optimism surrounding new start-up units that are born agile, like Air Force's Platform One and the DevSecOps Initiative, show a new appreciation for software and the importance of getting it right. There is no one right method but the authors highlighted multiple that are proven in industry and are most appropriate for the DOD along with some of the tools used. Finally, the chapter concluded with describing how the cloud initiative enables faster development to ensure the United States maintains its

military superiority, and the performance metrics managers could use to help steer their development teams in delivering software faster.

III. RESEARCH METHODS

A. INTRODUCTION

The methodology used for this research is in the form of qualitative case studies through interviews with developers and program managers familiar with the software development process. Qualitative research, one of the three types of research designs, "is a means for exploring and understanding the meaning individuals or groups ascribe to a social or human problem," using "open ended questions, emerging approaches, text or image data" (Creswell, 2009). A case study is a type of qualitative "strategy of inquiry in which the researcher explores in depth a program, event, activity, process, or one or more individuals" (Creswell, 2009). The authors chose qualitative research design for this research as a way to pose open-ended questions in a bid to explore and understand how TSO and other DOD organizations are developing their software, and what TSO should do to improve their process and evolve with the current threat landscape; hence the case study strategy of inquiry was chosen to explore the software development methodologies at three different DOD organizations in order to gain an understanding and compare and contrast the development methodologies used, and recommend a way forward for TSO.

B. RESEARCH METHOD

1. Case Study

The case study method is "the preferred strategy when 'how' or 'why' questions are being posed, when the investigator has little control over events, and when the focus is on a contemporary phenomenon within some real-life context" (Yin, 1994). The case study inquiry allows the researchers to capture a wider variety of variables and sources (Yin, 1994), thus allowing the researchers to interpret variables that are less tangible and harder to measure. The case study inquiry strategy allowed the authors to ask open ended questions on how the DOD organizations are developing their software, capturing answers that are not easily quantifiable. This method also allowed the authors to review qualitative documents, and audio and visual materials from the private sector, the current leader of software development. The biggest limitation the authors encountered with the inquiry

strategy is the level of details some organizations are willing to divulge due to security concerns.

2. Semi-structured Interview Questions

The semi-structured interview questions were developed with the help of the topic sponsor. The questions focused on the processes and tools—two out of the three pillars of software development of People, Processes, and Tools (Koch, 2009). Questions regarding testing and plans for cloud migration were also posed. Many of these questions are intertwined due to the interconnectedness between the development process and tools used during the SDLC. The interviews start with a general overview of the development methodology employed by the organization then move into processes and tools. The questions on the development process were focused on code migration between development, system integration testing, system acceptance testing, and production environments. These questions answer the *who*, *when*, *where*, and *how* of code migration between the environments. The questions on tools are partially overlapped by the *how* question asked about the migration process, but with more details on *what* and *why*. Questions on code testing dig further into both the process and tools used to accomplish the task.

All interviews started with a general and overview questions about the organization and the organizations it supports. The below questions are some of the questions asked by the interviewers followed by a brief discussion about the applicability of such questions:

a. Describe Your Software Development Methodologies

Beginning with this question, respondents provided the interviewee an opportunity to explain which common software development methodology their organization utilizes as well as provides the composition of the team size and interactions amongst different members of the team. It was purposefully left open-ended in order to allow the interviewee to describe the methodology in any way they determine is appropriate. This is important as it generally follows one of the well documented types of methodologies and allows the interviewers to observe the methodology described in practice versus the textbook

definition. Follow on questions that stemmed from this response became more specific depending on the organization.

b. Describe Your Code Migration Process

This is a process-oriented question. It is a three-part question that allows the interviewers to explore the nuances of different organizations migration processes from development through production. Identifying these nuances is important as it frames the major differences in the methodologies, organizations, and is the point of focus for TSO. The understanding of this process is critical to providing recommendations to TSO and add value to their organization.

c. Describe Your Software Development Tool Set

As a tool-based question, it differentiates between the multitude of tools used for version control, integrated development environments (IDE), and configuration management. This question is designed to give perspective on the different types of tools similar organizations to TSO use and to compare tools in order to provide alternative options for TSO's use.

d. Describe Your Software Testing Efforts

Testing is an important aspect of the development for both operability and security. This question focused on the testing process provides insights of how, where, and when the organizations test. The answers to this question are limited due to additional testing done by security teams outside of the development teams as well as overall security concerns. This question was emphasized due to the importance of security in DOD systems as well as the potential to slow the development and deployment cycles.

e. Cloud Migration Efforts

As the DOD focuses on the transition to the cloud, the researchers were curious as to how prepared the organizations were to transition and the effects of the transition on their software development environments.

