SEAN ROBSON, BONNIE L. TRIEZENBERG, SAMANTHA E. DINICOLA, LINDSEY POLLEY, JOHN DAVIS, MARIA C. LYTELL

Software Acquisition Workforce Initiative for the Department of Defense

Initial Competency Development and Preparation for Validation

For more information on this publication, visit www.rand.org/t/RR3145

Library of Congress Cataloging-in-Publication Data is available for this publication.
ISBN: 978-1-9774-0384-1

Published by the RAND Corporation, Santa Monica, Calif.
© Copyright 2020 RAND Corporation
RAND® is a registered trademark.

Cover: monsiti/Getty Images; PeopleImages/Getty Images

Preface

The Department of Defense (DoD) has experienced persistent challenges with software development across different kinds of acquisition programs. These challenges, which include schedule delays and cost overruns, can be attributed, in part, to an overreliance on outdated software development practices and methodologies. This report, requested in July 2017, by the Deputy Assistant Secretary of Defense for Systems Engineering, Major Program Support,[1] responds to these concerns by defining modern software competencies for the software acquisition workforce. The report should interest policymakers responsible for hiring, training, and managing software acquisition professionals.

This research was conducted within the Forces and Resources Policy Center and the Acquisition and Technology Policy Center of the RAND National Defense Research Institute, a federally funded research and development center sponsored by the Office of the Secretary of Defense, the Joint Staff, the Unified Combatant Commands, the Navy, the Marine Corps, the defense agencies, and the defense Intelligence Community.

For more information on the Forces and Resources Policy Center, see www.rand.org/nsrd/ndri/centers/frp; and for information on the Acquisition and Technology Policy Center, see www.rand.org/nsrd/ndri/centers/atp or contact the directors (contact information is provided on the center webpages).

[1] As of February 1, 2018, Section 901 of National Defense Authorization Act for Fiscal Year 2017 abolished the roles of the Under Secretary of Defense for Acquisition, Technology, and Logistics (AT&L) and subsidiary Assistant Secretaries. As a result, the project sponsor was restructured under the newly established Under Secretary of Defense for Research and Engineering.

Contents

Figures and Tables

Summary

The Department of Defense (DoD) has experienced persistent challenges with software development across different kinds of acquisition programs. These challenges, which include schedule delays and cost overruns, can be attributed, in part, to an overreliance on outdated software development practices and methodologies—a problem highlighted by both the Defense Science Board (DSB) and Defense Innovation Board (DIB) in 2018 reviews of software acquisition practices. Given these findings and the importance of software development in defense acquisition timelines and spending, the capability of the defense acquisition workforce to understand and address software needs for defense acquisition programs is critical. However, DoD faces significant challenges in ensuring the workforce has this capability.

One strategy for assessing this capability is to focus on workforce proficiencies in relevant competencies. *Competencies*, which consist of an "observable, measurable pattern of knowledge, skills, abilities, behaviors, and other characteristics (KSAOs) needed to perform work roles or occupational functions successfully,"[2] can support a wide range of talent management initiatives including recruitment and selection, training and development, career development, and proficiency gap assessments. A collection of competencies for a specific career field or functional area is generally referred to as a *competency model*.

[2] The Defense Civilian Personnel Advisory Service (DCPAS) adapted this definition from DoD Instruction 1400.25, "DoD Civilian Personnel Management System: Volume 250, Civilian Strategic Human Capital Planning (SHCP)," Washington, D.C.: Under Secretary of Defense for Personnel and Readiness, November 18, 2008, p. 8.

Competency models have been developed to support workforce management of DoD across acquisition career fields. These competency models are also updated on a regular basis. Specifically, functional leaders (FL) oversee Functional Integrated Product Teams (FIPTs), which review competencies and develop plans to addresses potential training gaps. Despite these existing processes, there is neither a dedicated FL for software nor a corresponding software career field in DoD.[3] Consequently, efforts to update software competencies are limited to processes used by FLs and FIPTs representing existing acquisition career fields. A lack of coordination across career fields to determine how software competencies are defined or managed may contribute, in part, to confusion and ineffective management of software functions. Considering these challenges, DoD asked RAND to help improve the ability of DoD's software acquisition workforce to rapidly and reliably deliver complex software-dependent capabilities through an enhanced understanding of necessary technical competencies and improvements to education and training and workforce management and assessment.

This report addresses three major objectives to support DoD goals to improve software acquisition. First, we discuss the development of a competency model that emphasizes modern software practices and technical competencies. Second, we review training and education courses offered by the Defense Acquisition University (DAU) to identify potential gaps in the current training of software acquisition professionals. Third, we present several options for tracking and managing a software acquisition workforce.

Because software is not an official career field, it is important to define a software acquisition professional. In developing a working definition for this study, we revised the DAU definition to arrive at the following:

> *Software acquisition personnel* are military, civilian, and contractor personnel engaged in the definition, development, deploy-

[3] However, as noted by the Executive Secretary of the IT FIPT, the IT career field was intended to cover Software Acquisition Management. In fact, the IT certification course track was initially named Software Acquisition Management.

ment, operation, and sustainment of software components and software reliant systems or ecosystems.

Although some of these software functions may be performed outside of the acquisition community, acquisition professionals may need to be at least familiar with these functions to effectively acquire software. It is also important to note that this definition is broader than the scope of the current study in that it includes software acquisition personnel coded in any career field, while our study is limited to those in three primary acquisition career fields, selected in coordination with the sponsor: (1) information technology (IT), (2) engineering (ENG), and (3) program management (PM).

A Software Acquisition Competency Model

But building competency models can be a complex undertaking. When used to make employment decisions, DoD and the federal government must ensure that competencies are legally defensible. Thus, the competency modeling approaches followed by DoD, the Office of Personnel Management (OPM), and this project are based on basic principles that adhere to widely accepted professional and scientific guidelines.

RAND's approach to developing its software acquisition workforce competency model included the following steps:

- *Review existing competency models* used by DoD and commercial industry. The models differed greatly in the naming structures used and different levels of specificity. But sufficient similarities helped us identify relevant competencies for the software acquisition workforce. For example, competencies such as data management, software development, and sustainment appeared in multiple models and were included in our initial competency model. This was a valuable early step in determining which competencies were most likely to be relevant and critical going forward in the development process.
- *Review commercial industry trends* and modern software practices. A review of relevant literature and discussions with subject matter

experts (SMEs) provided insights used in the model's development. Three themes emerged. First, strong technical skills were considered important factors in fulfilling software engineering roles. Second, the ability to work collaboratively and the ability to participate in rapid prototyping emerged as two very important meta-skills in the domain of software engineering. Third, previous experience was among the strongest indicators of a candidate's potential success in a software engineering role.
- *Draft initial competency model.* Guided by the review of models and modern software practices in commercial industry, RAND researchers with an accumulated 70 years of experience in software management, development, and acquisition identified 13 initial competencies.
- *Gather stakeholder feedback and revise competencies.* Following discussions with sponsor office SMEs, the initial RAND model was substantially revised and expanded to increase the depth and specificity of competencies. This revised model was then updated in several iterations based on inputs gathered from DoD SMEs in panel workshops. A key component of this phase was to ensure that the titles and definitions were meaningful, facilitated a shared understanding, and minimized ambiguity.
- *Review options for further validation.* This validation phase is used to determine which competencies are needed across the workforce and which are specific to particular subgroups (e.g., software subspecialties). Because the workforce has not yet been defined, validating the competency model at this time is not possible and therefore was not completed as part of this study. Once software acquisition professionals have been identified, however, DoD can administer a competency assessment to determine the relative importance of each competency.

The conclusion of the feedback efforts resulted in a final set of 48 competencies; detailed descriptions and additional context and related definitions are presented in Appendix F. Table S.1 contains the titles in the revised model. This model should not be considered final until it has been validated, which, as noted, is a task that DoD will need to complete.

Table S.1
Final RAND Draft Software Acquisition Competencies and Topics

Problem Identification	1. Capabilities elicitation
	2. Business case development
Solution Identification	3. Strategic risk/reward analysis
	4. Cloud computing
	5. Software ecosystems
	6. Model-based engineering
Development Planning	7. Development tempo
	8. Release planning
	9. Software development planning
	10. Planning for continuous delivery
	11. Planning for continuous deployment
	12. System engineering planning
	13. Software metrics
	14. Configuration and version control
Transition and Sustainment Planning	15. Software documentation
	16. Contracting for software development
	17. Data and proprietary rights management
System Architecture Design	18. Architectural design approach
	19. Software orchestration and choreography patterns
	20. Software deployment patterns
	21. Artificial intelligence and machine-learning applications
	22. Augmented and virtual reality applications
	23. Embedded systems
	24. Balancing quality attributes
	25. Emerging technologies
Modeling Functional Capabilities and Quality Attributes	26. Use/abuse case modeling
	27. Validation of performance requirements
	28. Validation of sustainability requirements
	29. High fidelity system modeling
Building Secure, Safe and High-Availability Systems	30. Software assurance
	31. Cybersecurity
	32. Safety critical systems
	33. High-availability systems
Software Construction Management	34. Life-cycle management
	35. Detailed backlog management
	36. Release management
	37. Change management
	38. Automated test and continuous integration
Software Program Management	39. Effort estimation
	40. Product roadmap and schedule management
	41. Cost management
	42. Legal policy and regulatory environment management
	43. Risk, issues, and opportunity management
Mission Assurance	44. Quality assurance
	45. Root cause, corrective action
	46. System integration and testing
Professional Competencies	47. Strategic planning and change management
	48. Innovation and entrepreneurship

NOTE: The hierarchical structure of topics to competencies is not fixed and can be reorganized to meet a variety of organizational objectives.

A Review of Software Training and Education

In addition to determining which competencies are needed for a software acquisition career, it is important to determine what training and education resources exist and may still be needed for developing these competencies. To achieve this, RAND reviewed 394 courses related to software offered by DAU along with other DoD and civilian institutions and found potential differences in the curriculum offered. We found that coursework provided by DAU emphasizes management and DoD-specific acquisition requirements. This finding reflects points raised by DAU SMEs that DAU is best positioned to provide training and education on DoD-specific issues related to acquisition, not to train personnel how to code. In contrast, courses offered by civilian institutions, in general, focused more on design development and specification, programming, and software engineering.

The research team mapped DAU's courses in the Information Systems Acquisition (ISA) curriculum to the final draft set of competencies identified in the RAND-developed software acquisition competency model. We found that most competencies were covered to varying degrees but that 14 potentially had either minimal or no representation.

Several options exist to address potential gaps in the training and education of software acquisition professionals. These options may include developing new courses or updating course material, leveraging other DoD institutions and courses, and expanding partnerships with commercial education providers (e.g., universities) and massive open online courses (MOOCs) such as Coursera and edX. Formal courses are most effective when training needs to be provided to a large group of people and the concepts are transferable across services, organizations, and programs. Informal and on-the-job training can also be effective when training needs are more localized to specific programs or a more limited number of personnel.

However, DoD should first determine the relative importance of each competency and identify competency gaps prior to investing further in training and education resources to address the potential gaps identified. Conducting a competency gaps assessment helps to determine if software acquisition professionals have already obtained the necessary KSAOs prior to joining DoD or if additional training and education are needed. Once gaps have been confirmed, DoD can decide among the different training and education options. The best option(s) will depend, in part, on the extent and pervasiveness of the gaps identified across DoD.

Identifying, Tracking, and Managing a Software Acquisition Workforce

Currently, there is no established system for identifying or tracking who performs software functions in DoD. That is, there is no accepted government job title or occupational series for software professionals. Until the software acquisition workforce is identified, it is not possible to take advantage of the competency model or the insights gained from this study on potential gaps in training provided by DAU.

Thus, the research team explored systems currently used by DoD and the federal government to track personnel and identified options for DoD to track and manage a software acquisition workforce. Each option has pluses and minuses and requires different levels of effort and resources. In some cases, for example, options can only provide a snapshot of the workforce suitable for short-term solutions, whereas other options may require significant long-term planning, coordination, and approval from external agencies. The six options are

- *Perform a data call*. This is the most direct strategy to identify personnel who perform software functions throughout DoD. It provides a snapshot of software workforce positions and major duties. This is a short-term solution and may require incentives to encourage response to the data call. It may also require DoD to take steps to verify the accuracy of self-identifications.
- *Flag positions/billets*. This provides information to count the number and type of software acquisition positions. However, positions may not provide accurate information about individual qualifications or proficiencies. Waivers may also need to be

tracked to determine positions that are filled by unqualified software professionals. Position duties may not reflect actual work performed.

- *Assign codes to identify skills, experiences, and education*. This provides information about workforce readiness and capabilities and can be useful in uncovering potential skill gaps. Verifying skills and quality of experiences is a resource-intensive effort.
- *Create unofficial job titles*. Tailored job titles for acquisition professionals facilitate tracking and recruiting. But this option creates a limited structure for workforce planning and management (such as compensation and training).
- *Define new acquisition career field(s)*. This provides software-specific career field(s) for acquisition-coded positions. It reinforces communications about the importance of software and provides a focal point for talent management efforts. DoD leadership can determine the level of training and education required for those assigned in these career fields. But such an approach would exclude software professionals outside the officially designated acquisition workforce, who may play other important roles such as software sustainment.
- *Develop a new occupational series*. This entails a government-wide implementation of a software workforce strategic plan. It requires a long-term commitment and considerable data.

Recommendations

There are two fundamental recommendations that DoD should immediately follow to address potential concerns with the software acquisition workforce.

1. **Identify who is in the software acquisition workforce.** Without an understanding of who is in the software acquisition workforce, DoD can neither validate the competency model nor make use of it to identify competency gaps in the workforce. A number of options are available for tracking and managing

a software acquisition workforce, each of which requires different levels of resources and offers different outcomes. Taking these considerations into account, *we recommend conducting a data call as the first step to identify personnel who perform software functions. Further we recommend limiting the data call initially to personnel within the acquisition community to promote rapid implementation of competencies across acquisition career fields.* Other options either require greater amount of resources and coordination or do not directly address who performs software functions. To the extent that software professionals are distributed across multiple career fields, an effective data call will require considerable coordination and support from DoD leaders (e.g., FLs, Service Directors of Acquisition Career Management, PMs). DoD could use the data call results to determine the need to refine estimates using other strategies or expand tracking efforts beyond the acquisition community. Finally, the data call results should be used to guide discussions on the need and level of effort required for more formal tracking mechanisms (e.g., do the data indicate a need to develop a software subspecialty or career field?).

2. **Validate the software acquisition competencies.** After the software acquisition workforce has been identified, the competencies should be validated. At a minimum, DoD needs to collect information from the workforce to evaluate the relative importance of each competency. This step will require coordination with the Defense Civilian Personnel Advisory Service (DCPAS) to determine the most appropriate way forward given limitations with existing competency management tools used by DoD. As with identifying the workforce, validation can be approached in a number of ways.
   a. *We recommend either reprogramming the existing competency management tools used by DoD or selecting another software tool. Most importantly, we recommend limiting the number of questions to focus on the relative importance of each competency.* Doing so will help to minimize survey fatigue. Future analyses, including assessments of proficiency, could then

focus on a more limited set of the most critical competencies relevant to the target software professional. If existing tools cannot be reprogrammed, the competencies could be administered in smaller chunks using appropriate sampling techniques so that each respondent "sees" only a small number of competencies.

b. *We further recommend consulting a statistician to ensure that the sample of respondents are representative of important perspectives* (e.g., service branch, years of experience, acquisition category). A well-designed sampling plan is needed so that appropriate statistical analyses can be conducted to address critical questions about the workforce.
c. *Finally, we recommend planning future validation studies that establish links between performance on competencies and outcome measures*. These types of criterion-related validation studies require considerable planning to develop and collect the appropriate performance measures, and they should therefore be integrated into a long-term strategy for evaluating and managing the software acquisition workforce.

Conclusion

The work described in this report should be viewed as first steps in a long-term strategy to define and manage a software acquisition workforce. Further analysis is required to validate the competencies and to determine who is performing software functions. To gain complete traction on this problem, DoD needs to appoint a senior leader who can implement these recommendations across the services. Without a champion, any improvements will be slow and sporadic.

Acknowledgments

We gratefully acknowledge the support of our project director, Sean Brady, for providing the vision and overall direction for this project. We also thank Jerry Tarasek and Tom Hickok, who served as our project liaisons to establish connections with the Functional Integrated Product Teams, coordinate meetings, and provide feedback on project tasks.

We thank all of the software subject matter experts who reviewed and provided feedback on the software competencies and the many individuals from the Information Technology, Engineering, and Program Management FIPTs who helped to connect us to SMEs.

We appreciate the support of Dave Pearson for facilitating the review of DAU curriculum. We thank the course managers for taking the time to review and map the competencies to the courses they manage.

We would like to acknowledge Graham Andrews for his software expertise and assistance in evaluating educational courses. We thank Clara Aranibar for helping to coordinate and schedule panel workshops. We are grateful to Barbara Bicksler for her support in organizing, editing, and summarizing project activities and key findings. Finally, we thank our reviewers, Bill Shelton, Brad Wilson, and James Belanich, for their feedback and recommendations, which helped improve the overall quality of this report.