C. SUMMARY OF PLACES INTERVIEWED

The three DOD organizations interviewed for this research were Marine Corps Technology Services Organization (TSO), Navy Supply Systems Command Business Systems Center (NAVSUP BSC), and Defense Manpower Data Center (DMDC). In-person interviews were planned prior to the start of the research, however, due to the COVID-19 pandemic, only the interviews at TSO were conducted in person. Interviews at NAVSUP BSC and DMDC were conducted over the phone and email.

1. Marine Corps Technology Services Organization (TSO)

TSO, located in Indianapolis, Indiana, is the Marine Corps' in-house development team that supports its business operations systems including, but not limited to "pay, personnel, budget execution, orders writing, accounting and installation systems" (Marine Corps, n.d.). It is a government-owned, government-operated organization composed of four divisions: Marine Corps Total Force Systems (MCTFS), Marine Corps Integrated Pay and Personnel Systems (MCIPPS), Standard Accounting, Budgeting and Reporting System (SABRS), and Enterprise Services Division (ESD). This research is focused on the three development divisions: MCTFS, MCIPPS, and SABRS. Each of these divisions is responsible for a set of products. MCTFS is responsible for the integrated military pay and personnel system. MCIPPS is responsible for Marine Online and various orders writing systems. SABRS is responsible for SABRS and Marine Corps Financial Integrated Analysis System (MCFIAS). Each division employs a software development methodology that suits their needs.

2. Navy Supply Systems Command Business Systems Center (NAVSUP BSC)

NAVSUP BSC headquartered in Mechanicsburg, Pennsylvania, is the Navy's provider for Information Technology/Information Management solutions in "functional areas of logistics, supply chain management, transportation, finance and accounting" (Naval Supply Systems Command, n.d.). BSC is a government-owned, government-operated organization composed of seven departments: Code 91 Business Management & Comptroller, Code 92 Logistics Solutions, Code 93 Core Business Solutions, Code 94

Technology Services, Code 95 Data/Analytics Solutions, Code 97 ERP Business Office & FLC Support, and Code 98 ERP Services. The department interviewed in this research is Code 92. They are made up of 25 to 30 application development teams and are responsible for providing a wide variety of logistics and supply systems.

3. Defense Manpower Data Center (DMDC)

DMDC is the IT provider for Defense Human Resources Activity (DHRA), serving under the Office of the Secretary Defense (OUSD). It provides identity management for both past and present DOD personnel. Other than identity management, DMDC provides a wide range of services involving personnel, manpower, training, and finance to an extensive list of customers. Its mission can be summarized in three parts. The first is to "collect and maintain an archive of automated manpower, personnel, training, and other databases for the Department of Defense" (DMDC, n.d.). The second is to "support the information requirements of the OUSD for Personnel & Readiness (P&R) and other members of the DOD manpower, personnel, and training communities with accurate, timely, and consistent data" (DMDC, n.d.). Lastly, to "operate DOD-wide personnel programs and conduct research and analysis as directed by the OUSD P&R" (DMDC, n.d.).

Due to the size and variety of DMDC's portfolio, some projects are government-owned and operated, while others are government-owned, contractor-operated.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ANALYSIS

A. INTRODUCTION

This chapter will summarize the responses from the organizations interviewed on their development methodology, code migration process, development tool set, testing efforts, and cloud migration efforts. A discussion will follow the interviewees' responses, covering changes TSO could adopt to optimize its development process.

B. INTERVIEWEE RESPONSES

1. TSO

a. Development Methodology

Each of TSO's software development branches use a development methodology best suited to their needs. MCTFS uses an iterative method similar to the waterfall due to the scale and use of financial and personnel data, making it an imperative to ensure the code does what it is intended to do before being deployed to production on the mainframe. MCTFS' development teams are made up of a minimum of a designer, programmer, and tester, and its teams range from 3 to 10 members. MCIPPS uses Scrum for most of its teams with the tiger teams employing the Kanban method. The department is split 60 percent government civilians and 40 percent contractors. SABRS uses Scrum with 4-weeks sprints consisting of three per team. As much as MCIPPS and SABRS try to stay true to an agile approach, there are still steps in the development process that require queuing and waiting for an approval to move to the next step.

b. Code Migration Process

The code migration process from development to production are similar for each of the branches with some variation on the tools used. The first environment is the development (DEV) environment. Programmers write their code and conduct unit testing in DEV. Once the code passes unit testing, it is promoted to the System Integration Testing (SIT) environment to make sure it works as intended with all systems it interacts with. From SIT environment, the code is moved to System Acceptance Testing (SAT), or User

Acceptance Testing (UAT) environment, for further testing and customer validation. Once the code is ready for delivery, it is migrated to production (PROD). For each branch, once the code is deemed ready for migration to the next environment, a work ticket is submitted to an application administrator in the Enterprise Services Division for migration.

c. Development Tool Set

The development toolset each branch uses are more varied. MCTFS uses mainframe specific IDEs which are Topaz workbench, IDz, and Roscoe/TSO (Time Sharing Option). MCIPPS and SABRS uses IntelliJ IDEA, Eclipse, and Visual Studio Code as their IDEs for their work on mid-tier servers. These are limited sets of IDEs programmers can choose from based on their preference. MCIPPS and SABRS also use other plug-ins and extensions with their IDEs to help streamline their work while coding. For version control and configuration management, MCTFS uses ISPW. MCIPPS uses Subversion for version control and Jenkins for configuration management. SABRS uses Subversion for both version control and configuration management.

d. Testing Efforts

MCTFS conducts security testing for all its web services through ReadyAPI, and vulnerability assessment is done by an Information System Security Officer (ISSO). MCIPPS and SABRS developers both use HP Fortify to conduct vulnerability testing during their development process with further assessments completed by the ISSO. MCTFS uses manual testing for the acceptance testing and a mix of manual and automated testing for regression and performance testing. Most of MCIPPS testing, to include unit testing, system integration testing and system acceptance testing is conducted manually.

e. Cloud Migration Efforts

TSO is in the early phase of the cloud migration effort and is using Microsoft's Cloud Adoption Framework as a guideline for the migration. While TSO is planning on using JEDI as the contract vehicle for the entire migration, it is unknown when JEDI will become operational. In the meantime, MCIPPS is exploring different options to move their development teams into a cloud based DevSecOps environment.

2. NAVSUP BSC

a. Development Methodology

The main development methodology employed by Code 92 at NAVSUP BSC is waterfall. Some teams opt to employ Agile, Scrum, and pseudo Agile methodologies. Pseudo Agile is a hybrid between waterfall and Agile. The waterfall teams are on quarterly or yearly release schedule. The Agile and Scrum teams are on 3 to 4-weeks release schedule, and the pseudo-Agile teams are on 6 to 8-weeks release schedule.

b. Code Migration Process

The code migration process at Code 92 is like the process at TSO. Once testing criteria are satisfied in each environment, a work ticket is submitted to the operations team at Code 94 for the code to be migrated to the next environment.

c. Development Tool Set

Code 92 uses .NET and Azure DevOps in their development tool set. The developers use Visual Studio, JDeveloper, Eclipse, SQL Developer, and SQL Server Management Studio (SSMS) as their IDEs. Version control is accomplished by Git and Azure DevOps, and configuration management is accomplished with Azure DevOps.

d. Testing Efforts

Code testing in DEV, TEST, and PROD are done manually within the development teams. User Acceptance Testing are done with the customers for big releases. Vulnerability testing are done through HP Fortify, VS code analysis, SpotBugs, and FindBugs. Further testing is conducted by ISSO and Information System Security Manager (ISSM) in a separate department, Code 94.

e. Cloud Migration Efforts

Cloud migration effort is a slow process at Code 92. The department is exploring options while keeping an eye on JEDI progress.

3. DMDC

At the writing of this thesis, DMDC is in the process of migrating to the cloud environment and adopting DevSecOps. The interview gave the authors a small glimpse into the transition process since many of the questions were considered sensitive. Overall, the code migration process at DMDC is dependent on the underlying platform the system is developed and its corresponding DevSecOps build pipeline. Most development tools DMDC uses are Java-based with the main IDE being Eclipse. Source code and version control are managed by Git and GitLab. Regarding code testing, the development teams try to achieve an 85% code coverage during unit testing. Independent tests are also done by quality assurance teams. Other tests and assessments are done as a part of build pipelines.

C. DISCUSSION

The focus of software development in the DOD is providing functional, secure software to the warfighters in a timely manner. The DevSecOps methodology is currently the best suited methodology to accomplish that task. DevSecOps focuses on reducing the mean time to production and increasing the deployment frequency, which are accomplished through automated development pipelines with security baked-in from the start. The effectiveness of DevSecOps in producing software is backed up by various metrics. Air Force's Platform One's metrics show an average deployment frequency of 20.8 per day (C. Chew, PowerPoint slides email to authors, July 30, 2020). That is a minimum of 20.8 changes ready to be delivered to the customer in a single day versus one deployment of batched changes every two to four weeks using Scrum.

As the DOD is pushing for an agile and secure software development process, the discussion will focus on how TSO can adopt DevSecOps for its mid-tier development teams. Specifically, the changes required to its people, process, and tools. The discussion will also cover possible challenges and limitations TSO might face while implementing DevSecOps, and some of the readily available service options for implementation.