Abbreviations

ACAT	acquisition category
ACM	Association for Computing Machinery
AFIT	Air Force Institute of Technology
ASI	Additional Skill Identifier
AT&L	acquisition, technology, and logistics
ATM	automatic transaction machine
AWF	acquisition workforce
AWQI	Acquisition Workforce Qualification Initiative
BLS	Bureau of Labor Statistics
CMMI	Capability Maturity Model-Integrated
COTS	commercial off-the-shelf
DACM	Director of Acquisition Career Management
DAU	Defense Acquisition University
DAS	Defense Acquisition System
DCAT	Defense Competency Assessment Tool
DCPAS	Defense Civilian Personnel Advisory Service
DIB	Defense Innovation Board
DISA	Defense Information Systems Agency
DMDC	Defense Manpower Data Center
DoD	Department of Defense
DoDI	DoD Instruction

DSB	Defense Science Board
EA	enterprise architecture
EDP	electronic data processing
ENG	engineering
FE	facilities engineering
FIPT	Functional Integrated Product Team
FL	functional leader
GAO	Government Accountability Office
GOTS	government off-the-shelf
GS	General Schedule
HCI	Human Capital Initiative
IaaS	Infrastructure as a Service
IEEE	Institute of Electrical and Electronics Engineers
IoT	internet of things
IOT&E	initial operating test and evaluation
IPT	integrated product team
ISA	Information Systems Acquisition
IT	information technology
JCIDS	Joint Capabilities Integration and Development System
KPP	key performance parameter
KSAOs	knowledge, skills, abilities, and other characteristics
MOOC	massive open online course
NDU	National Defense University
NPS	Naval Postgraduate School
OPM	Office of Personnel Management
OTA	other transactional authority
PaaS	Platform as a Service
PCS	Position Classification Standards
PM	Program Management/Manager

PWA	progressive web app
QA	quality assurance
QE	quality engineering
RFP	request for proposal
RR	risk reduction
SaaS	Software as a Service
SATEWG	Software Acquisition Training and Education Working Group
SDLC	software development life cycle
SEE	skills, experiences, and education
SEI	Special Experience Identifier
SHRM	Society for Human Resource Management
SME	subject matter expert
SQI	Special Qualification Identifier
SwE	software engineering
SWEBOK	Software Engineering Body of Knowledge
SWECOM	Software Engineering Competency Model
USD (A&S)	Under Secretary of Defense for Acquisition and Sustainment

CHAPTER ONE
Introduction

The Department of Defense's (DoD's) multimillion-dollar acquisitions programs have been under scrutiny by Congress for schedule delays and cost overruns.[1] The Government Accountability Office (GAO) has included DoD's weapon systems acquisitions and DoD business systems modernization as high-risk areas for many years. In 2015, due in part to high-profile DoD failures such as Expeditionary Combat Support Systems, GAO added information technology (IT) acquisition and operations to the high-risk list for the federal government in 2017.[2]

In particular, DoD has experienced persistent challenges with software development across different kinds of acquisition programs. For example, GAO recently reported that the four major software-intensive space programs it reviewed "are estimated to cost billions of dollars, have experienced overruns of up to three times originally estimated cost, and have been in development for periods ranging from 5 to over 20 years."[3] These challenges can be attributed, in part, to an overreliance on outdated software development practices and methodologies.

In a recent review of software acquisition practices in DoD, the Defense Science Board (DSB) emphasized several differences between

[1] Congressional Research Service, "The Department of Defense Acquisition Workforce: Background, Analysis, and Questions for Congress," July 29, 2016.

[2] See U.S. Government Accountability Office (GAO), *High-Risk Series: Progress on Many High-Risk Areas, While Substantial Efforts Needed on Others*, GAO-17-317, February 2017.

[3] GAO, "DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs," GAO-19-136, March 18, 2019.

DoD and commercial industry. Notably, "Software development in the commercial world has undergone significant change in the last 15 years, while development of software for defense systems has continued to use techniques developed in the 1970s through the 1990s."[4] The Defense Innovation Board (DIB) shared a similar conclusion in opening its draft report of *Ten Commandments of Software*: "The latest industry best practices for developing, fielding, and sustaining software applications and information technology systems are substantially outpacing the US government's . . . methods."[5]

Given these findings and the importance of software development in defense acquisition timelines and spending, the capability of the defense acquisition workforce (AWF) to understand and address software needs for defense acquisition programs is critical. One strategy for assessing this capability is to assess workforce proficiencies in relevant competencies. *Competencies*, which consist of an "observable, measurable pattern of knowledge, skills, abilities, behaviors, and other characteristics (KSAOs) needed to perform work roles or occupational functions successfully,"[6] can support a wide range of talent management initiatives, including recruitment and selection, training and development, career development, and proficiency gap assessments. A collection of competencies for a specific career field or functional area is generally referred to as a *competency model*.

Competency models have been developed to support workforce management of DoD across acquisition career fields. These competency models are also updated on a regular basis. Specifically, functional leaders (FLs) oversee Functional Integrated Product Teams (FIPTs), which review competencies and develop plans to addresses potential training gaps. Despite these existing processes, there is neither a dedicated FL for software nor a corresponding software career

[4] Defense Science Board, *Design and Acquisition of Software for Defense Systems*, February 14, 2018, p. 1.

[5] Defense Innovation Board (DIB), *Ten Commandments of Software*, Version 0.14, last modified April 15, 2018, p. 2.

[6] DCPAS adapted this definition from DoDI 1400.25, 2008, p. 8.

field in DoD.[7] Consequently, efforts to update software competencies are limited to processes used by FLs and FIPTs representing existing acquisition career fields. A lack of coordination across career fields to determine how software competencies are defined or managed may contribute, in part, to confusion and ineffective management of software functions. Considering these challenges, DoD asked RAND to help improve the ability of DoD's software acquisition workforce to rapidly and reliably deliver complex software-dependent capabilities through an enhanced understanding of necessary technical competencies and improvements to education and training and workforce management and assessment.

Objectives

This report addresses three major objectives to support DoD goals to improve software acquisition. First, we discuss the development of a competency model that emphasizes modern software practices and technical competencies (i.e., combinations of KSAOs needed).[8] Second, we review training and education courses offered by the Defense Acquisition University (DAU) to identify potential gaps in the current training of software acquisition professionals. We also explore alternative training options, including other DoD institutions and select civilian programs offering software-related courses, to address any potential gaps. Third, we present several options for tracking and managing a software acquisition workforce, which are necessary for taking advantage of the competency model and the

[7] However, as noted by the Executive Secretary of the IT FIPT, the IT career field was intended to cover Software Acquisition Management. In fact, the IT certification course track was initially named Software Acquisition Management.

[8] Society for Human Resource Management (SHRM), *Content Validation Study of the SHRM Competency Model*, undated; Michael A. Campion, Alexis A. Fink, Brian J. Ruggeberg, Linda Carr, Geneva M. Phillips, and Ronald B. Odman, "Doing Competencies Well: Best Practices in Competency Modeling," *Personnel Psychology*, Vol. 64, No. 1, 2011, pp. 225–262; Richard S. Mansfield, "Building Competency Models: Approaches for HR Professionals," *Human Resource Management*, Vol. 35, 1996, pp. 7–18.

insights gained from this study on potential gaps in training provided by DAU. For each option, we review potential benefits and limitations and conclude with recommendations. Before turning to these topics in the following chapters, we begin with a discussion of the software acquisition workforce and recent changes in DoD's organizational structure, which have implications for management of the acquisition workforce.

Software Acquisition Workforce

Section 1 of the DoD *Defense Acquisition Workforce Program Guide* (2017) outlines the definition and process for designating acquisition workforce positions. DoD defines acquisition as "the conceptualization, initiation, design, development, test, contracting, production, deployment, logistics support (LS), modification, and disposal of weapons and other systems, supplies, or services (including construction) to satisfy DoD needs, intended for use in, or in support of, military missions."[9] If more than 50 percent of a position's duties and responsibilities fit within this definition, it is designated and coded as part of the *acquisition workforce*.

All acquisition positions are assigned to one of 15 acquisition career fields described in Appendix A, which cover acquisitions ranging from radios and large communication systems to major defense systems, ground vehicles, aircraft, and ships. All acquisition personnel must meet certification requirements that are established by the functional leaders of each career field. DAU maintains these certification requirements and manages the courses required for certification. Three certification levels are used to manage standards and qualifications: Level I—Basic or Entry Level, Level II—Intermediate or Journeyman Level, and Level III—Advanced or Senior Level.

Within this system, software professionals are not currently coded or tracked in any systematic way and therefore may reside in

[9] DoD, *Defense Acquisition Workforce Program Desk Guide*, Washington, D.C., July 20, 2017a, p. 1.

variety of acquisition career fields across DoD or even work in a non-acquisition position. The focus of this report, however, is limited to the acquisition workforce and uses inputs primarily from three primary career fields—(a) information technology (IT), (b) engineering (ENG), and (c) Program Management (PM)—which were selected in coordination with the sponsor to encourage subject matter expert (SME) participation from a range of software perspectives. Table 1.1 provides representative roles and activities for each of these career fields. Although the descriptions for each of the three career fields list activities relevant to software, there are several questions that cannot be answered at this time:

- Who performs software functions in the PM, IT, and ENG career fields?
  - Which software functions are performed by these professionals?
  - How relevant are the existing PM, IT, and ENG competencies for these individuals?
- Would a new career field improve software acquisition?
  - How many individuals perform software functions as core duties?
  - Are the existing career field competencies sufficient for hiring, training, and managing individuals who have core functions related to software?
- What training and education is needed?
  - Is the current training provided by DAU sufficient?
  - What software competency gaps exist in the workforce?
  - What are the most effective ways to address potential gaps?

This report does not intend to directly answer these questions, but rather provides an initial and fundamental step by detailing software competencies that may be needed by acquisition professionals. These competencies will need to be further validated in future work (as described in Chapters Seven and Eight) to determine if they are truly required and, if so, to determine the required level of proficiency, and to identify proficiency gaps in the workforce. This validation step is critical to determine the most effective way forward for DoD.

Table 1.1
Acquisition Career Fields in Current Study

Acquisition Career Field	Representative Roles and Activities
Engineering	*Functional Engineer* • Plans, organizes, conducts, and/or monitors engineering activities relating to the design, development, fabrication, installation, modification, sustainment, and/or analysis of systems or systems components for a functional specialty (i.e., reliability and maintainability, systems safety, materials, avionics, structures, propulsion, chemical/biological, human systems interfaces). • Demonstrates how systems engineering technical processes and technical management processes guide engineering activities for a functional specialty.
	General Engineer • Plans, organizes, conducts, and/or monitors engineering design, development, and sustainment activities for systems or systems components. • Demonstrates how systems engineering technical processes and technical management processes guide design, development, and sustainment activities.
Information Technology	*Central Design Activity* • Identifies and describes: basic concepts of software engineering and development activities; enterprise architecture; best practices; IT systems engineering; information assurance/cybersecurity; IT-related technologies; test and evaluation processes; and verification and validation processes.
	Project Office/Field Activities • Identifies and describes: IT program management approaches; emerging IT acquisition strategies; best practices; IT-related performance measures and quality management; acquisition planning, solicitation, and administration; information assurance/cybersecurity; test and evaluation processes; verification and validation processes; and fielding and sustaining IT systems.
Program Management	*Weapon Systems* • Participates in an integrated product team (IPT) delivering a weapon system, command and control/network-centric system, or space system. • Performs financial and status reporting and basic logistic activities. • Supports pre-award contract activities and workload planning and scheduling.
	Services • Assists in acquisition planning, assessing risk (technical, cost and schedule), and contract tracking and performance evaluation.
	Business Management Systems/IT • Participates in a business process IPT, fundamentals of enterprise integration, and outcome-based performance measures.

SOURCE: DAU, “Career Fields,” webpage, undated.

Given that software acquisition functions or personnel are not tracked in a systematic way, how does one define a software acquisition professional? We initially adopted DAU's definition of software acquisition. However, through discussions and feedback from DoD SMEs, we revised that definition to arrive at the following working definition:

> *Software acquisition personnel* are military, civilian and contractor personnel engaged in the definition, development, deployment, operation, and sustainment of software components and software reliant systems or ecosystems.[10]

Although this definition is broad, it is designed to differentiate software acquisition personnel from other acquisition professionals such as IT purchasing agents. IT purchasing agents only purchase software (e.g., desktop software); they do not define, develop, deploy, operate, or sustain the software. It is also important to note that some software functions may be performed primarily outside of the acquisition community. Nonetheless, acquisition professionals may need to be at least familiar with these functions to effectively acquire software.

[10] *Development* refers to the processes, procedures, people, material, algorithms, and information required to conceive, specify, design, program, document, test, deliver, and deploy the software aspects of a system. This includes oversight activities required to determine adherence and compliance to contract requirements. *Sustainment* refers to the processes, procedures, people, material, algorithms, and information required to support, maintain, and operate the software aspects of a system. This includes software development, documentation, operations, deployment, security, configuration management, training (users and sustainment personnel), help desk, commercial off-the shelf (COTS) product and license management, and technology refresh. *Software* refers to a collection of data and instructions that executes on a processing unit. Software includes, but is not limited to, applications, scripts, databases, operating systems, device drivers, and firmware. *Software reliant systems* refers to a hardware-software system (such as a radar) that would fail to meet its mission use if the software were to fail, or a software system (such as mission planning or intelligence dissemination tools) with its accompanying computing infrastructure and network. *Ecosystems* refers to a set of entities functioning as a unit and interacting with a shared end-user constituency for software and services, together with relationships among them. Ecosystems form when a set of core components (the keystone) are complemented by peripheral components (e.g., apps or services) developed by autonomous entities (i.e., organizationally independent of the core developer) to address specific user needs. Ecosystems are characterized by interoperability and co-innovation enabled thru common interfaces and shared knowledge.

The relative importance and proficiency level required by acquisition professionals across these functions can be determined only once the software workforce has been defined. It is also important to note that this working definition is broader than the scope of the current study in that it includes software acquisition personnel coded in any career field. However, we did not explicitly include feedback or inputs from career fields outside PM, ENG, and IT. Therefore, the competencies presented in this study should be systematically reviewed by relevant SMEs prior to adoption by other acquisition and nonacquisition career fields.

Evolving Organizational Structure and Guidance for DoD Acquisition

In response to Section 901 of National Defense Authorization Act for Fiscal Year 2017, DoD restructured the roles of the Under Secretary of Defense for Acquisition, Technology, and Logistics to "better pursue the goals of technological superiority, affordable systems, and well managed business operations."[11] As of February 1, 2018, the new organizational structure includes two new roles:

- Under Secretary of Defense for Research and Engineering to "drive innovation and accelerate the advancement of our warfighting capability"
- Under Secretary of Defense for Acquisition and Sustainment (USD [A&S]) "to deliver proven technology into the hands of the Warfighter more quickly and affordably."[12]

USD (A&S) has oversight of the acquisition workforce and the responsibilities for establishing accession, education, training, and experience requirements. Furthermore, under the authority of USD (A&S),

[11] DoD, *Report to Congress: Restructuring the Department of Defense Acquisition, Technology and Logistics Organization and Chief Management Officer Organization*, August 1, 2017b, p. 3.

[12] DoD, 2017b, p. 3.

Human Capital Initiatives (HCI) maintains responsibility for position category descriptions and job specialty descriptions and for providing metrics on the acquisition workforce and guidance on the development of career models for education, training, and experience needed for career progression.[13]

In addition to the reorganization, under Subtitle I, Development and Acquisition of Software-Intensive and Digital Products and Services, of the 2018 National Defense Authorization Act, Congress directed several efforts to improve software acquisition. In particular, Section 872 directs DIB to conduct a study with the aim of addressing several objectives, among which are "produce specific and detailed recommendations for any legislation, including the amendment or repeal of regulations, as well as non-legislative approaches, that the members of the Board conducting the study determine necessary to—:

- Streamline development and procurement of software;
- Adopt or adapt best practices from the private sector applicable to Government use;
- Promote rapid adoption of new technology."[14]

Sections 873 and 874 each provide specific direction to the services to consider and pilot Agile methods for select major programs, defense business systems, and software development activities. For example, Section 873 directs the services to select major programs that will be part of a pilot to use Agile methods. These programs include major software-intensive warfighting systems that have "identified software development as a high risk," "experienced cost growth and schedule delay," and "did not deliver any operational capability within the prior calendar year." They also include defense business systems that "have experienced cost growth and schedule delay," "did not deliver any operational capability within the prior calendar year," and

[13] DoDI 5000.66, *Defense Acquisition Workforce, Education, Training, Experience, and Career Development Program*, Washington, D.C.: U.S. Department of Defense, August 31, 2018.

[14] Public Law 115-91, National Defense Authorization Act for Fiscal Year 2018, December 12, 2017.