1. Implementation Requirements for DevSecOps

At TSO, MCIPPS and SABRS would benefit greatly by leveraging automated testing and reengineering their testing processes. Both units described their testing processes as mostly manual and that operations and security staff move the code from one environment to the next. Automation provides speed that might be lacking from the current process. A restructuring of personnel is another aspect of DevSecOps that will enhance TSO's capabilities. Most of the promotion of code through the lifecycle is conducted by operations or security personnel (system administrators/application administrators or information system security officer/engineer). The scope of the developers' testing is limited, but by creating one cohesive team the operations and security team members can conduct simultaneous testing earlier in the development lifecycle allowing program efficiency. DevSecOps will allow developers to provide secure code by "shifting security left" and by ensuring everyone is responsible for security. Automation will provide speed and consistency during the build, testing, integration, and delivery processes. Transitioning to DevSecOps will require changes to culture, structure, processes, and tools that are currently in place. DevSecOps is more than a methodology, it is a philosophy.

a. People

For TSO to successfully transition to DevSecOps, the most critical aspect involves their people. Educating the entire organization on the close collaboration necessary in order to effectively implement DevSecOps is likely the toughest hurdle to overcome. Personnel that usually do not work closely with each other will be expected to develop an integrated and capable team. Communication and transparency between the teams will lead to more secure and effective applications. By integrating the development, operations, and security teams on smaller projects they will begin to break down the cultural barriers that separate them. Embedded within this philosophy is that everyone is responsible for security. A strong champion and change management leader will aide in this process, but it is incumbent on the entire leadership team to have a comprehensive plan incorporating their employees' recommendations. Restructuring personnel into more horizontally aligned teams from the beginning will assist in a smoother transition.

Training developers on secure coding practices is critical to the "shift-left" principle that seeks to embed security into the development process early. The first line of defense for security and speed is secure code. TSO leadership should empower the developers with the necessary skills to code securely. Instead of mandating training from the top-down, creative approaches will likely create more buy-in. The training should be developed by a combination of senior developers and security members to facilitate teamwork and minimizing negative stigmas between the groups. This training needs to be in TSO's preferred language, so it is relevant.

Most importantly, DevSecOps is a cultural change that TSO must embrace. A security first, teamwork-oriented mindset is the first step in adopting DevSecOps. The process and tools changes are only effective if teams are willing to collaborate and everyone believes that security is their responsibility.

b. Process

A business processing reengineering effort is required to investigate the current process in order to identify processes that automation replaces. The current structure of TSO slows down the process when development teams must manually enter the queue to get their codes tested or approved to be promoted to the next environment. DevSecOps success is focused on automation and integration. TSO's manual operations and approval processes are great inhibitors to an effective DevSecOps approach. The manual handoffs occur when developers need to get their code tested by the security team or promoted to the next environment by the operations team. A thorough examination of the current process to identify security and operational chokepoints will focus the necessary procedural changes, which generally happen between environments.

Automation is essential to DevSecOps. As a process, it is faster. It will allow TSO to reduce the meantime to production. This is the time between the need for new features until they are running in production. A fully automated risk monitoring and mitigation process across the development lifecycle provides more in-depth security awareness. With automation, TSO can eliminate human error and the latency between code migrations. Most releases happen monthly or bi-annually but with automation and DevSecOps, TSO

could increase their deployment frequency from weeks to days with fewer change failures and less costly security errors. At Platform One, a DevSecOps organization, their lead times are 30 minutes for minor changes and two days for major ones (C. Chew, PowerPoint slides email to authors, July 30, 2020).

Speed is one of the most important metrics for measuring software (DIB, 2019). CI/CD is a part of the DevSecOps process that is responsible for providing speed and requires automation. Continuous integration is when developers enter code into the main branch which then triggers tests to run against the build before committing it to the main branch. Continuous delivery implies that you have automated your release process. CI/CD adds speed to TSO's process by eliminating the current "gate-keeper" method whereby a person approves and migrates code. DevSecOps and CI/CD would effectively reduce the lead time for changes from weeks, to hours or days depending on the complexity of the change.

c. Tools

DevSecOps cannot be bought through a package of tools since DevSecOps is a philosophy that includes the interaction between people, process, and tools, however there are certain tools that facilitate the DevSecOps methodology. Many tools used in DevSecOps are already being used by the development teams at TSO. While planning for an automated pipeline, TSO will need to decide if MCIPPS and SABRS will use the same toolchain. Using the same tools in the pipeline is likely to decrease the cost of implementing and maintaining the pipeline, but the two departments might require the use of different tool.

It is challenging to differentiate between people, process, and tools at times, since the premise behind the methodology is the interconnectedness of the triad. For example, in order to leverage automation, leadership must be willing to give up control of the manual process and trust the tools to work as designed. This trust and the trust of the three different teams to code securely is part of larger cultural change. Many of the tools that TSO already uses are compatible with the DevSecOps methodology, like JIRA, Jenkins, GitHub, Fortify and Splunk to name a few.