"are underperforming other systems within a defense business system portfolio with similar user requirements."[15]

Organization of This Report

Using our working definition of the acquisition workforce, the remainder of this report addresses the three study objectives. Chapters Two through Five explore the development of a competency model for software acquisition professionals, our first objective. Specifically, Chapter Two presents our methodology and roadmap for developing and validating competencies. Chapters Three and Four provide an overview of relevant data sources used to develop the initial competency model, with Chapter Three highlighting other related DoD and commercial industry competency models and Chapter Four summarizing relevant findings from our evaluation of commercial industry trends. Chapter Five describes our initial competency model, the steps taken to gather and integrate stakeholder feedback to revise the competencies, and the final competency model that resulted from this process. Chapter Six turns to the topic of training and education and our second objective of mapping competencies to software courses provided by DAU and other DoD and civilian education institutions and identifying gaps and options for addressing them. Chapter Seven, which fulfills our third objective, presents options for identifying and tracking a software acquisition workforce. Conclusions and recommendations are presented in Chapter Eight.

[15] Public Law 115-91, Section 873, p. 218.

CHAPTER TWO

Methodology for Developing Competencies

Competencies can be characterized by KSAOs that are important to organizational success and generally reflect the technical knowledge or behaviors required to perform a job or role within an organization.[1] Competencies can be useful tools to support a wide range of talent management initiatives, including recruitment and selection, training and development, career development, and proficiency gap assessments.

Competencies are used government wide and across the commercial industry. However, not all have been developed following best practice guidelines,[2] in part, because there is no single accepted standard definition or process for developing and implementing competencies. As a result, competency models, which organize competencies for a specific occupation or function, have many different flavors and range in their level of specificity. In a recent review of competency models used by different organizations, the Institute for Defense Analyses found that "some taxonomies focused on detailed tasks and specific actions that a job incumbent needs to perform, while others emphasized broader personal characteristics, attitudes, and traits."[3]

[1] SHRM, undated; Campion et al., 2011; J. S. Shippmann, R. A. Ash, M. Battista, L. Carr, L. D. Eyde, B. Hesketh, J. Kehoe, K. Pearlman, E. P. Prien, and J. I. Sanchez, "The Practice of Competency Modeling," *Personnel Psychology*, Vol. 53, 2000, pp. 703–740.

[2] Campion et al., 2011.

[3] J. Belanich, F. L. Moses, and P. Lall, *Review and Assessment of Personnel Competencies and Job Description Models and Methods*, Alexandria, Va.: Institute for Defense Analyses, 2016, p. 11.

Competencies and competency models clearly differ in their levels of specificity, with some being heavily detailed to apply to specific positions or tasks and others being broader to satisfy needs across the entire organization.[4]

Two competencies from OPM's Cybersecurity Competency Model demonstrate how competency models often capture general as well as more technical KSAs:

> **Organizational Awareness**—Knows the organization's mission and functions, and how its social, political, and technological systems work and operates effectively within them; this includes the programs, policies, procedures, rules, and regulations of the organization.
>
> **Distributed Systems**—Knowledge of the principles, theoretical concepts, and tools underlying distributed computing systems, including their associated components and communication standards.

When competencies are used to make employment decisions, DoD and the federal government must ensure that those competencies are legally defensible. Thus, the competency modeling approaches followed by DoD, OPM, and in this project are based on principles that adhere to widely accepted professional and scientific guidelines.[5] This chapter discusses some of the complexity surrounding competency models, but also provides a roadmap describing our approach to developing competencies.

[4] Belanich et al., 2016.

[5] Society for Industrial and Organizational Psychology, "Principles for the Validation and Use of Personnel Selection Procedures," *Industrial and Organizational Psychology,* Vol. 11, No. S1, 2018, pp. 1–97; Equal Employment Opportunity Commission, Civil Service Commission, Department of Labor, and Department of Justice, *Uniform Guidelines on Employee Selection Procedures*, August 25, 1978.

Department of Defense Competency Definition and Process

To guide our competency development efforts, we adopted the definition used by the Defense Civilian Personnel Advisory Service (DCPAS):

> An observable, measurable pattern of knowledge, skills, abilities, behaviors, and other characteristics (KSAOs) needed to perform work roles or occupational functions successfully.[6]

Competencies can serve different tiers in the DoD-wide Five-Tiered Competency Framework, illustrated in Figure 2.1. Tier 1 focuses on core competencies that apply across DoD and are not specific to a position or agency. An example Tier 1 competency could be "demonstrates integrity." Tier 2 competencies apply across an occupational series—for example, "cybersecurity" could be applied across all of IT. Tier 3 competencies focus on KSAOs specific to subspecialties that may exist in one or more occupational series—for example, "software assurance" could be considered a suboccupational competency for a specialty within IT. Tier 4 adds further specificity to components and agencies—for example, competencies required to work at the Air Force Sustainment Center. Finally, Tier 5 competencies are meant to capture any additional KSAOs needed for a specific position that are not already addressed by Tiers 1 to 4.

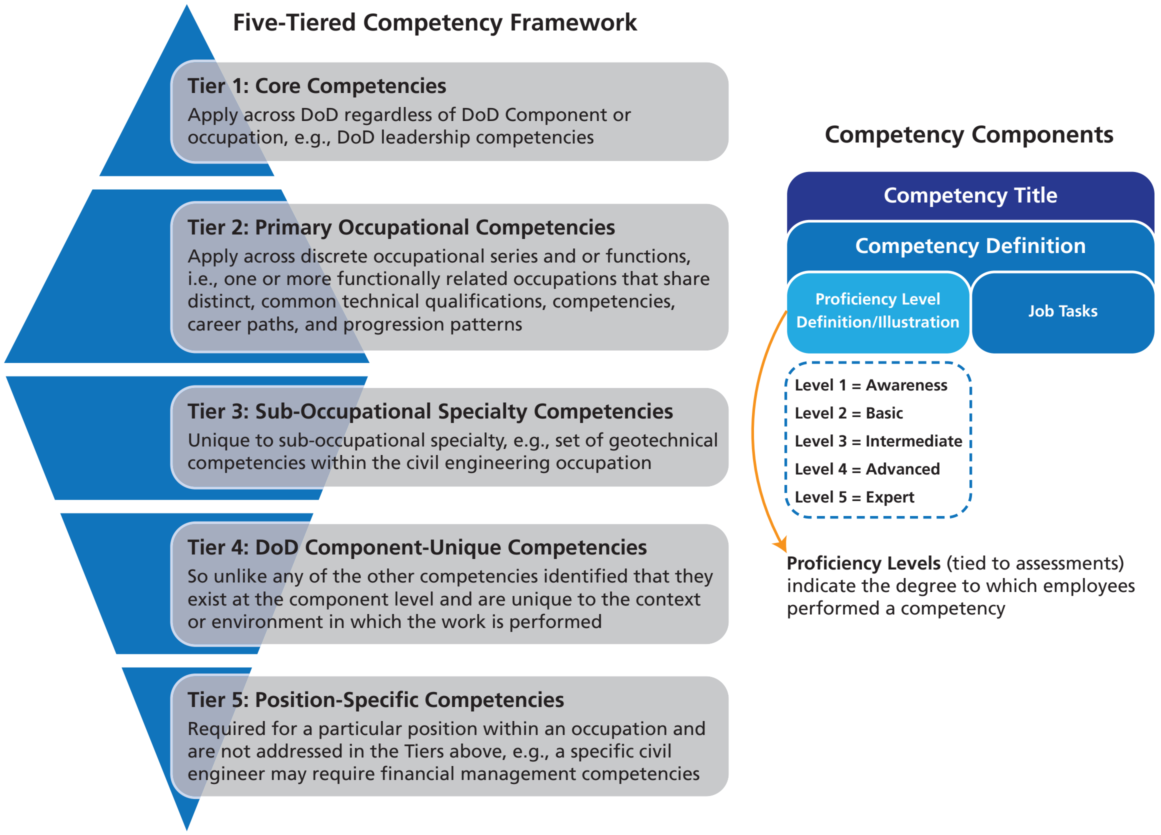
Following this framework and consistent with the DCPAS competency definition, our software acquisition competency model addresses Tiers 2 and 3 using KSAO-based competencies supported by sets of example behaviors and job tasks.

Competency Modeling Approaches

As indicated previously, there is no single approach to developing competency models. Nonetheless, many organizations including DCPAS follow a few basic steps, which include a thorough review of existing

[6] DCPAS adapted this definition from DoDI 1400.25, 2008, p. 8.

Figure 2.1
Five-Tiered Competency Framework



Five-Tiered Competency Framework

Tier 1: Core Competencies
Apply across DoD regardless of DoD Component or occupation, e.g., DoD leadership competencies

Tier 2: Primary Occupational Competencies
Apply across discrete occupational series and or functions, i.e., one or more functionally related occupations that share distinct, common technical qualifications, competencies, career paths, and progression patterns

Tier 3: Sub-Occupational Specialty Competencies
Unique to sub-occupational specialty, e.g., set of geotechnical competencies within the civil engineering occupation

Tier 4: DoD Component-Unique Competencies
So unlike any of the other competencies identified that they exist at the component level and are unique to the context or environment in which the work is performed

Tier 5: Position-Specific Competencies
Required for a particular position within an occupation and are not addressed in the Tiers above, e.g., a specific civil engineer may require financial management competencies

Competency Components

Proficiency Levels (tied to assessments) indicate the degree to which employees performed a competency data, drafting an initial model, gathering inputs from SMEs, refining the model, and validation. We first outline DCPAS's approach to competency modeling, which we followed to the extent possible. We then outline where our approach aligns with theirs and the challenges with following their approach for this study, particularly in terms of the final validation step.

Defense Civilian Personnel Advisory Service Approach

DCPAS follows three general phases to competency development: (1) Competency Development, (2) Competency Assessment and Validation, and (3) Implementation. Phase 1, Competency Development, involves reviewing data to develop a draft list of technical competencies, followed by gathering SME feedback, which is collected in two stages. The first stage of feedback involves a pre-panel activity, which is a self-administered assessment (i.e., survey) to evaluate the relative importance of competencies. The results of the pre-panel activity are used to screen out irrelevant competencies and to focus subsequent steps on the most important competencies. The second stage occurs in SME panels, which are face-to-face discussions with SMEs about the competencies.

Once revisions to competencies have been completed, DCPAS begins Phase 2, Competency Assessment and Validation, which involves administering and analyzing results from the Defense Competency Assessment Tool (DCAT). Policy guidance (DoDI 1400.25) provides instruction for civilian personnel management including the use of DCAT to "assess and report workforce competency gaps and proficiency levels."[7] DCAT provides a platform linked to the Defense Civilian Personnel Data System to evaluate competencies through DoD occupational assessment surveys. Once DCAT has been completed, SMEs provide one final round of reviews of the competencies, definitions, and tasks. SMEs also review the results of DCAT and recommend any necessary changes to ensure the model is accurate and ready to be implemented. The final phase in the DCPAS process is implementation, which involves submitting an executive summary

[7] DoDI 1400.25, 2008, p. 16.

and validated model to the Office of the Secretary of Defense functional community managers.

Limitations of the Defense Competency Assessment Tool

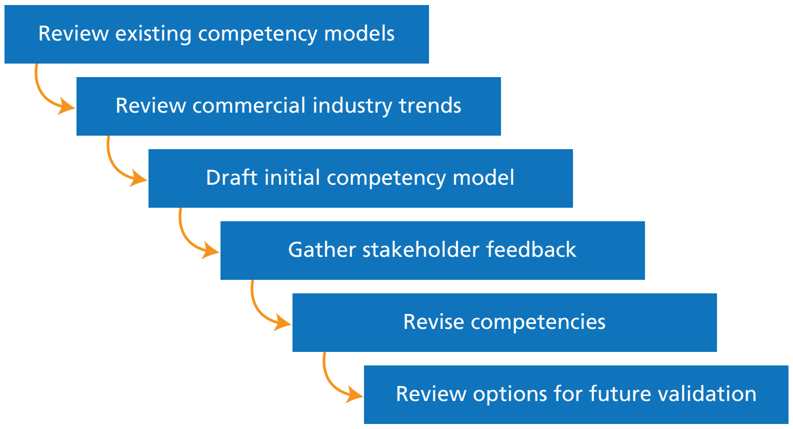
Although the questions in the DCAT are designed to gather critical information needed to support the validation of competencies, the software tool is limited in several ways. Most important is that the number of competencies that can be evaluated at any one time is no more than 12. According to discussions with DCPAS, using more than 12 competencies can cause the software to function slowly and may significantly reduce response rates. The current design of DCAT requires employees and their supervisors to complete assessment questions. Responses are counted only when a match is made (i.e., supervisor and employee both complete). This design may not be problematic when there are 12 competencies or fewer or when the supervisor has only a few subordinates completing the assessment. As the number of competencies or subordinates increase, the number of overall questions that a supervisor must complete also increases, thereby quickly increasing survey fatigue and risk of nonresponse.

Another important factor to consider is that DCAT is designed to be used primarily with the DoD civilian workforce. Although DCAT could also be administered to military personnel, additional steps would need to be taken to merge results with military databases to evaluate competencies by subgroups (e.g., occupational specialty, rank).

RAND Competency Approach

We follow the general process DCPAS uses to develop a software acquisition competency model. As illustrated in Figure 2.2, we started with a thorough review of relevant competency models used by DoD and commercial industry (Chapter Three). Next, we explored commercial industry trends and modern software practices (Chapter Four). This step involved several efforts: a review of literature, interviews with a small number of commercial industry SMEs, analysis of commercial industry position announcements, and a review of top university software course descriptions and objectives. Guided by the review of existing models and modern software practices used by commercial industry, we drafted an initial competency model comprised of 13 competencies.

Figure 2.2
RAND Approach to Competency Model Development


Review existing competency models
Review commercial industry trends
Draft initial competency model
Gather stakeholder feedback
Revise competencies
Review options for future validation

Following discussions with sponsor-office SMEs with expertise in software development, engineering, and training, the initial RAND model was substantially revised and expanded to increase the depth and specificity of competencies. The expanded competencies were designed to increase understanding and minimize misinterpretation of content and definitions. This revised model was then updated in several iterations based on inputs gathered from DoD SMEs in panel workshops. A key component of this phase was to ensure that the titles and definitions are meaningful, facilitate a shared understanding, and minimize ambiguity. The initial competency model, stakeholder feedback, and the revised model are discussed in more detail in Chapter Five.

The competencies should then be validated by collecting information about the relative importance of each competency. This validation phase is used to determine which competencies are needed across the workforce and which are specific to particular subgroups (e.g., software subspecialties).

Because the workforce has not yet been defined (as discussed further in Chapter Seven), validating the competency model at this time is not possible and therefore was not completed as part of this study. Once software acquisition professionals have been identified, however, DoD can administer a competency assessment to determine the relative importance of each competency. Specifically, data should be gathered from the software acquisition workforce (e.g., survey of supervisors and job incumbents) to confirm which competencies are needed and by whom. A more complete discussion of validation options is presented in Chapter Eight.

CHAPTER THREE

Review of Existing Competency Models

In this chapter, we describe our review of existing competency models that contain relevant software competencies. This review provides a foundation on which to draft an initial set of competencies relevant to the software acquisition workforce. To initiate our search for relevant software acquisition competencies, we considered competency models currently used by DoD and commercial industry. DoD models included career field functional competencies, which overlap with efforts to provide additional detail by the Acquisition Workforce Qualification Initiative (AWQI) for the PM, IT, and ENG career fields. These also include commercial models, such as the Software Engineering Competency Model (SWECOM) and version 3 of the *Guide to the Software Engineering Body of Knowledge* (SWEBOK Guide). Finally, we reviewed additional DoD efforts related to software acquisition competencies, particularly the Software Acquisitions Training and Education Working Group (SATEWG).

Department of Defense Models

DoD's career field functional models come from the December 2005 DoDI 5000.66, "Operation of the Defense Acquisition, Technology, and Logistics Workforce Education, Training, and Career Development Program," which

> implements reference (a) and provides uniform guidance for managing positions and career development of the Acquisition,

> Technology, and Logistics (AT&L) Workforce. This includes the designation and identification of AT&L positions; specification of position requirements; attainment and maintenance of AT&L competencies through education, training and experience; AT&L Performance Learning Model; management of the Defense Acquisition Corps; selection and placement of personnel in AT&L positions; and workforce metrics.[1]

Reissued in July 2017 as "Defense Acquisition Workforce Education, Training, Experience, and Career Development Program," DoDI 5000.66 similarly "establishes policies, assigns responsibilities, and provides procedures for the conduct of the Defense Acquisition Workforce (AWF) Education, Training, Experience, and Career Development Program, referred to in this issuance as the 'AWF Program.'"[2]

These instructions are part of a DoD effort to define requirements for the acquisition workforce and resulted in FIPT-reviewed competencies for the PM, ENG, and IT career fields. Each list is divided into high-level competency units, such as business management (PM), technical management (ENG), or acquisition planning (IT), with a varying number of competencies and accompanying descriptions, in each unit. There is some overlap in competencies across the different career fields. The PM career field functional competencies are also described in terms of expectations for a basic level, intermediate level, and advanced level of competence.

To ensure the acquisition workforce has the appropriate competencies to meet job requirements, USD AT&L launched AWQI as an element of the Better Buying Power 2.0 initiatives in 2012.[3] This

[1] DoDI 5000.66, "Operation of the Defense Acquisition, Technology, and Logistics Workforce Education, Training, and Career Development Program," Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, December 21, 2005, p. 1.

[2] DoDI 5000.66, "Defense Acquisition Workforce Education, Training, Experience, and Career Development Program," Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, July 27, 2017, p. 1.