2. Challenges and Limitations

Implementing DevSecOps is not without challenges and limitations. From an organizational point of view, the biggest challenge TSO will have to overcome is the challenge of changing its current culture. Successful adoption of DevSecOps will require all members of the team to be on board. Team members must utilize soft skills in order to form a cohesive and productive team. The transition will also require developers to learn new skill sets and practice developing secure code. From a technical point of view, the biggest challenge is implementing the continuous delivery part of the CI/CD pipeline. Continuous delivery involves automated testing and promotion to the next environment, leaving deployment to production the only manual part in the process. The adoption process for continuous delivery will require trust in automation and likely changes to current policies. The current policies are designed to mitigate risks through approvals authorities at higher levels, however this slows down delivery. This change is as much procedural and cultural since the trust in automation is a must-have for success.

Other challenges and limitations TSO might face include budget and prioritizing between cloud migration efforts and DevSecOps adoption. Adopting DevSecOps will incur the initial cost to set up the development environment and train employees on new tools and practices of DevSecOps. Recurring cost includes licensing and maintenance of the development environment. A budget constraint might limit the ability to adopt DevSecOps, and it is a likely possibility as fiscal austerity looms over the federal government. The cloud migration planning efforts might also limit resources available for DevSecOps adoption depending on TSO's prioritization. TSO should consider prioritizing DevSecOps first if it must prioritize between cloud migration and DevSecOps. Having a good DevSecOps process and CI/CD pipeline in place will aid with cloud application development when cloud migration occurs.

3. Available Options

DevOps and DevSecOps as a service are becoming popular in the commercial sector with companies like Amazon Web Services and Microsoft Azure offering their services. DevSecOps as a service is considered as PaaS where the provider maintains the

development environments. There are various efforts in the DOD to provide DevSecOps development environments for its organizations. As of May 2020, Air Force's Platform One was designated as one of DOD's enterprise service provider for DevSecOps (DOD, 2020). The Marine Corps also has a DevSecOps solution in the works with Marine Corps Business Operations Support System (MCBOSS). The benefits of choosing a DOD or service approved DevSecOps service provider is eliminating the time and paperwork required for the Authority to Operate (ATO) when setting up the development environment and eliminating the need to maintain the environment. The potential drawbacks to using a DevSecOps service provider are cost, tool availability, and configuration control over the development environment.

4. DevSecOps and the Cloud

The cloud is considered a key enabler for a full implementation of DevSecOps' CI/CD pipeline. Cloud computing provides scalability and flexibility for seamless continuous integration. TSO is in the early phase of their cloud migration effort at the writing of this thesis and is planning on using JEDI as the contract vehicle for the entire migration. It is unknown when JEDI will become operational. This will provide TSO more time with cloud migration planning as well as optimizing their development process for the cloud. DevSecOps is traditionally associated with cloud application development, but that does not mean DevSecOps practice cannot be used to optimize the current development process. By adopting DevSecOps early, TSO will place itself in an advantageous position for the eventual migration to JEDI cloud.

D. SUMMARY

The DevSecOps transition will result in require complete commitment from everyone at TSO but it will result in securely developed code and faster releases. Communication and collaboration must replace the siloed approach between developers, operations, and security staff. The cultural transition will take time and the involvement of a strong champion will be important. Changing the culture will drive the acceptance of new processes, like automation, and leverage the full capabilities of the toolchain. This

transition will make TSO more efficient and effective in achieving their mission to “deliver secure, auditable, and proficient capabilities to their stakeholders.”

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This comprehensive research sought to explore the software methodologies of organizations like TSO in order to provide a road map to facilitate the transition to DevSecOps. It strived to answer four research questions:

1. How could TSO go from a multitude of approaches to DevSecOps?
2. How will the transition to DevSecOps impact the software developers at TSO to continue to develop critical software securely and efficiently?
3. How do similar software development activities within the Department of Defense develop software and might the Marine Corps consider similar approaches?
4. What is the optimal way to migrate the existing software development environment to the cloud, and continue to allow access to the requisite software development tools?

The first phase of the research involved a comprehensive literature review and interview questions formulation. The questions provided the framework for the researchers during the interviews of personnel and software development teams from TSO, NAVSUP BSC, and DMDC. The focus was on DevSecOps and the supporting resources.

DevSecOps is the culture and practice of integrating the software development, security, and operations teams (DOD, 2019a). TSO leaders are interested in transitioning to the DevSecOps model in order to provide frequent, smaller, and more secure releases. Teams that are already operating in an agile manner are best suited to adapt their procedures and structure to benefit from the advantages of DevSecOps since it is a significant culture shift. It incorporates security throughout the development lifecycle which minimizes costs relating to resolving security issues. When developers are taught to code securely, it decreases the amount of security errors and increases the chances of finding and fixing security issues early.

DevSecOps is not without its limitations. The most significant hurdle to practicing DevSecOps is likely the required culture change and collaboration amongst the developers, operations staff, and security personnel. Embedded within this is that the developers must code with security best practices instead of expecting to add it as a bolt-on late in the development lifecycle. Due to the frequent release of code the two teams must coordinate closely with operations to ensure the environment and the infrastructure can handle the changes. The three teams must work in unison to ensure that secure, reliable, and functional code can be delivered rapidly.