[3] Frank Kendall, Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, "Better Buying Power 2.0: Continuing the Pursuit for Greater Efficiency and Productivity in Defense Spending," memorandum, Washington, D.C.: Department of Defense, November 13, 2012.

employee-development tool is intended to "ensure that everyone who touches acquisition has the skills required to ensure successful acquisition outcomes."[4] DAU's Continuous Learning Center uses the AWQI eWorkbook to allow individuals to track their on-the-job experience with various acquisition skill sets and to determine which areas might require additional developmental opportunities.[5] Like DoD career field functional competencies, AWQI competencies are divided into career fields (e.g., IT, PM, and ENG) with a varying number of competencies in each one and some overlap across the three career fields. AWQI is structured to have a competency unit, similar to the functional competencies, followed by a competency element, product, and, at the lowest level, a task.

Commercial Models

Version 3 of the SWEBOK Guide was developed by the Institute of Electrical and Electronics Engineers (IEEE) Computer Society to help "promote the advancement of both theory and practice for the profession of software engineering."[6] The SWEBOK Guide is organized into 15 software "knowledge areas," which are then broken down into "topics," including descriptions of the critical aspects of the topics. These subcategories include, but are not limited to, Software Requirements, Software Testing, Software Maintenance, and Software Engineering (SwE) Process. This effort was designed to reach five objectives:

1. promote a consistent view of software engineering worldwide
2. specify the scope of, and clarify the place of, software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics

[4] DAU, Acquisition Workforce Qualification Initiative, undated.

[5] DAU, "About AWQI," webpage, undated.

[6] Institute of Electrical and Electronics Engineers (IEEE) Computer Society, *Software Engineering Body of Knowledge (SWEBOK)*, undated; P. Bourque and R. E. Fairley, eds., *SWEBOK V3.0: Guide to the Software Engineering Body of Knowledge,* Piscataway, N.J.: IEEE Computer Society, 2014, p. xviii.

3. characterize the contents of the software engineering discipline
4. provide a topical access to the Software Engineering Body of Knowledge
5. provide a foundation for curriculum development and for individual certification and licensing material.[7]

SWECOM lists and defines the competencies that are relevant to "software engineers who participate in development of and modifications to software-intensive systems." The model is based on version 3 of the SWEBOK Guide and was also developed by IEEE. It consists of 13 "competency areas" and 60 "competency labels" that are critical for those in engineering-intensive roles, including both technical and behavioral KSAOs. The competency areas included in SWECOM cover the following software-related skills:

- Requirements
- Design
- Construction
- Testing
- Sustainment
- Process and Life Cycle
- Systems Engineering
- Quality
- Security
- Safety
- Configuration Management
- Measurement
- Human-Computer Interaction

In addition to SWEBOK and SWECOM, we reviewed two education-focused efforts that contain guidelines for software engineering curricula. First, the IEEE Computer Society joined efforts with the Association for Computing Machinery (ACM) to develop a set of guidelines for undergraduate degree programs to aid both aca-

[7] Bourque and Fairley, 2014, p. xxxi.

demic institutions and accreditation agencies.[8] In addition to describing the kinds of material that should be covered, examples of courses and curricula are provided as a reference for what a software engineering undergraduate degree program could look like.[9] The Integrated Software & Systems Engineering Curriculum Project involved authors from a variety of academic institutions, software corporations, and professional societies.[10] These authors developed a similar set of guidelines for graduate degree programs in software engineering in order to

- improve existing graduate programs in software engineering (SwE) from the viewpoint of universities, students, graduates, software builders, and software buyers
- enable the formation of new graduate programs in SwE by providing guidelines on curriculum content and advice on how to implement those guidelines
- support increased enrollment in graduate SwE programs by increasing the value of those programs to potential students and employers.[11]

Other Department of Defense Competency Efforts

SATEWG was formed of organization representatives tasked with developing a software competency framework that could be used to inform acquisition career field competency models and courses.[12] To

8 Joint Task Force on Computing Curricula, *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, New York: IEEE Computer Society and Association for Computing Machinery, 2015.

9 Joint Task Force on Computing Curricula, 2015.

10 A. Pyster, ed., *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering,* Integrated Software & Systems Engineering Curriculum Project, Hoboken, N.J.: Stevens Institute of Technology, September 30, 2009.

11 Pyster, 2009, p. vi.

12 D. S. Lucero, "Influencing Software Competencies Across the DoD Acquisition Workforce," *Journal of Defense Software Engineering,* 2010, pp. 4–7.

accomplish this, the group "reviewed 234 software competencies and 790 competency elements" from DAU courses, best practices, and existing competency models.[13] This review resulted in four knowledge areas or high-level descriptions, and 29 competencies to be included in their software competency framework. The four knowledge areas included are

- Software Acquisition and Sustainment Planning
- Software Development Considerations
- Software Management
- Post-Deployment Software Support.

The representatives also reviewed persistent issues in the field of software acquisition, of which they found 123.[14] Additionally, the working group aimed to identify and address gaps in the Defense Acquisition Workforce Improvement Act curriculum.[15]

Similarities and Differences Among Competency Models

The competency models were assessed for differences and similarities in the way they were structured and the competencies that were included. As the previous sections indicate, the models differed greatly in how they labeled the various "levels" of the competencies. In particular, different levels of specificity were given different names. Whereas SWECOM used "competency area" as the highest level and "competency label" and "competency definition" as the most specific levels, the SATEWG model had only "knowledge area" and "competency." The AWQI models contain detail across multiple levels starting with "career field" at the top, followed in order of specificity by "functional

[13] Lucero, 2010, p. 5.

[14] Lucero, 2010.

[15] J. L. Finley, "Establishment of Software Acquisition Training and Education Working Group," memorandum, Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, February 19, 2008.

unit of competence," "competency," "competency element," "product," and "task."

Despite these differences, similarities among the competency models helped us identify relevant competencies for the software acquisition workforce. For example, Data Management appeared as a competency in the AWQI PM and ENG models, as well as in the SATEWG model, and it was included in our software acquisition competency model. Competencies related to contracting, software development, and sustainment followed similar patterns. As an early phase of developing a competency model for the software acquisition workforce, this review proved a valuable step in determining which competencies were most likely to be relevant and critical going forward. Appendix E presents further details on each of the competency models reviewed, including information on the number of "levels" and competencies and examples of some of the competencies for each model.

Other Occupational Sources Reviewed

Although the federal government does not have specific career fields for software, we reviewed occupational requirements for occupational series in which software professionals are likely to reside, including computer science, computer engineering, general engineering, and IT. The information about these occupational series provides lists of typical functions. For example, a typical function for a computer scientist is the "development of software systems using a knowledge of techniques, procedures, and processes such as operating system theory, data structures, computer system architecture, software engineering, and computer communications."[16] Although topic areas overlapped with the software acquisition definition, the functions are not comprehensive and serve primarily as examples of occupation duties.

[16] OPM, "Position Classification Flysheet for Computer Science Series, GS-1550," *Computer Science Series, GS-1550*, 1988, p. 2.

Another potential source of information about software occupations is provided by O*NET,[17] which is the online repository for occupations across the entire U.S. economy. O*NET collects information from workers in each occupation to describe work activities, knowledge, skills, and ability requirements. Although O*NET provides information about several software-specific occupations such as “Software Developers, Systems Software,” “Software Developers, Applications,” and “Software Quality Assurance Engineers and Testers,” the information is both very broad and less comprehensive than the competency models described previously in this chapter.

[17] O*Net Online, webpage, undated.

CHAPTER FOUR

Commercial Industry Perspective

In this chapter, we describe several efforts to identify modern software competencies based on commercial industry trends. To identify these trends and potential competencies, we reviewed relevant research literature and trade publications; held discussions with a select number of commercial industry subject matter experts;[1] and compared DoD and commercial industry job announcements, training, and education course descriptions and objectives. We also provide a more detailed literature review of modern software practices in Appendix B.

Software Industry Trends

We began with a literature review seeking input on trends in the software industry. Software development is a relatively young industry that continues to rapidly evolve its practice. And like any other profession, software development is subject to hype and changing fashions. As such, research in this area is not always straightforward. First, there is active marketing around every aspect of software development as firms jostle for position and influence. Distinguishing real trends from hype is challenging. Second, there are very few quantitative comparative studies of "what works" in software development. There is general agreement that software project success is highly dependent on factors that are not easily measured or quantified, such as the novelty of the

[1] This study was conducted according to all Human Subjects Protection Committee guidelines.

software, the complexity of internal and external dependencies, and development staff expertise. Conducting evidence-based comparative studies of specific development practices is extremely difficult due to our inability to quantify these critical factors. Finally, today's best practice often becomes tomorrow's obsolete practice.

Considering these challenges, we identified four principal industry trends in our review:

- **Changes in the sequencing of the activities used in the production of software, described as a software development lifecycle (SDLC) model**: While this change is commonly referred to as the change from a Waterfall to an Agile model, at root it is the trend to shorten the time between defining what the software needs to do (e.g., "the idea") and producing working product.
- **Changes in software development architecture from monolithic development to ecosystems**: Whereas in the past, a single vertically integrated firm would produce a monolithic software product including hardware, operating system, and application (e.g., the original release of Microsoft Word), today's software is more likely to be produced by a more organic ecosystem of development teams that include commercial entities, open-source foundations, and individual developers.[2]
- **Increasing diversity in software deployment architectures**: While in the past, software was envisioned as running on a standalone processor, today's software is typically spread across multiple processors of varying capability, ranging from large clusters of servers linked by hypervisors (e.g., the cloud) to the phone in our pocket. Stand-alone software products are becoming increasingly rare in today's world.[3]

[2] While the original releases of MS Word ran only on Microsoft operating systems, Word dominates the word processing market today because Microsoft moved to a software stack and business model that allowed later versions of the application to run on operating systems besides Windows.

[3] The software deployment architecture defines how the software is organized for deployment onto the computing infrastructure. A typical application today has elements deployed

- **Increasing automation in software practices**: There has been an explosion of software products that automate the activities needed to develop software. These tools provide a means to automate tasks ranging from tracking change, to producing documentation, to performing security vulnerability scans, to checking for compliance to standards for design, code, or test, to fully automated build, integration, and test. Recently, these automation "pipelines" have expanded to include the automated delivery and deployment of software into operations.

While each of these trends could have developed on its own, they are highly interdependent and have co-evolved in a highly symbiotic relationship. The desire to shorten the time from idea to working software is supported by the improvements in automating the tasks of software development; changes in both the development and deployment architectures are enabled by the concept of a software *stack*, where layers can be independently developed within an ecosystem and deployed to distributed processors. In turn, the software stack concept grew from the desire to reuse software in an attempt to shorten the time from idea to working software. Automated pipelines for products developed in specific ecosystems are tailored to specific deployment architectures and life-cycle models. These trends are described in the sections below.

We relied heavily on SMEs both in the commercial software industry and those working within DoD to investigate how those trends are evolving and their impact on the software competencies documented in this report. An in-depth discussion of each of these trends and of its impact on our software competency model is provided in Appendix B.

Industry Subject Matter Expert Perspectives

To obtain the perspective of software professionals in commercial industry, we conducted four unstructured interviews between Janu-

across servers (including the cloud), personal computers, phones, and the small devices that make up the internet of things (IoT).

ary and April of 2018. Each SME, selected from a RAND researcher's professional network, had extensive software development experience in the private sector. All were well versed in identifying and assessing software competencies, industry trends in software development, and the management of software development projects. All had undergraduate degrees, and three had earned doctorates in computer science or electrical engineering. All SMEs had seven or more years of experience working at large, multinational technology corporations, and three had worked at or co-founded technology start-ups over the course of their careers. They worked in a wide range of technical domains that included speech recognition, ad hoc networking, machine learning, hardware verification, mobile app development, deep learning, and data science.

The objective of the interviews was to learn about the key competencies that were important for software engineers working in the private sector today and that would be needed to support near-term (three- to five-year horizon) trends in the industry, and how to assess those competencies. Three themes emerged from these interviews:

- Strong technical skills are important for fulfilling software engineering roles
- Two meta-skills that are very important in the domain of software engineering are the ability to work collaboratively and the ability to participate in rapid prototyping.
- Previous experience is among the strongest indicators of a candidate's potential success in a software engineering role.

Strong Technical Skills

All of the SMEs spoke about the importance of general computer science skills. These skills included familiarity with general-purpose programming languages (e.g., Java and Python) and a thorough understanding of specific languages/methods needed for web development and cloud computing. Understanding data structures and foundational computer science is key. It was pointed out that modern web technologies, and advanced programming languages more generally, are a very fast-moving competency set with constant changes. When hiring for a role that would use such skills, employers should look for a candidate who either demonstrates the specific technical skill relevant to the job role he or she will be filling or has a demonstrated track record showing an ability to continuously learn technologies and languages.[4]

The SMEs believed a solid background in computer science is critical for those working in product, project, or program management (all abbreviated as PMs) roles.[5] The SMEs cautioned that to be successful in software acquisition, PMs must have the technical expertise to

- Understand what software can do and why: A good PM understands both engineering and product features and thus has technical depth to understand how a product should be developed but also know how the product will be used and is able to specify metrics against which the usage would be measured.
- Vet the offerings of third-party vendors or the planning of their own teams: Software is easy to fake, especially when it hasn't been built yet or is very new. As one SME put it, a PM needs to have a little light that goes off when someone is just throwing buzzwords.
- Achieve the best technical solutions: If the PM is not asking sophisticated questions, it may be impossible for the developers to volunteer appropriate recommendations or the best technical solution.

[4] It was striking that none of the SMEs felt it was industry's job to train people for these skills. The modern software industry has elements of the gig economy, with the burden of keeping skillsets current falling squarely on the shoulders of the employee. This may change if software talent becomes harder to find.

[5] A SME who worked at a sophisticated, top-tier multinational went as far as to say that they were unaware of any PMs at their company who did not have a technical background. PMs in private sector technology companies are overlapping roles with loosely defined boundaries that involve management of the software effort. This includes defining the features or attributes needed, the overall product design, identification of the target market, and prioritization of bugs and features. Product or program managers generally focus on end-to-end considerations for the software's entire life cycle. PMs are more focused on shorter time periods; for example, if a project launch is scheduled for a given date, the PM will focus on what needs to happen up until that date. In each of the roles, PMs work closely with the technical professionals to produce cost and schedule estimates.

Meta-Level Skills

Two meta-level skills emerged during the SME interviews: the importance of collaborative work and the need for rapid prototyping. Given that most software is developed by teams, an appreciation for collaborative work is important for all software professionals. SMEs looked for candidates who had experience using collaboration tools such as Confluence, Jira, Git, and other feature-tracking and software-version control solutions, as well as team management techniques such as the daily standup (i.e., the "Scrum").

Rapid prototyping, the ability to quickly create a preliminary software artifact that is then evaluated for refined implementation, was also considered a key skillset. Implicit in rapid prototyping is the notion that a prototype may be deemed unsatisfactory and will be scrapped. As one SME put it, software acquisition professionals must accept the idea that failure is a necessary step toward success.

Experience

All the SMEs shared their thoughts on the challenge of assessing the competencies and predicting the performance of a candidate for a software engineering role. They typically broke this out by two categories of positions: generalist and specialist. Broadly speaking, a generalist may work with a general-purpose programming language to implement logic, algorithms, or other processing. In contrast, very specialized skills may be sought to fill specific niches on the team. For example, a robotics team may have an opening for a computer vision specialist with knowledge of motion-detection algorithms that execute efficiently on small embedded processors. In this case, neither general knowledge of algorithms nor general knowledge of embedded software is sufficient alone; the candidate must have both.

One SME described a process of evaluating generalists and specialists. In the case of generalists, the first step is to assess candidates via a triaging decision tree. Candidates who get past the decision tree are then given a programming test in which they solve a generic data structure or algorithms problem. In contrast, the résumés of specialist candidates are reviewed by a committee of employees who currently work in the role being hired for. The committee then gives selected specialist candidates a programming assignment to build a custom system (candidates are typically given about a week to complete the implementation). Our SMEs acknowledged that the programming exercises are, in fact, often not an accurate gauge of whether a candidate would be successful in the role. However, in "prestige" private-sector technology companies with many applicants, companies may prefer a bias toward false negatives as opposed to false positives in hiring.