B. FINDINGS AND RECOMMENDATIONS

Out of the three departments within TSO, MCIPPS and SABRS are best suited for a transition to DevSecOps due to their familiarity with agile methodology and use of tools compatible with DevSecOps. Currently, the development, operations, and security teams are siloed which is an inhibitor to the collaboration required for DevSecOps to work efficiently. Collaboration between the three teams is tightly coupled with the development pipeline. The current pipeline is mostly manual with the developers handing off their code to the security and operations teams for testing and promotion to the next environment. The manual handoffs create inefficiencies and delays. Several tools used by the teams are compatible with DevSecOps, and can be configured for automation into a CI/CD pipeline. The departments within TSO, specifically MCIPPS and SABRS, have been optimizing their development process individually. Development teams have started experimenting with automated unit testing, a first step for continuous integration. Moving forward it will require formal formation of DevSecOps teams to effectively configure and implement a CI/CD pipeline.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

With the changing landscape of technology and the speed at which it changes, there is ample opportunity for continued work. DevSecOps is a relatively new endeavor by the DOD that it must leverage in order to remain flexible and deploy secure software releases in a timely manner. As the DOD becomes more digital, these topics are closely linked to national security.

1. Combat Systems

Combat systems are increasingly becoming more software intensive whether through autonomous systems or through manned platforms like the Joint Strike Fighter (JSF). As this trend continues, software and security must be delivered rapidly and securely in order to remain combat effective. DevSecOps offers opportunities to deliver reliable software patches quickly increasing the warfighter's competitive advantage.

2. Mainframes

Software development teams in mainframe environments are still predominately using waterfall methodology as their development process. New tools are becoming available that supports a more agile approach that shifts security to the left. Research could be conducted on how organizations with mainframe environments can adopt a more agile methodology.

3. Culture

DevSecOps involves a culture change. It is more than purchasing new tool kits and incorporating those into the developers' repertoire. A fundamental shift in management and interactions between various stakeholders must occur in order to effectively implement DevSecOps. A broader culture change is necessary as well to meet our needs in developing software. An interested author might explore the change management side of implementing a new development methodology, especially one as interaction heavy as DevSecOps.

4. Talent Pool

As software and security pervade all the DOD's systems, it must be able to recruit and retain competent talent. A future researcher might be interested in developing a roadmap for recruiting and retaining digital talent, including education, training and developing alternate career paths for the DOD's digital workforce. The DOD cannot fall behind modern digital trends and practices.

5. Acquisition Process

Software is not like hardware. It becomes obsolete faster than most hardware and requires constant updates and patches. The DOD's acquisition system is designed for hardware which is to time-consuming for software. The DOD must modernize its policies and regulations to support faster development and acquisition of software and the resources that enable its development and use. This involves ensuring our networks support the required resources and that the DOD is leveraging the best practices and tools available. A robust study focused on refining the acquisition pipeline for software would likely prove valuable for the DOD.

LIST OF REFERENCES

Agile Alliance. (n.d.). *Extreme Programming*. Retrieved January 7, 2020, from https://www.agilealliance.org/glossary/xp

Air Force. (n.d.). *DOD enterprise DevSecOps initiative (DSOP)*. Retrieved May 27, 2020, from https://software.af.mil/dsop/#problemstatement

Al-Rousan, T. (2015). Cloud computing for global software development: Opportunities and challenges. *International Journal of Cloud Applications and Computing (IJCAC), 5*(1), 58–68. https://doi.org/10.4018/ijcac.2015010105

Amazon Web Services. (n.d.). *What is DevOps?* Retrieved May 27, 2020, from https://aws.amazon.com/devops/what-is-devops/

Anderson, A., Hendrickson, & C., Jeffries, R. (2000). *Extreme Programming installed*. O'Reilly.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. ...Grenning, J. (2001). *The Agile Manifesto*. Agile Alliance. http://agilemanifesto.org/

Beck, K. (1999). Embracing change with Extreme Programming. *Computer, Volume 32* (10). 70–77. https://doi.org/10.1109/2.796139

Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Pearson Education Inc.

Bloch, M., Blumberg, S., & Laartz, J. (2012, October). Delivering large-scale IT projects on time, on budget, and on value. McKinsey Digital. https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

Byrd, T. (2001, April–June). Information Technology, core competencies and sustained competitive advantage. *Information Resources Management Journal, Volume 14*(2). 27–36. https://www.igi-global.com/gateway/article/full-text-pdf/1198

Byrd, T., Lewis, B., & Turner, D. (2004, April - June). The impact of IT personal skills on infrastructure and competitive IS. *Information Resources Management Journal, Volume 17*(2). 38–62. https://www.igi-global.com/gateway/article/full-text-pdf/1255

Carter, K. (2017). Francois Raynaud on DevSecOps. *Software Engineering, 93–96.* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8048652

Cois, C. (2014, June 2). A generalized model for automated DevOps. *Software Engineering Institute.* https://insights.sei.cmu.edu/sei_blog/2014/06/a-generalized-model-for-automated-devops.html

Coyne, B, & Sharma, S. (2015). *DevOps for dummies*. (2nd ed). John Wiley and Sons.

Creswell, J. (2009). *Research design: Qualitative, quantitative, and mixed methods approaches* (3rd ed.). Sage Publications.

Dastin, Jeffrey. (2020, January 22). Amazon asks court to pause Microsoft's work on Pentagon's JEDI contract. *Reuters*. https://www.reuters.com/article/us-amazon-com-pentagon/amazon-asks-court-to-pause-microsofts-work-on-pentagons-jedi-contract-idUSKBN1ZM0FB

Defense Innovation Board (2019, May 3). *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 Manpower Data Center. (n.d.). *DMDC overview*. Retrieved July 13, 2020. https://www.dmdc.osd.mil/appj/dwp/dmdc_overview.jsp

Department of Defense. (2012). *Cloud computing strategy*. Retrieved June 10, 2019. https://apps.dtic.mil/dtic/tr/fulltext/u2/a563989.pdf

Department of Defense. (2018a). *DOD cloud strategy*. https://media.defense.gov/2019/Feb/04/2002085866/-1/-1/1/DOD-CLOUD-STRATEGY.PDF

Department of Defense. (2018b, July 26). *Draft JEDI cloud RFP #HQ0034-18-R-0077*. https://beta.sam.gov/opp/3860a4f4fe9d9ffc31e722ece82a143c/view

Department of Defense. (2019a, August 12). *DOD enterprise DevSecOps reference design*. https://dodcio.defense.gov/Portals/0/Documents/DOD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583

Department of Defense (2018c). *Design and Acquisition of Software for Defense Systems.* https://dsb.cto.mil/reports/2010s/DSB_SWA_Report_FINALdelivered2-21-2018.pdf

Department of Defense. (2019b, October 25). *Contract for Oct. 25, 2019*. https://www.defense.gov/newsroom/contracts/contract/article/1999639/

Department of Defense. (2019c, September 23). *Agile metrics guide*. https://www.dau.edu/cop/it/DAU%20Sponsored%20Documents/Agile%20Metrics%20v1.1%2020191122.pdf

Department of Defense. (2020, May 22). *Designation of enterprise service provider for DevSecOps*. https://software.af.mil/wp-content/uploads/2020/05/DoD-CIO-Signed-Memo-Enterprise-Service-Provider-for-DevSecOps.pdf

Ebert, C., Gallardo, G., Hernantes, & J., Serrano, N. (2016, May-June). DevOps. *IEEE Software Volume 33*. 94–100.https://doi.org/10.1109/MS.2016.68

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate*. Google Books.

Hochstein, L., Schott, B., & Graybill, R. (2011). Computational engineering in the cloud: Benefits and challenges. *Journal of Organizational and End User Computing (JOEUC), 23*(4), 31–50. https://doi.org/10.4018/joeuc.2011100103

Hou, T. (n.d.). IaaS vs PaaS vs SaaS enter the ecommerce vernacular: What you need to know, examples & more. Ecommerce Technology. *Big Commerce*. https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas/#the-key-differences-between-on-premise-saas-paas-iaas

IBM. (n.d.). *What is Containers as a Service (CaaS)?* Retrieved May 21, 2020. https://www.ibm.com/services/cloud/containers-as-a-service

Jeffries R. & Lindstrom, L. (2004). Extreme Programming and Agile software development methodologies. *Information Systems Management, Volume 21*(3). 41–52. https://doi.org/10.1201/1078/44432.21.3.20040601/82476.7

Kelman, Steven (2018, April 18). Striking a blow for agile with DOD weapons systems. *The Lectern. Federal Computer Week.* https://fcw.com/blogs/lectern/2018/04/dod-agile-kelman.aspx

Koch, A. S. (2009, June 22). People, processes and tools: The three pillars of software development. *CM Crossroads*. https://www.cmcrossroads.com/article/people-processes-and-tools-three-pillars-software-development

Mahalakshmi, M., & Sundararajan, M. (2013). Traditional SDLC vs Scrum methodology – A comparative study. *International Journal of Emerging Technology and Advanced Engineering, 3*(6), 192–196. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.2992&rep=rep1&type=pdf

Mahmood, Z., & Saeed, S. (2013). *Software engineering frameworks for the cloud computing paradigm*. https://doi.org/10.1007/978-1-4471-5031-2

Mall, R., Panigrahi, C., & Pati, B. (2017). *Software development methodology for cloud computing and its impact.* https://doi.org/10.4018/978-1-5225-1721-4.ch012