In the case of PM roles, while all of the SMEs indicated that formal training in a technical field was very helpful for determining if a candidate PM would be successful, it was not a sufficient indicator. Instead, they stated that the best way to gauge a candidate PM is to see evidence that the PM has worked in the role previously.[6]

Review of Position Announcements and Course Descriptions

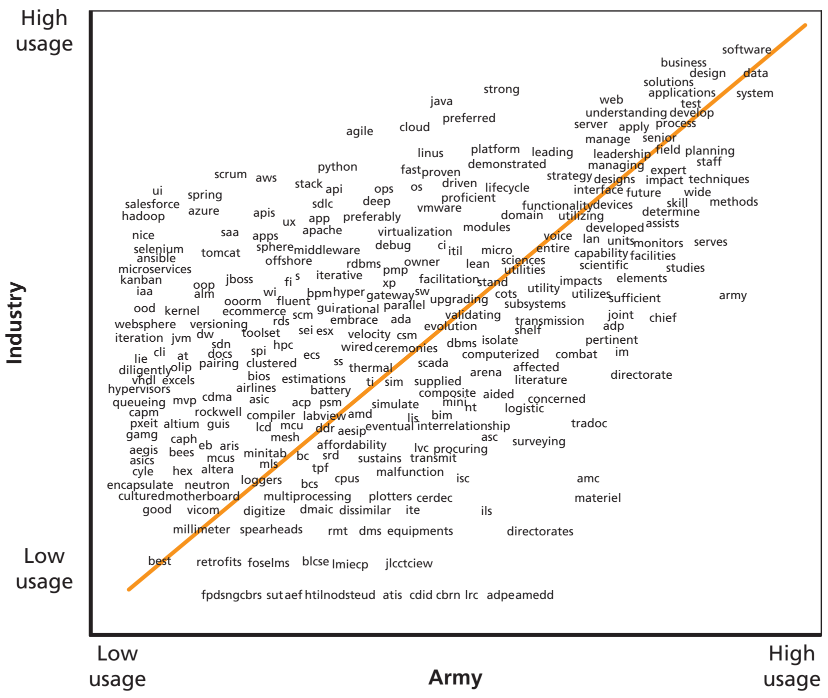
Our review of open software job positions and of software course descriptions in industry compared with those in DoD was exploratory but provided valuable insights. To evaluate the content of job positions, we downloaded 20,637 software job openings from an online website using a public application programming interface with the search keywords "acquisition" and "software." We repeated this process using the same search terms to download 14,402 open positions listed in the Army's Fully Automated System for Classification (FASCLASS).[7]

To explore differences in position announcements, we compare the frequency of words used by the Army with words used by commercial industry (Figure 4.1). The horizontal position of a word in the plot

[6] Presumably this experience is gained in another company that is more risk acceptant. Short of direct experience, one SME shared a method for evaluating candidate PMs in which the candidate is asked to respond to a set of emails that simulate a real-world scenario over the course of a day. For example, the candidate might be asked to imagine working for a consulting firm that depended on a cloud computing service and the service was down; in such a situation, how would he or she manage the crisis?

[7] FASCLASS is web-based system that contains information on Army civilian positions to include details on OPM occupational series, position titles, and job duty descriptions.

Figure 4.1
Word Usage in Army and Industry Software Job Postings

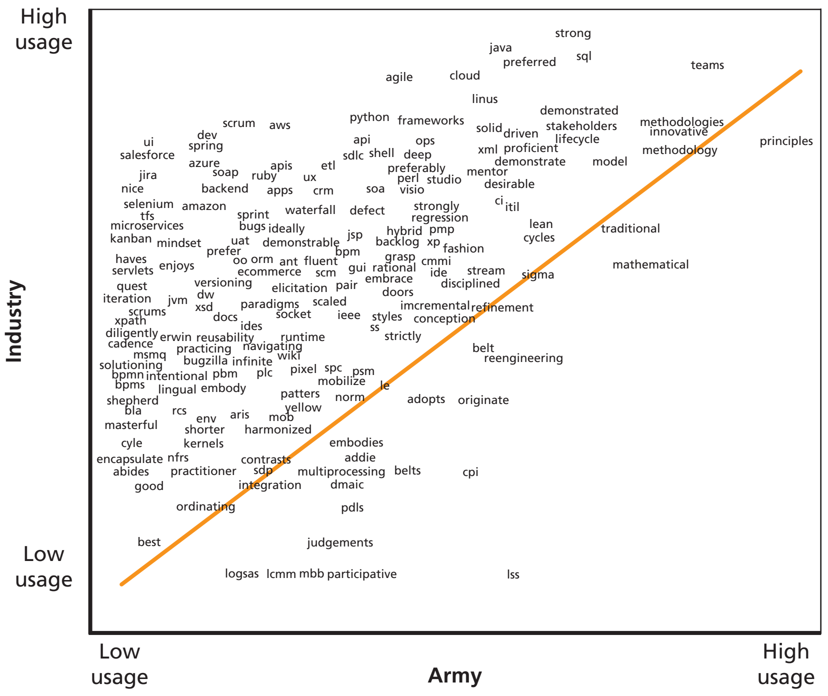


describes its frequency in Army position descriptions, and its vertical position denotes how frequently it is used in industry job postings.[8] Words that are close to the red line share similar relative frequencies. As one might expect, "software," "systems," "data," "design," and "networks" are frequently used in both Army and industry job postings. However, we noticed that words specific to software practices appeared much more frequently in industry postings.

[8] Figure 4.1 was developed by training a word2vec model on both the industry and Army position descriptions and then selecting 500 words for each of the following five models. The models were then used to select words that are frequently used in close proximity to terms that we found to be of significance in the software trends analysis described in Chapter Three.

Words used in models: 1. embedded, systems; 2. Agile, Waterfall, Iterative, methodologies; 3. sustainment, integration; 4. container, cloud; 5. software, engineering.

Figure 4.2
Software Life-Cycle Management Terms in Army and Industry Job Postings



To further explore potential differences, we filtered just those words used in close proximity to terms commonly associated with software life-cycle management: "Agile," "Waterfall," "Iterative," and "methodologies." The results, shown in Figure 4.2, demonstrate a striking lack of emphasis in software life-cycle management practices in Army software acquisition job postings. These life-cycle management and related commercial industry terms reinforced findings from the literature review to further guide software competency development. For example, terms such as "scrum," "kanban," and "sprint" suggested commercial industry values professionals with knowledge of modern methodologies focused on process improvements, Iterative development, and adaptive workflows. These terms were also integrated as examples in the competency definitions (for example, see Development Tempo, which is detailed in Appendix F).

It is important to note that observed differences in these analyses may simply reflect differences in how positions are tagged by different search engines. For example, it is possible that the Army positions returned by the search terms were not as software-specific compared with commercial industry jobs. Although beyond the scope of this effort, a more thorough analysis could ask SMEs to identify comparable positions in the Army and commercial industry to determine if these observed differences remain. This type of follow-on analysis would help to address any potential bias in search-engine selection of positions.

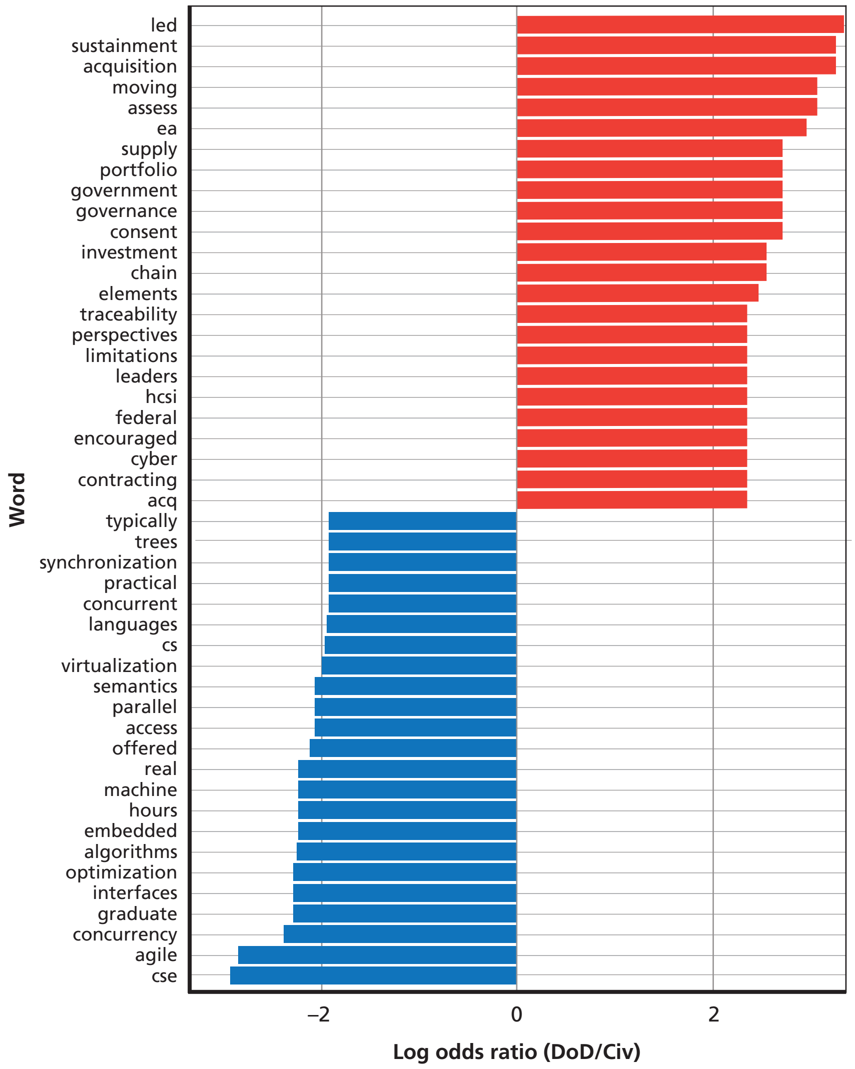
Building on the comparisons in position announcements, we conducted a similar comparison of words used in course descriptions by DoD and civilian academia. We downloaded software course descriptions from 14 U.S. universities considered to be top educators in software engineering, computer engineering, or computer science. We did the same for five DoD academic institutions: Naval Postgraduate School (NPS), National Defense University (NDU), DAU, Air University, and the Air Force Institute of Technology (AFIT). We then used text analysis to identify key differences in terms used by DoD and industry.

Figure 4.3 shows the result of our analysis of words used in the software course descriptions. Words that are most likely to be used in DoD course description but not in civilian academic institutions (as measured by the log odds ratio[9]), are shown in red. Words that are more likely to be used in civilian academia (as measured by log odds ratio), are shown in blue. Strikingly, DoD course descriptions contain very few words that are specific to software development, such as "concurrency," "algorithms," "virtualizations," "embedded," and "Agile." Instead, they emphasize words specific to acquisition: "governance," "investment," "tracking," and "assessing."

These differences may simply reflect DoD's emphasis on management and DoD-specific acquisition requirements. That is, DAU does

[9] Log odds is used to represent the ratio of relative term usage by DoD to the relative term usage by civilian academia. The terms represented in red have a relative frequency that is more than two times greater in DoD course descriptions compared with civilian academic course descriptions.

Figure 4.3
Commonly Used Words in Software Acquisition Course Descriptions



NOTE: Red indicates words most likely to be used in DoD course description but not in civilian academic institutions; blue indicates words more likely to be used in civilian academia.

not expect to train acquisition professionals how to write code. Given the different educational objectives across these institutions, it will be an important step in the future to determine how DoD will ensure acquisition professionals have and maintain important software competencies. Some competencies may be used to establish hiring criteria, and others may need to be taught by DAU. These questions can be addressed as part of a broader validation study (see Chapter Eight).

It is important to note that these course description (and position announcement) analyses were largely exploratory, intended to identify potential areas of emphasis. In general, we found the terms used both in commercial industry position announcements and course descriptions to be helpful in reviewing the comprehensiveness and definitions of competencies. Common terms were also reviewed during discussions with SMEs during the competency revision process, as described in the following chapter.

CHAPTER FIVE

From Initial to Revised Competency Model

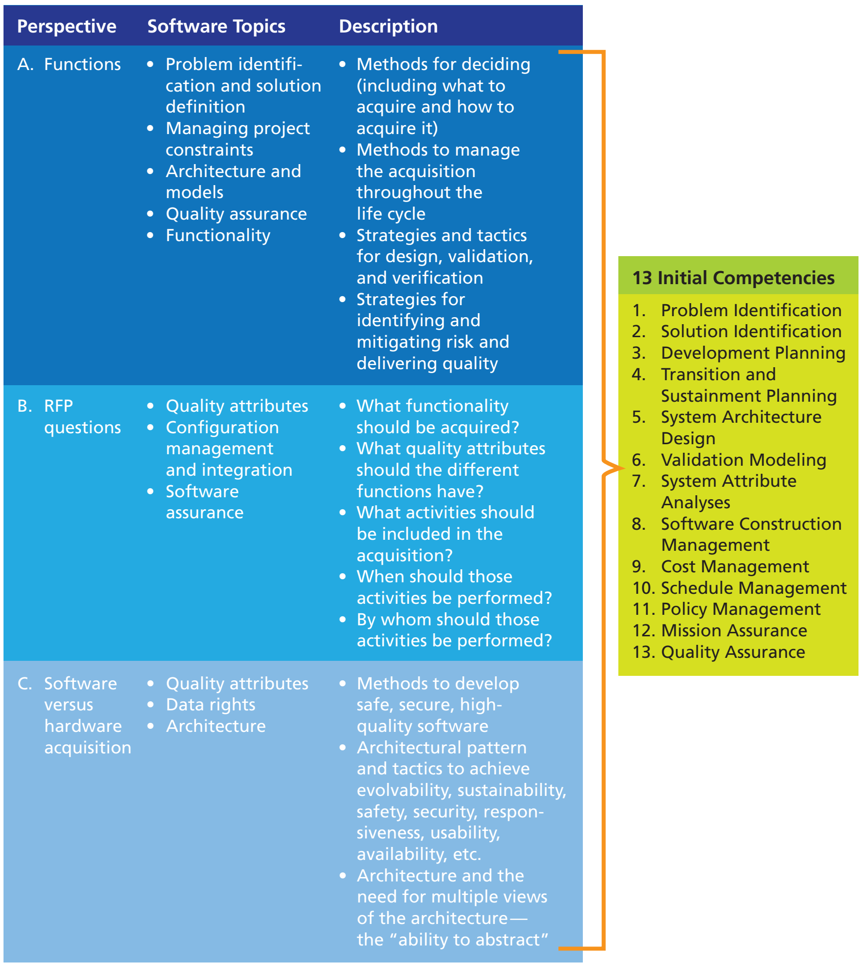
This chapter presents the initial competency model, steps taken to gather and integrate stakeholder feedback, and the revised competency model. The initial model, containing 13 competencies, was informed by the review and findings presented in Chapters Three and Four. It was then significantly revised and expanded in several iterations, eventually evolving into a model with 48 detailed competencies. The full model with detailed definitions is provided in Appendix F.

Initial Competency Model

To adhere to the DCPAS approach for competency model development, we were initially asked to develop 10 to 12 competencies in which DoD software acquisition professionals need to be capable and successful.[1] With guidance from two industrial-organizational psychologists with backgrounds in competency model development, three independent RAND researchers with an accumulated 70 years of experience in software management, development, and acquisition reviewed existing competency models and modern practices to identify potential competencies. Researchers also considered three different perspectives when generating competencies: (1) primary software functions, (2) topics included on a request for proposal (RFP) for software-dependent systems, and (3) primary differences between acquiring software versus

[1] Ten to twelve is the number of competencies that the tools used by DCPAS to validate competency models is designed to handle.

Figure 5.1
Guiding Perspectives for Initial Competency Development



Perspective	Software Topics	Description
A. Functions	• Problem identification and solution definition • Managing project constraints • Architecture and models • Quality assurance • Functionality	• Methods for deciding (including what to acquire and how to acquire it) • Methods to manage the acquisition throughout the life cycle • Strategies and tactics for design, validation, and verification • Strategies for identifying and mitigating risk and delivering quality
B. RFP questions	• Quality attributes • Configuration management and integration • Software assurance	• What functionality should be acquired? • What quality attributes should the different functions have? • What activities should be included in the acquisition? • When should those activities be performed? • By whom should those activities be performed?
C. Software versus hardware acquisition	• Quality attributes • Data rights • Architecture	• Methods to develop safe, secure, high-quality software • Architectural pattern and tactics to achieve evolvability, sustainability, safety, security, responsiveness, usability, availability, etc. • Architecture and the need for multiple views of the architecture—the "ability to abstract"

13 Initial Competencies

1. Problem Identification
2. Solution Identification
3. Development Planning
4. Transition and Sustainment Planning
5. System Architecture Design
6. Validation Modeling
7. System Attribute Analyses
8. Software Construction Management
9. Cost Management
10. Schedule Management
11. Policy Management
12. Mission Assurance
13. Quality Assurance

hardware systems (Figure 5.1). To ensure comprehensiveness, competencies were compared against existing models discussed in Chapter Three and Appendix E (see Appendix C), against the System Engineering Capability Maturity Model-Integrated (CMMI), and against Scott Ambler's detailed model of the Agile SDLC.

Revising the Competency Model Through Stakeholder Feedback

The following sections describe the steps taken to gather stakeholder feedback and use that information to revise the initial set of competencies. In general, modifications were made to the initial set of competencies following each phase of feedback. We obtained feedback from the sponsor office, SME panel workshops, and software professionals, who provided written feedback.

Feedback on the First Draft of Competencies

Representing the sponsor's office and DAU, seven key stakeholders who had backgrounds in software engineering, development, training, management, and acquisition reviewed and provided feedback on the initial competency list and definitions. There was a strong consensus among these stakeholders that the competencies did not adequately reflect the nuanced differences between how software is developed today versus how it was developed 15 to 20 years ago. Perhaps more critically, the model did not adequately capture the strong differences in competencies needed for hardware versus software acquisition. In other words, SMEs suggested that the initial competencies were a better reflection of primary occupational competencies (Tier 2, as described in Chapter Two) commonly used to define competencies that apply across functionally related occupations. However, these Tier 2 competencies are less useful for describing the requirements of Tier 3, suboccupational specialties within software (e.g., a set of software architecture competencies within a software occupation[2]).

The key stakeholders were also concerned that if this initial set of competencies were adopted for DoD software acquisition, it would breed complacency and that acquisition professionals would not obtain the training in modern software development methodologies believed to be necessary to improve the effectiveness of software acquisition in DoD. The specific feedback suggested potential gaps in six key areas:

[2] As noted throughout this report, software does not exist as an official OPM or DoD occupation. Therefore, this example is provided to demonstrate the relationship between Tier 3 competencies and a hypothetical software occupation.

(1) Agile software development including DevOps, continuous integration, and continuous deployment; (2) machine learning and artificial intelligence; (3) software estimation, measures, and metrics; (4) software development and testing; (5) software site reliability engineering; and (6) information assurance and cybersecurity.

Feedback on Subsequent Drafts

With this feedback in mind, the RAND team revisited the commercial industry analysis findings (Chapter Four and Appendix B) with the goal of capturing suboccupational competencies (Tier 3), which more extensively reflect modern software practices. Although not formally mapped to specific sources, the revised model containing 46 competencies, organized into 11 competency topics, generally corresponds with the initial draft competencies. The competencies were then reviewed in workshops with SMEs from across DoD. Directors of Acquisition Career Management (DACMs) and functional career field leaders were asked to provide contact information for SMEs using the following criteria:

- Experience: professionals with at least five years of experience performing and/or supervising software acquisition activities
- Recent knowledge and/or experience in software acquisition (within the past two years)
- Representation: covers range of duties across component/agency, commands, and programs.

Characteristics of the SMEs who participated are provided in Tables 5.1 and 5.2.

Due to the greatly expanded number of competencies, we divided them into three overlapping sets for review, loosely titled (1) Software Program Management and System Engineering, (2) Software Technologists, and (3) Software Project Managers (Table 5.3).

Subject Matter Expert Panel Workshops

We invited 119 SMEs identified by DACMs to participate in one of nine workshops scheduled during the summer of 2018, with each workshop covering one of the three sets. Of the 31 SMEs who participated,

Table 5.1
Number and Background of Workshop Participants

Years of Experience	Number of Software SMEs	Number of Acquisition SMEs
Less than 10	3	3
10 to 20	10	12
20 or more	14	12

NOTE: Some participants had experience in both software and acquisition, so the table sum is greater than the total number of participants.

Table 5.2
Subject Matter Expert Area of Expertise

Area of Expertise	Number of Participants
Acquisition	6
Development	8
Education	2
Sustainment	5
Test & Evaluation	2
Acquisition; Development	2
Acquisition; Education	2
Acquisition; Sustainment	1
Development; Sustainment	3

most were currently working in software or had extensive experience in managing software programs in the past. SMEs had a range of expertise in sustainment, development, acquisition, and training and education. The number and distribution of SMEs is presented in Table 5.4.

SMEs were provided with an advance copy of the competencies and corresponding definitions for review. Each half-day workshop was structured in exactly the same way. We introduced the purpose of the

Table 5.3
Expanded Set of 46 Software Acquisition Competencies

Competency Topic	Competency	Software Program Mgt. and System Engineering	Software Technology	Software Project Management
Problem Identification	• Capabilities Elicitation	X		
	• Business Case Development	X		
Solution Identification	• Solution Risk/Reward Analysis	X		X
	• Cloud Computing	X		X
	• Software Ecosystems	X		X
	• Prototyping			X
Development Planning	• Development Management Approach	X	X	
	• Agile Software Development		X	X
	• Agile System Engineering	X	X	
	• Software Metrics	X	X	
	• Configuration and Version Control		X	X
Transition and Sustainment Planning	• Software Documentation		X	X
	• Sustainment Management Approach		X	X
	• Continuous Delivery			X
	• Continuous Deployment			X

Table 5.3—Continued

Competency Topic	Competency	Software Program Mgt. and System Engineering	Software Technology	Software Project Management
System Architecture Design	• Agile Enterprise Architecture	X		
	• Software Orchestration and Choreography Patterns			X
	• Software Deployment Patterns			X
	• Artificial Intelligence and Machine-Learning Applications			X
	• Augmented and Virtual Reality Applications			X
	• Cyber-Physical Applications and the Internet of Things (Embedded Systems)			X
	• Balancing Quality Attributes	X		X
	• Emerging Technologies	X		X
Modeling Functional Capabilities and Quality Attributes	• Use/Abuse Case Modeling	X		
	• Validation of Performance Requirements (Responsiveness, Latency, and Throughput)	X		X
	• Validation of Maintainability Requirements	X		
	• High Fidelity System Modeling	X		X
Building Secure, Safe, and High-Availability Systems	• Software Assurance	X		
	• Cybersecurity	X		
	• Safety Critical Systems	X		X
	• High-Availability Systems	X		X

Table 5.3—Continued

Competency Topic	Competency	Software Program Mgt. and System Engineering	Software Technology	Software Project Management
Software Construction Management	• Life-Cycle Management		X	
	• Backlog Management		X	
	• Release Management		X	
	• Agile Change Management		X	X
	• Automated Test and Continuous Integration		X	X
Software Program Management	• Effort Estimation		X	
	• Product Roadmap and Schedule Management	X	X	
	• Cost Management	X	X	
	• Legal Policy and Regulatory Environment Management	X	X	
	• Risk, Issues, and Opportunity Management	X	X	
Mission Assurance	• Quality Assurance	X		
	• Root Cause, Corrective Action	X		
	• System Integration and Testing	X		X
Professional Competencies	• Vision and Change Management	X	X	X
	• Design Thinking, Innovation, and Entrepreneurship	X	X	X

RAND study and then discussed the primary objectives for the workshop. These objectives included evaluating (1) the relevancy of each competency for DoD software acquisition, (2) the clarity of definitions for each competency, (3) potential gaps in content or missing competencies, and (4) potential to consolidate and merge competencies that

Table 5.4
Number and Representation of Subject Matter Experts in Panel Workshops

	Program Management and Engineering	Project Management	Software Technical Execution	Total
Air Force	7	4	3	14
Army	1	1	1	3
Navy	0	1	3	4
Defense Contract Management Agency	2	1	3	6
DAU	1	2	1	4
Total	11	9	11	31

may be redundant or too similar. The competencies were presented with definitions one at time. For each competency, participants were asked to provide feedback in relation to each objective. At the end of each workshop, we asked participants for closing thoughts on how the competency model could be further improved.

We modified the competencies based on SME feedback after each round of workshops and saw a decrease in the number of substantive comments received each round. As a result of the feedback, we tempered our use of "Agile" in favor of "modern" and reduced our use of "buzzwords" often associated with software development. We added definitions for software-unique terminology and moved some explanatory information from the competency itself to "context" statements. Over the course of the workshops, we dropped the grouping by categorization because we found it distracted reviewers from concentrating on the competencies themselves.[3]

[3] Social science researchers trying to understand how various groups think about a complex subject, which software development certainly is, ask individuals to sort elements of a list into piles and then name the piles. Often two individuals do not sort into the same categories. Researchers then probe for why individuals sorted and named the piles to better understand how they think about the subject. Categorization techniques often reveal more about the person doing the categorization than they do about the subject at hand. A categorization that is useful to a program manager may be very different from one that is useful to a software sustainment lead.

At all times, we remained vigilant in our efforts to express the competencies in terms that are agnostic of particular software development "tribes" or entities with a monetary interest in particular tools or methods.

Almost inevitably, the number of competencies, tasks, definitions and context statements grew with each round as we added nuance and refined concepts. The result, however, was an infinitely improved statement of the 48 competencies.

Written Feedback

As a final step in revising the competencies, RAND distributed them by email to 120 DoD software professionals to provide feedback. The feedback period remained active for two months. In total, we received more than 40 comments from eight reviewers in that time period. We adjudicated each comment and updated the competencies to address key concerns. No new competencies were developed during this phase of feedback. None of the changes substantially altered the content of the competencies, but all did improve the clarity and accuracy of the descriptions.

The RAND-Developed Software Acquisition Competency Model

The conclusion of the feedback efforts resulted in a revised set of 48 competencies with detailed descriptions and additional context and related definitions. Table 5.5 contains the titles in the revised model and Appendix F provides detailed definitions and additional material. As mentioned previously, this model should not be considered final until it has been validated, which is a task that DoD will need to complete, and which we discuss further in the final chapter.

The competencies should be viewed as part of a living document that can be updated over time. Future evaluations and feedback from the software acquisition community may suggest areas in need of improvement. For example, redundancies may exist where competencies overlap too much and cannot be easily differentiated. Other competencies may need to be further split, or new competencies may

Table 5.5
Revised Set of Software Acquisition Competencies and Topics

Competency	Topic
Problem Identification	1. Capabilities elicitation
	2. Business case development
Solution Identification	3. Strategic risk/reward analysis
	4. Cloud computing
	5. Software ecosystems
	6. Model-based engineering
Development Planning	7. Development tempo
	8. Release planning
	9. Software development planning
	10. Planning for continuous delivery
	11. Planning for continuous deployment
	12. System engineering planning
	13. Software metrics
	14. Configuration and version control
Transition and Sustainment Planning	15. Software documentation
	16. Contracting for software development
	17. Data and proprietary rights management
System Architecture Design	18. Architectural design approach
	19. Software orchestration and choreography patterns
	20. Software deployment patterns
	21. Artificial intelligence and machine-learning applications
	22. Augmented and virtual reality applications
	23. Embedded systems
	24. Balancing quality attributes
	25. Emerging technologies
Modeling Functional Capabilities and Quality Attributes	26. Use/abuse case model
	27. Validation of performance requirements
	28. Validation of sustainability requirements
	29. High fidelity system modeling
Building Secure, Safe, and High-Availability Systems	30. Software assurance
	31. Cybersecurity
	32. Safety critical systems
	33. High-availability systems
Software Construction Management	34. Life-cycle management
	35. Detailed backlog management
	36. Release management
	37. Change management
	38. Automated test and continuous integration
Software Program Management	39. Effort estimation
	40. Product roadmap and schedule management
	41. Cost management
	42. Legal policy and regulatory environment management
	43. Risk, issues, and opportunity management
Mission Assurance	44. Quality assurance
	45. Root cause, corrective action
	46. System integration and testing
Professional Competencies	47. Strategic planning and change management
	48. Innovation and entrepreneurship

NOTE: The hierarchical structure of topics to competencies is not fixed and can be reorganized to meet a variety of organizational objectives.

emerge. Once validated, these competencies provide a valuable framework to guide software acquisition talent initiatives. DoD could use this model to identify which competencies are needed for specific positions, craft position announcements, and determine program needs. If the competencies are needed at the time of hire, specific assessments (e.g., structured interview questions, knowledge tests) should be developed and used to ensure job applicants have the requisite level of proficiency. DoD should also consider using these competencies to inform assignment decisions and to evaluate training needs. The full potential of any competency model requires an effective implementation plan and integration across DoD. Simply having a list of competencies is necessary but not sufficient to improve software acquisition practices. DoD also needs to ensure software professionals are tracked and receive the necessary training. We discuss these topics in the following chapters.

CHAPTER SIX

A Review of Software Training and Education

In addition to determining which competencies are needed for a software acquisition career, it is important to determine what training and education resources exist and what may still be needed for developing these competencies. To achieve this, RAND reviewed 394 courses related to software offered by DAU and by other DoD and civilian institutions. A full list of these courses can be found in Appendix G. In this chapter, we provide an overview of the findings of this review and also discuss potential gaps in select DAU courses and options for addressing those gaps. These options may include leveraging a mix of other DAU, DoD, or civilian courses, on-the-job training, and self-directed learning programs.

This review should be viewed as an initial first cut since the competencies have not been validated. Validation is necessary to determine the relative importance of each competency. In addition, efforts should be taken to determine the current and required proficiency levels for each competency. The results from these efforts are necessary to provide a more robust analysis of potential competency gaps and to ensure resources addressing gaps are appropriately prioritized. Recognizing these limitations, this review provides a preliminary linkage between competencies and educational courses. The linkages can be used as a first step toward a more thorough review of courses that may address proficiency gaps identified by DoD in the future.

Software Courses

Defense Acquisition University

Course descriptions and objectives for 93 DAU courses relevant to the IT, ENG, and PM career fields and software more generally were reviewed. For a majority of the courses, the study team was able to review both the course description and course objectives. However, some course objectives were not available, so it is possible that our evaluation missed things that actually are covered in the courses but were not apparent from the course description. An initial review of these courses suggests an emphasis on management and DoD-specific acquisition requirements, reflecting points raised by DAU SMEs that DAU is best positioned to provide training and education on DoD-specific issues related to acquisition, not to train personnel how to code.

Other Department of Defense Educational Institutions

Four other DoD institutions were reviewed including AFIT, NDU, NPS, and Air University (Table 6.1). These institutions offer degree programs in at least one of the following specialties: SwE, Computer Science, Systems Engineering, Electronic Data Processing (EDP) Systems, Cybersecurity, IT Program Management, and Information Systems and Technology. The team reviewed course descriptions from 78 relevant courses offered by these institutions.

Table 6.1
Other Department of Defense Institutions Offering Software Training and Education

DoD Institution	Program(s)	Number of Relevant Courses
AFIT	Computer Science; Software Engineering; Systems Engineering	29
Air University	Data Systems (EDP)	18
NDU	Cybersecurity; IT Program Management	10
NPS	Computer Science; Information Systems and Technology; Software Engineering	21

Academic Programs

The top 14 computer engineering programs in the United States were pulled from the list developed by *U.S. News and World Report* in February 2018 and resulted in 223 course descriptions for the team to review. The civilian institutions, programs, and numbers of relevant courses are listed in Table 6.2.

Mapping Competencies to Software Courses

The study team started with a thorough review of all course descriptions and objectives. In general, civilian institution courses focused more on design development/specification, programming, and software engineering than did Information Systems Acquisition (ISA) courses (ISA 101—Basic Information Systems Acquisition, ISA 201—Intermediate Information Systems Acquisition, ISA 301—Advanced Enterprise Information Systems Acquisition, and ISA 320—Advanced Program Information Systems Acquisition) provided by DAU. To map courses to competencies, we asked DAU course managers to identify which competencies were represented in DAU ISA courses. More specifically, each ISA course manager evaluated the extent to which each competency was fully covered, partially covered, or not covered by his or her respective course. These evaluations were also reviewed by the Learning Director for Information Technology at DAU and forwarded to RAND for consolidation with our analyses of other courses.

In addition, a RAND SME reviewed the remaining courses (i.e., other DAU, DoD, and civilian) that could address potential gaps in DAU's core ISA curriculum. More specifically, the SME first reviewed all of the competencies and corresponding definitions. Next, the SME worked sequentially through each course and marked an "X" to indicate when a competency definition overlapped with content in the course objectives or description. Future analyses could further strengthen these linkages by reviewing course syllabi and materials. For the results of the ISA and other course mapping, please see Appendix H.

The results presented in the following sections should be interpreted cautiously, given the limited number of SMEs providing inputs.

Table 6.2
Top U.S. News and World Report Academic Software Programs

Civilian Institution	Program(s)	Number of Relevant Courses
California Institute of Technology	Project Management Certificate; Project Management Short Courses; Systems Engineering Short Courses	3
Carnegie Mellon University	Master of Science in Information Technology; Master of Software Engineering	16
Cornell University	Computer Science	10
George Mason University	MS Software Engineering	22
Georgia Institute of Technology	Computer Systems and Software	4
Michigan Institute of Technology	BS Computer Science and Engineering	12
Princeton University	Master of Science in Engineering	9
Purdue University	MS in Computer Science; Software Engineering Track	12
San Jose State University	BS Software Engineering; MS Software Engineering	36
University of California, Berkeley	BA in Computer Science; Master of Information Management and Systems; MS in Computer Science	20
University of Illinois, Urbana-Champaign	BS in Computer Science (Engineering); MS in Computer Science; Software Engineering Certificate	15
University of Michigan, Ann Arbor	MS in Computer Science and Engineering	11
University of Texas at Austin	MS in ECE with a concentration in Software Engineering	22
University of Washington	BS/MS in Computer Science; Graduate Courses in Computer Science; Professional Master's Program in Computer Science	31

SOURCE: "Best Computer Engineering Programs," *U.S. News & World Report*, 2018. Note: This list has since been updated with 2019 data, but we used the 2018 data.

A more thorough analysis following competency validation should be conducted using at least three independent SMEs to link course curriculum to competencies. Using multiple independent SMEs helps minimize potential biases and potentially informs changes needed to further clarify instructional objectives and/or competency definitions. In addition to SME linkages, other sources of information, such as student evaluations collected by DAU, could be used to determine how well competencies are being taught.

Potential Gaps in Software Curriculum at Defense Acquisition University

Fourteen of the competencies appear to have either minimal or no representation in the ISA curriculum. Additional reviews of the curriculum are needed to confirm gaps really do exist and to determine the best options for addressing the gaps. These options are discussed in the subsequent section. Noting the limitations we described previously, we find that seven of the software acquisition competencies are minimally represented in the ISA courses, meaning that only one of the courses very lightly touches upon material related to the competency (that is, the topic is introduced but not discussed further).