Marine Corps. (n.d.). *Technology Services Organization*. Retrieved May 21, 2020, from https://www.hqmc.marines.mil/pandr/organization/Technology-Services-Organization/

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (Special publication 800–145). https://permanent.access.gpo.gov/gpo17628/SP800-145.pdf

Mendonca, N. (2014). Architectural options for cloud migration. *Computer, 47*(8), 62–66. https://doi.org/10.1109/MC.2014.203

Mersino, A. (2018, April 1). Project success rates: Agile versus waterfall. *Vitality Chicago.* https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/

Microsoft Azure. (2019, April 4). *Cloud adoption framework migration model*. https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/migrate/migration-considerations/

Misra, S., & Omorodion, M. (2011). Survey on agile metrics and their inter-relationship with other traditional development metrics. *ACM SIGSOFT Software Engineering Notes, 36*(6), 1–3. https://doi.org/10.1145/2047414.2047430

Moore, S. (2018, June 1). *7 options to modernize legacy systems*. Gartner. https://www.gartner.com/smarterwithgartner/7-options-to-modernize-legacy-systems/

Naval Supply Systems Command. (n.d.). *NAVSUP Business Systems Center*. Retrieved May 21, 2020, from https://www.navsup.navy.mil/public/navsup/bsc/

Pace, H. (2019). *Software challenges* [Class notes for MN3309: Software Acquisition Management for Strategic and Tactical Systems]. Graduate School of Defense Management, Naval Postgraduate School. https://cle.nps.edu/portal/site/c14b8156-3ade-44cf-8c34-35d2c619a7f9/tool/9caae9a2-6eaf-4de6-913a-45cd9dde5c28?panel=Main

Patidar, S., Rane, D., & Jain, P. (2011). Challenges of software development on cloud platform. *2011 World Congress on Information and Communication Technologies*, 1009–1013. https://doi.org/10.1109/WICT.2011.6141386

Pettit, S. (n.d.). Continuous Integration vs. Continuous Delivery vs. Continuous Deployment. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Ragunath, P., Velmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving a new model (SLDC Model 2010) for Software Development Life Cycle (SDLC). *International Journal of Computer Science and Network Security. 10*(1), 112–119. ResearchGate.

Rico, D. F. (2008). *Business value of Agile methods* [Presentation]. University of Virginia, United States. https://www.commerce.virginia.edu/sites/default/files/Centers/Rico%20slides.pdf

Rico, D. F. (2008). *What is the Return on Investment (ROI) of Agile methods?* [Unpublished manuscript].

Rigby, D., Southerland, J., Noble, A. (2018, May-June). Agile at Scale. *Harvard Business Review*. https://hbr.org/2018/05/agile-at-scale

Rubens, P. (2017, June 27). What are containers and why do you need them? *CIO.* https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html

Schwaber K & Southerland, J. (2017). The definitive guide to Scrum: The rules of the game. *The Scrum Guide.* https://scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

Salesforce. (2008). *Agile development meets cloud computing for extraordinary results at Salesforce.com*. The Landmark One Market. http://www.developerforce.com/media/ForcedotcomBookLibrary/WP_Agile_112608.pdf

Scrum Process. (2009, January 9). In *Wikipedia*. Retrieved January 20, 2020, from https://commons.wikimedia.org/wiki/File:Scrum_process.svg

Surianarayanan, C., Ganapathy, G., & Pethuru, R. (2019). *Essentials of Microservices Architecture*. https://doi-org.libproxy.nps.edu/10.1201/9780429329920

Thones, J. (2015). Microservices. *Software Engineering, 113–116.* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7030212

Tuli, A., Hasteer, N., Sharma, M., & Bansal, A. (2014). Empirical investigation of agile software development: cloud perspective. *ACM SIGSOFT Software Engineering Notes, 39*(4), 1–6. https://doi.org/10.1145/2632434.2632447

United States Marine Corps. (2011). *Marine Corps Enterprise Network (MCEN).* https://www.hqmc.marines.mil/Portals/156/Newsfeeds/SV%20Documents/Summaries/Marine%20Corps%20Enterprise%20Network.pdf

Webster, G., Creemers, R., Triolo, P., & Kania, E. (2017, Aug 1). Full translation: China's 'new generation Artificial Intelligence development plan' (2017). *New America.* https://www.newamerica.org/cybersecurity-initiative/digichina/blog/full-translation-chinas-new-generation-artificial-intelligence-development-plan-2017/

Winder, D. (2018, December). DevSecOps. *PC Pro, 290.* ProQuest.

Younas, M., Ghani, I., Jawawi, D., & Khan, M. (2016). A framework for agile development in cloud computing environment. *Journal of Internet Computing and Services, 17*(5), 67–74. https://doi.org/10.7472/jksii.2016.17.5.67

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California