- Automated Test and Continuous Integration
- Capabilities Elicitation
- Change Management
- Embedded Systems
- High-Availability Systems
- Software Deployment Patterns
- System Engineering Planning.

Seven of the competencies have no representation across DAU's ISA courses:

- High Fidelity System Modeling
- Innovation and Entrepreneurship
- Model-Based Engineering
- Software Ecosystems
- Software Orchestration and Choreography Patterns

- Use/Abuse Case Modeling
- Validation of Performance Efficiency Requirements.

More information on the representation of the competencies in ISA curriculum can be found in Appendix H.

Options for Addressing Gaps

Several options exist to address potential gaps in the training and education of software acquisition professionals. However, DoD should first determine the relative importance of each competency and identify competency gaps prior to investing further training and education resources to address the potential gaps identified. Conducting a competency gaps assessment (e.g., DCAT) helps to determine if software acquisition professionals have already obtained the necessary KSAOs prior to joining DoD or if additional training and education are needed. Once gaps have been confirmed, DoD can decide among the different training and education options.

These options may include developing new courses or updating course material, leveraging other DoD institutions and courses, and expanding partnerships with other commercial education providers (e.g., universities) and massive open online courses (MOOCs) such as Coursera and edX. Formal courses are most effective when training needs to be provided to a large group of people and the concepts are transferable across services, organizations, and programs.

Other Courses Addressing Competencies

All of the competencies that appear to have minimal or no representation in DAU ISA curricula are covered in at least one course offered by other DoD institutions or by the top civilian institutions, and most are covered in multiple courses (Table 6.3). Assuming these competency gaps are confirmed and deemed important enough to address across the acquisition workforce, it would likely benefit DAU to partner with these institutions to either use their existing courses or determine how to best develop curricula for DAU to fill its gaps in covering software acquisition competencies. Additionally, a review of other DAU courses that cover software content showed that some of the competencies are represented in non-ISA DAU courses.

Table 6.3
Department of Defense, Civilian and Other Defense Acquisition University Courses That Can Fill Defense Acquisition University's Potential Gaps

Competency	Other DAU Courses	Other DoD Courses	Civilian Courses
Automated Test and Continuous Integration		• NPS Software Testing • AFIT Software Test Engineering	• George Mason University Software Testing • San Jose State University Software Quality Assurance and Testing • San Jose State University Software Quality Engineering
Capabilities Elicitation	• Introduction to Agile Software Acquisition (CLE 076) • Fundamentals of Systems Engineering (ENG 101) • Program Manager's Skills Course (PMT 400) • Core Concepts for Requirements Management (RQM 110)	• NPS Human-Computer Systems Interaction • NPS Requirements Engineering • AFIT Current Software Acquisition and Management Topics • AFIT Software Requirements Management • AFIT Software Requirements Engineering • AFIT Software Project Management	• University of California, Berkeley, Needs and Usability Assessment • University of Washington Principles of Software Engineering • Carnegie Mellon University Agile Software Development Frameworks: Theory • George Mason University Software Project Laboratory • Cornell University Software Engineering • University of Texas at Austin Requirements Engineering: Acquisition and Modeling • Purdue University Software Engineering • California Institute of Technology Software Engineering and Management • California Institute of Technology Agile Project Management Certificate Program

Table 6.3—Continued

Competency	Other DAU Courses	Other DoD Courses	Civilian Courses
			• San Jose State University Software Engineering Processes • San Jose State University Software Engineering I
Change Management	• Configuration Management (LOG 204)	• AFIT Software Deployment and Sustainment Techniques • AFIT Software Requirements Management • AFIT Software Requirements Engineering	
Embedded Systems			• Carnegie Mellon University Distributed Embedded Systems • Carnegie Mellon University Real-Time Embedded Systems • University of California, Berkeley, Introduction to Embedded Systems • University of Illinois, Urbana-Champaign, Embedded Systems • Cornell University Embedded Systems • Carnegie Mellon University Embedded System Software Engineering • University of Washington Advanced Topics in Human-Computer Interaction
High Availability Systems		• NPS Software Reliability • NPS Project Management for Enterprise Systems	• University of Michigan, Ann Arbor, Principles of Real-Time Computing

Table 6.3—Continued

Competency	Other DAU Courses	Other DoD Courses	Civilian Courses
Software Deployment Patterns		• NDU Information Technology Program Management	• Carnegie Mellon University Embedded System Software Engineering • University of Washington Advanced Topics in Software Engineering • San Jose State University Software Engineering Processes
System Engineering Planning	• Fundamentals of Systems Engineering (ENG 101) • Applied Systems Engineering in Defense Acquisition, Part I (ENG 201) • Applied Systems Engineering in Defense Acquisition, Part II (R) (ENG 202)	• NDU Enterprise Architectures for Leaders • NPS Requirements Engineering	• University of Illinois, Urbana-Champaign, Software Engineering I
High Fidelity System Modeling		• NPS Enterprise Architecture • NPS Software Reliability	• George Mason University Software Modeling and Architectural Design
Innovation and Entrepreneurship			• Carnegie Mellon University Agile Software Development Frameworks: Practice • Carnegie Mellon University Agile Software Development Frameworks: Theory • California Institute of Technology Agile Project Management Certificate Program

Table 6.3—Continued

Competency	Other DAU Courses	Other DoD Courses	Civilian Courses
Model-Based Engineering	• Introduction to Agile Software Acquisition (CLE 076) • Applied Systems Engineering in Defense Acquisition, Part II (R) (ENG 202) • Engineering Management Workshop (WSE 006)	• NPS Principles of Software Design • NPS Requirements Engineering	• University of California, Berkeley, User Interface Design and Development • University of California, Berkeley, Principles and Techniques of Data Science • University of California, Berkeley, User Interface Design and Development • University of California, Berkeley, Software Prototyping for Data Science and Information Management • San Jose State University Computer and Human Interaction • University of Washington Human Computer Interaction • University of Washington Software Development for Data Scientists • University of Texas at Austin Advanced Programming tools
Software Ecosystems	• Designing for Supportability in DoD Systems (CLL 008) • Sustainment of Software Intensive Systems (CLL 056)	• NPS Software Architecture • AFIT Managing Software Deployment and Sustainment • NPS Enterprise Architecture • NDU Emerging Technologies	• Cornell University Open-Source Software Engineering • University of Texas at Austin Middleware • Carnegie Mellon University Architectures for Software Systems

Table 6.3—Continued

Competency	Other DAU Courses	Other DoD Courses	Civilian Courses
		• NDU Data Management Strategies and Technologies: A Managerial Perspective • AFIT Current Software Acquisition and Management Topics • NPS Project Management for Enterprise Systems	• California Institute of Technology Software Engineering and Management • Carnegie Mellon University Management of Software Development for Technology Executives
Software Orchestration and Choreography Patterns		• AFIT Software Architecture and Design Management	• University of Washington Advanced Topics in Software Engineering • San Jose State University Software Engineering Processes
Use/Abuse Case Modeling		• NPS Software Reliability	• George Mason University Software Requirements Analysis and Specification • University of Michigan, Ann Arbor, Principles of Real-Time Computing
Validation of Performance Efficiency Requirements		• AFIT Software Requirements Engineering	• University of California, Berkeley, Software Engineering • University of Texas at Austin Verification and Validation • Purdue University Software Engineering • University of Washington Advanced Topics in Software Engineering • San Jose State University Software Engineering I

Informal and On-the-Job Training

A broader range of options exist when training needs are more localized to specific programs or a more limited number of personnel. In these cases, DoD could leverage or expand programs to provide more tailored education and training. For example, structured on-the-job training, job shadowing, and job rotations could be beneficial especially if DoD could identify highly effective programs to mentor more junior software professionals. Similarly, DoD could consider further developing exchange programs with the commercial industry partners to promote modern software practices. It is also possible that simple refresher training or self-directed learning could help address gaps especially if content was previously mastered. Some degree of self-directed learning may even be expected or necessary given the constant evolution of technology and software development practices.

In conclusion, we recommend conducting a competency gap assessment once the software acquisition workforce has been identified. The results of this assessment should guide discussions on how best to address any confirmed gaps. The best option(s) will depend, in part, on the extent and pervasiveness of the gaps across DoD.

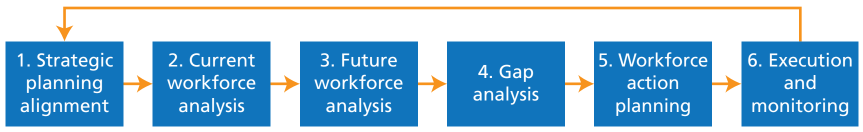
CHAPTER SEVEN

Identifying, Tracking, and Managing a Software Acquisition Workforce

In the previous chapters we have described the approach used to identify and define software competencies and to uncover potential gaps in training provided by DAU. The results of these efforts cannot move forward without first identifying the target workforce. Therefore, in this chapter, we present systems currently used by DoD and the federal government to track personnel and suggest options for DoD to track and manage a software acquisition workforce. To frame the discussion of these topics, we begin with a brief overview of steps commonly used in workforce planning.

A strategic workforce plan is a useful tool that can guide current and future workforce analyses and drive actions to identify and close proficiency gaps (Figure 7.1). DoD has initiated efforts to define the first step, "Strategic Planning Alignment," through studies conducted by DSB and the DIB. These studies reinforce senior leader priorities to improve software acquisition.

Figure 7.1
Strategic Workforce Planning Process Steps


1. Strategic planning alignment
2. Current workforce analysis
3. Future workforce analysis
4. Gap analysis
5. Workforce action planning
6. Execution and monitoring

SOURCE: U.S. Department of Defense, Civilian Personnel Advisory Services, *Strategic Workforce Planning Guide*, November 23, 2016.

Executing step 2, Current Workforce Analysis, is complicated by a lack of available data defining the software acquisition workforce. Therefore, DoD must first determine who performs software acquisition functions before a workforce analysis can be conducted.

Currently, there is no established system for identifying or tracking who performs software functions in DoD. That is, there is no accepted government job title or occupational series for software professionals. Without a way to identify who performs software functions, the following workforce and gap analysis (steps 2–4) questions cannot be answered at this time:

- How many and what types of software professionals work in DoD (e.g., software engineers, software developers, software sustainment)?
- What is the distribution of software talent across DoD (e.g., by service, active duty military versus civilian)?
- What is the retention rate among software professionals in DoD, and how does this compare with the commercial sector?
- What factors can be used to attract and retain software talent?
- What are the proficiency strengths and gaps in DoD software talent?

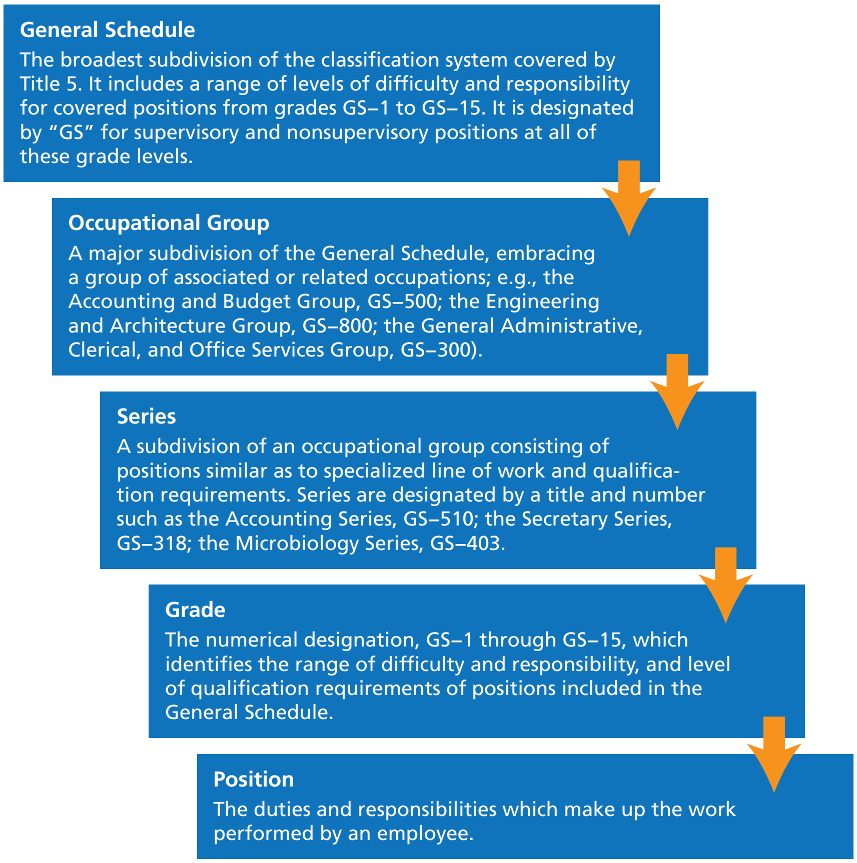
In the following sections, we present several strategies that DoD could consider to systematically identify and track the software acquisition workforce. We also discuss the advantages and disadvantages for each strategy and recommend a way ahead that balances resource requirements with desired outcomes. We begin with a brief overview of current workforce management practices used in the federal government, DoD, and the acquisition community.

Current Systems for Tracking and Managing a Workforce

Federal Government Occupational Classification

OPM provides guidance and oversight for "designing, developing, and promulgating government-wide human resources systems, programs, and policies that support the current and emerging needs of Federal agencies."[1] These policies govern how civil service employees are classified, including civilian employees working for DoD. Figure 7.2 outlines the hierarchical structure for how civil service positions are classified. As part of this structure, OPM uses a two-digit alphanumeric

Figure 7.2
Office of Personnel Management Classification Structure



General Schedule
The broadest subdivision of the classification system covered by Title 5. It includes a range of levels of difficulty and responsibility for covered positions from grades GS–1 to GS–15. It is designated by "GS" for supervisory and nonsupervisory positions at all of these grade levels.

Occupational Group
A major subdivision of the General Schedule, embracing a group of associated or related occupations; e.g., the Accounting and Budget Group, GS–500; the Engineering and Architecture Group, GS–800; the General Administrative, Clerical, and Office Services Group, GS–300).

Series
A subdivision of an occupational group consisting of positions similar as to specialized line of work and qualification requirements. Series are designated by a title and number such as the Accounting Series, GS–510; the Secretary Series, GS–318; the Microbiology Series, GS–403.

Grade
The numerical designation, GS–1 through GS–15, which identifies the range of difficulty and responsibility, and level of qualification requirements of positions included in the General Schedule.

Position
The duties and responsibilities which make up the work performed by an employee.

SOURCE: OPM, "Introduction to the Position Classification Standards," 2009.

[1] OPM, "Our Mission, Role & History," 2019.

code to designate positions in a federal civilian pay plan. In addition to the General Series (GS), OPM uses several other plans including "ST" to designate scientific and professional positions and "VN" for Department of Veterans Affairs nurses.

Many occupational series probably include some number of employees performing software functions, including the 19 different engineering series (e.g., General, Civil, Computer, Electrical, Electronic). However, there is no software engineering series. Furthermore, there are no official position titles or parenthetical titles that can be used to fully identify all software professionals. Table 7.1 lists example occupational series and corresponding position titles available within that series that may be related to software.

OPM is the authority for establishing official position titles, which must be used in official documents such as position descriptions and

Table 7.1
Possible Occupational Series with Software Professionals

Occupational Series	Series Code	Official Position Titles	Example OPM Parenthetical Titles
General Engineering	GS-0801	• N/A[a]	• N/A[a]
Computer Engineering	GS-0854	• Computer Engineer	• Data Systems • Embedded Systems • Networks • Simulations
Computer Science	GS-1550	• Computer Scientist • Supervisory Computer Scientist	• N/A
Information Technology	GS-2210	• IT Program Manager • IT Project Manager • Information Technology Specialist or IT Specialist	• Policy and Planning • Enterprise Architecture • Systems Analysis • Applications Software • Network Services • Data Management • Systems Administration

NOTE: [a] No basic titles or parenthetical specialty titles are specified for this series. Guidance indicates that agencies may construct titles that appropriately describe the work.

personnel actions.[2] However, it is important to note that federal agencies are not precluded from developing unofficial position titles, which can be used to support talent management initiatives (e.g., recruiting). Agencies may also "designate the official title of positions in occupational series for which OPM has not prescribed titles; i.e., those not specifically covered by classification standards. The title selected by the agency should not be one that has been prescribed by OPM as an official title for positions in another series."[3]

Military Occupational Classification and Skill Tracking

The military occupational classification systems are specific to each service. For example, the Air Force uses Air Force Specialty Codes, the Army uses Military Occupational Specialties, and the Navy uses Naval Ratings. Specialties are defined separately for enlisted personnel and commissioned officers. Although the services have unique position titles, DoDI 1312.01 requires the services to complete and maintain crosswalks to federal government occupational classifications (e.g., OPM classification) and civilian occupations, which allow for analyses of comparable jobs to be conducted.

The services also use a limited number of codes to track specific experiences, skills, and qualifications. The Air Force uses Special Experience Identifiers (SEIs) for experience and training not specified within the military personnel data system. According to the *Air Force Enlisted Classification Directory* (AFECD), SEIs

> are established when identifying experience or training is critical to the job and person assignment match, and no other identification is appropriate or available. SEIs permit rapid identification of a resource already experienced to meet unique circumstances, contingency requirements, or management needs. They provide a means to track individuals and identify positions requiring or providing unique experience or training that otherwise would be

[2] (5 U.S.C. 5105 (a)(2)).

[3] U.S. Code, Title 5, Section 5105, Standards for Classification of Positions. OPM, "Introduction to the Position Classification Standards," 2009, p. 14.

> lost. SEIs may be used to better distribute personnel and optimize the job and person match insofar as possible.[4]

The Air Force tracks three Agile Software Development SEIs for enlisted personnel as described in AFECD: (a) Developer, (b) Designer, and (c) Product Manager. There are also several potentially relevant experience identifiers for Air Force officers, including (1) programming—general, (2) computer programming—analysis, (3) systems analysis, (4) programming productivity techniques, and (5) systems computer program support. Officers can also be classified by a limited number of activity codes, which can help to identify officers who are directly or indirectly involved with the acquisition or technical acquisition of Air Force systems. There is also an officer activity code for computer systems, which identifies officers involved with the design, development, or application of software systems.

The Army and the Navy also track enlisted personnel and officers who have specific skills and qualifications. The Army uses an Additional Skill Identifier (ASI) to identify skills or formal school training that soldiers can take to expand their knowledge. In additional to ASIs, the Army also manages a list of Special Qualification Identifiers (SQIs), which are associated with specialty schools (e.g., Court Reporter School, Special Forces Underwater Operations School). Although it is not immediately clear if any ASIs or SQIs directly relate to the software workforce, they do provide a useful mechanism for tracking personnel skills and capabilities.

The Navy uses Tracking Naval Enlisted Classification codes to identify sailors' unique training, skills, and experiences that are not official requirements. The Navy uses a subspecialty system for officers to define advanced education, training, and experience required for a specific position and to track officers with the corresponding skills and training. One subspecialty in particular, Computer Science and System Design, identifies several core skill and educational requirements relevant to software:

[4] Headquarters Air Force Personnel Center, *Air Force Enlisted Classification Directory (AFECD)*, Randolph Air Force Base, Tex.: HQ AFPC/DPSIDC, October 31, 2018, p. 360.

- **Fundamental Computer Science**: architectures, virtualization, operating systems, computer networks, high- and low-level languages and their translation, software systems, human-computer system interaction, and supporting mathematical foundations of Computer Science
- **Software Development**: planning and development of large software projects to include specification of requirements, design, technical documentation, implementation, risk analysis, testing, quality assurance, maintenance, process metrics, and measures of effectiveness through the use of modern software engineering techniques and tools
- **Analysis**: Application of scientific methods to determine reliability, efficiency, and performance of computer systems; modeling, simulation, and analysis of algorithms, processes, and systems in support of Naval operations
- **Data Systems and Management**: Devices, interfaces, and interconnects; storage architectures and data organizations, addressing and indexing; continuity, backup, and recovery; resilience; models, analytics, and visualization; large data sets and data mining
- **Autonomous Systems**: design, construction, and operation of autonomous systems including unmanned vehicles; analysis tools for security, forensics, and intelligence; basic skills including artificial intelligence, knowledge management and representation, machine learning, heuristic search, and data mining.[5]

While the above examples demonstrate that the services have their own systems and infrastructures for tracking relevant skills and experiences, these data are often used to supplement formal manpower tracking and therefore cannot be used to fully define the software acquisition workforce. Nonetheless, these data are a useful resource for preliminary analysis and to locate areas where software functions are clearly being performed.

[5] Naval Postgraduate School, Computer Science—Curriculum 368 (Resident), Curriculum 376 (Distance Learning), Academic Catalog, 2019.

Defense Acquisition Workforce

DoD regularly collects and reports data on the acquisition workforce by linking files maintained by the Defense Manpower Data Center (DMDC).[6] These data include information about the composition (e.g., demographic characteristics, education level, and certification level) of each acquisition career field over time. However, because no acquisition career field is specific to software, the data do not provide any insights into the number, composition, or distribution of software acquisition professionals. DMDC also has some information about the specific degrees held by military and civilian personnel (e.g., computer science), which could be useful for future analyses to describe the educational backgrounds of software acquisition professionals.

Options to Track and Manage a Software Acquisition Workforce

In the following sections, we present several options that have been used by DoD and related organizations to track personnel. Each option and its key considerations are summarized in Table 7.2. In some cases, options can provide only a snapshot of the workforce suitable for short-term solutions, whereas other options may require significant long-term planning, coordination, and approval from external agencies. We consider these factors in our discussion and offer courses of action that balance resource requirements with desired outcomes in the next chapter.

Data Call

A data call requesting the services and specifically DACMs to identify personnel who perform software functions is the most direct strategy for identifying personnel who perform software functions. This approach has been used to identify personnel who perform earned value management functions that are designed to measure and monitor program performance. Although the size of the workforce performing

[6] Susan M. Gates, Brian Phillips, Michael H. Powell, Elizabeth Roth, and Joyce S. Marks, *Analyses of the Department of Defense Acquisition Workforce: Update to Methods and Results Through FY 2017*, Santa Monica, Calif.: RAND Corporation, RR-2492-OSD, 2018.

Table 7.2
Summary of Workforce Tracking Options

Option	Primary Outcome	Example Inputs	Key Considerations
Data Call	Snapshot of software workforce positions and major duties	• Criteria for eligibility as software acquisition professional • Desired data (e.g., major job functions) • Supervisor responses to data call	• Short-term solution • May require incentive to respond to data call
Coding Positions/ Billets	Information to count number and type of software acquisition positions	• Criteria for eligible positions and coding • Training supervisors to code positions	• Positions may not provide accurate information about individual qualifications or proficiency • Position duties may not reflect actual work performed
Coding skills, experiences, and education (SEE)	Information about workforce readiness and capabilities	• Transcripts, degrees, and certificates • Supervisor and self-evaluations • Assignments	• Less reliable coding for subjective SEE • Resource-intensive to verify skills and quality of experiences
Unofficial Job Titles	Tailored job titles to facilitate tracking and recruiting	• New job titles and corresponding duties • Supervisor mapping of existing positions to new job titles	• Limited structure for workforce planning and management (e.g., compensation, training)
New Acquisition Career Field(s)	Software-specific career field(s) for acquisition coded positions	• HCI coordination with acquisition programs and DAU • New DAU course curriculum and certification criteria • Supervisor mapping of existing positions to new jobs	• USD (A&S) and HCI can determine the level of training and education required • No information on software professionals outside of acquisition
New Occupational Series	Government-wide implementation of software workforce strategic plan	• Many inputs required by OPM	• Requires long-term commitment and data

earned value management is arguably considerably smaller than that of software acquisition, similar principles apply. That is, a detailed definition is provided to supervisors and/or employees, who then indicate if they perform those functions.

Data calls can be limited in several ways. First, a data call provides a snapshot of the workforce at a specific point in time. Hence, additional data calls would be required to capture changes in workforce structure and composition. Second, the success of the data call is dependent on the response rates across the services. A low response rate may limit the value gained from a data call especially if a primary objective is to identify all personnel who perform software functions. A third limitation is that the accuracy of the data may be difficult to verify particularly in cases of underreporting. Follow-on surveys may also be needed to ensure that identified personnel meet the criteria to be included in the software acquisition workforce.

Flagging Positions/Billets (Coding Positions as Software)

Position or billet codes are commonly used to track employees in specific subspecialty areas. For example, OPM recently developed the Cybersecurity Category/Specialty Area Code to identify cybersecurity positions government-wide. Similar to the software acquisition workforce, "the cyber workforce is occupationally cross-cutting, multi-faceted, and encompasses a variety of contexts, roles, and occupations."[7] In consideration of these characteristics, OPM established the cybersecurity code to allow agencies to "more effectively identify the cybersecurity workforce, determine baseline capabilities, examine hiring trends, identify skill gaps, and more effectively recruit, hire, train, develop and retain an effective cybersecurity workforce."[8]

DoD could pursue a similar strategy to define and code software positions across the services. Coding software positions would provide the same benefits and help to establish the data required to determine if an official OPM position title is needed government-wide. At a mini-

[7] OPM, *Interpretative Guidance for Cybersecurity Positions: Attracting, Hiring and Retaining a Federal Cybersecurity Workforce*, October 11, 2018, p. 21.

[8] OPM, 2018, p. 19.

mum, DoD could conduct the analyses to address critical questions about the software workforce and subsequently use these analyses to determine how best to manage this workforce.

Skills, Experiences, and Education Identifiers

The services already track a variety of skills, experiences, and education (SEE). Adding a series of codes to identify software skills and qualifications could be useful to identify who performs software functions and to uncover potential skill gaps. Positions/billets could also be coded using the same SEE to ensure that assignments are filled with qualified personnel. To achieve these benefits, DoD would need to identify a limited number of SEE that are important to effective job performance. If there are too many SEE to easily track, the codes may become unreliable and lose utility. Another challenge in effectively using SEE is to ensure that there are clear criteria that specify when someone should be coded with SEE. Some SEE will be fairly simple to evaluate (e.g., completed DoD course), but other codes may be more subjective (e.g., participated in Agile software development). To the extent possible, SEE should be objective, verifiable, and tied to important outcomes.

Unofficial Job Titles

The flexibility to use unofficial job titles, per OPM, may provide DoD with another option for tracking software professionals. Job titles could also be tailored to increase probability that job search results return relevant job opportunities and to better coincide with applicant expectations. These benefits assume that DoD agencies coordinate to decide on which job titles to use, identify the associated roles and responsibilities, and code existing and future positions. As with other options (e.g., skill identifiers), retroactively changing job titles for current employees would require an extensive effort to coordinate and then update the various systems of record.

New Acquisition Career Field

Directly under the control of USD (A&S), HCI could coordinate the development of one or more acquisition career fields specific to software acquisition.[9] Creating a new acquisition career field would reinforce communications about the importance of software and would require substantially less time and effort compared with creating a new OPM occupational series (discussed next). Furthermore, a new acquisition career field would provide a focal point for talent management efforts including recruiting, training, and certification, which are tailored specifically to meet career field and acquisition program needs.

There are a few drawbacks with this option. First, a new acquisition career field would be limited to those who are currently coded as an acquisition professional (e.g., greater than 50 percent of duties). Therefore, software professionals who have their primary duties as nonacquisition could not be tracked using this approach. The implications of this limitation will be unknown until data that can determine how software professionals are distributed across DoD are collected.

New Office of Personnel Management Occupational Series

Estimates provided by the Bureau of Labor Statistics (BLS) indicate that there were over 1.25 million software developer jobs in the United States in 2016, with a growth projection of 24 percent through 2026.[10] These positions include two broad types of software developers: those for applications and those for systems software. Although tracked by BLS, these occupations are not part of the federal government occupational series; therefore, software professionals in DoD cannot be easily identified. A new occupational series would provide similar capabilities to track military and civilian (federal government) software professionals. If the occupational series aligned with the BLS classification system, comparisons could be made in total compensation and job growth between DoD, the federal government, and the commercial sector. These types of comparisons provide critical data for understanding factors and policies that could influence recruitment and retention

[9] As suggested by the Executive Secretary of the IT FIPT, DoD could also consider redesigning the IT career field to formally incorporate software acquisition—a suggestion, which follows from the original designation of the IT acquisition career field as Software Acquisition Management.

[10] Bureau of Labor Statistics, U.S. Department of Labor, "Software Developers," *Occupational Outlook Handbook*, April 13, 2018.

of skilled personnel. An occupational series would also provide structure for developing career paths that could support transitions between DoD and other agencies.

The primary disadvantage to this approach is the limited amount of government-wide data on software professionals needed to establish a new occupational series. OPM specifies the criteria needed to fully consider the creation of a new occupational series. The specific details of these criteria were provided in email response by OPM:

> All OPM GS Position Classification Standards (PCS) are consistent with the grade-level definitions of work established by law. These definitions are based on the difficulty and responsibility of the work at each level and the qualifications required to do that work. All occupations change over time, some more rapidly and profoundly than others, but the fundamental duty and responsibility patterns and qualifications required in an occupation normally remain stable. Therefore, careful application of the appropriate PCS to work performed should yield the correct grade for a position irrespective of date written. Any duties not specifically referenced in a PCS can be evaluated properly by comparison with similar or related duties the PCS does describe, as well as with the entire pattern of grade-level characteristics.
>
> Agencies may submit requests for updates to standards through their Chief Human Capital Officers for OPM to consider establishing new or revising existing standards at any time. The formal request must come through an agency's Headquarters Human Resources Office. Agency requests for new standards and revisions must include the following basic information in order to be properly considered:
>
> a. The current classification of the covered positions;
> b. The positions duties and responsibilities;
> c. Employment data—the number of impacted positions and their current classification (e.g., pay plan, title, occupational series, and grade);
> d. Number of employees working in specialty areas and/or mixed jobs;
> e. Organizational charts clearly identifying positions;

f. Explanation of why the current classification/qualification standard(s) is/are not effective;
g. Supporting documentation of classification difficulties;
h. Required competencies and/or knowledge, skills, and abilities required for performing work;
i. Job analysis supporting required competencies and/or knowledge, skills and abilities required;
j. Copies of current official position descriptions and other related position classification information;
k. Copies of current performance standards for the work;
l. Copies of agency and OPM appeal decisions for the work;
m. Copies of job opportunity announcements used to fill positions;
n. Statistical data/information showing current recruitment efforts and challenges filling positions;
o. Statistical data/information showing current retention and turnover issues;
p. Information concerning use of HR Flexibilities such as 3Rs [recruitment, relocation, and retention incentives] and Special Salary Rates;
q. Related agency studies/surveys of positions;
r. Any other pertinent information related to this work at your agency; and
s. Government-wide impact (e.g., any other agencies likely to have covered positions).[11]

A review of these criteria suggests that developing a request to create a new occupational series would require a long-term strategy to coordinate across agencies and DoD to collect the required data. Furthermore, coordinating and securing OPM approval may require a multiyear commitment as demonstrated by resources and time dedicated to the recent creation of the cybersecurity occupational series.

[11] OPM, email exchange on requirements to request a new occupational series, October 18, 2018.

CHAPTER EIGHT

Recommendations

There are two fundamental recommendations that DoD should immediately follow to address potential concerns with the software acquisition workforce. First, DoD needs to adopt a strategy to identify who is in the software acquisition workforce. Second, steps should be taken to validate the software acquisition competencies presented in this report. We elaborate in this chapter on each of these recommendations.

Recommendation 1: Identify Who Is in the Software Acquisition Workforce

As described in Chapter Seven, the options for tracking and managing a software acquisition workforce may require different levels of resources and offer different outcomes. Options range from short-term solutions such as asking supervisors to identify who performs software acquisition functions (data call) to modifications of existing occupational classification structures (i.e., occupational series, acquisition career field). Midrange solutions (e.g., coding positions or individual SEE) can capture desired data without changing occupational structures, but still require considerable resources to develop the criteria for coding and training individuals to carry out the coding.

Considering the resources required and desired outcomes to begin tracking the software acquisition workforce, *we recommend initiating a data call that would identify personnel who perform software functions.* Other options either require greater amount of resources and coordination or do not directly address who performs software functions.

More specifically, identifying new career fields either across the federal government or within the DoD acquisition community requires not only a baseline knowledge of who performs software functions but also updating other talent management components (e.g., hiring criteria, training, and education). Such an option should be pursued only once a clear need has been identified. A less resource-intensive option would be to code positions or billets. However, this option does not provide information on the qualifications or competencies of individuals filling those positions; instead, it provides only a better indication of the demand for software professionals. Understanding the demand for software professionals is an important part of a workforce planning strategy but does not directly address the question of who performs software functions and what their competencies are.

Considering the different options, we recommend that a data call should be adopted as an initial step that can provide the information necessary to determine if more permanent tracking mechanisms are needed. For example, if DoD determines that there is sufficient justification to submit a formal request to OPM for a new occupational series, the results of a data call will be critical to providing information on how many personnel will be impacted and the current classification of those personnel to include current title, pay, and grade.

We recommend limiting the data call initially to personnel within the acquisition community to promote rapid implementation of competencies across acquisition career fields. DoD could use the data call results to determine the need to expand tracking efforts beyond the acquisition community. Finally, the data call results should be used to guide discussions on the need and level of effort required for more formal tracking mechanisms (e.g., do the data indicate a need to develop a software subspecialty or career field?).

Identifying who performs software functions may at first appear simple but requires significant time, resources, and coordination. To ensure accurate and reliable data on the workforce, DoD should appoint a senior leader with the authority to direct data collection efforts. Such authority is necessary to ensure the cooperation and timely responses from civilian and military personnel, including active-duty, guard, and reserve components. The time and resources needed to plan and execute these data collection efforts will require coordination and support across services, programs, and functional career fields. Without full support from key stakeholders, any effort to identify the workforce will be delayed and likely incomplete.

Recommendation 2: Validate the Software Acquisition Competencies

After the software acquisition workforce has been identified, the competencies should be validated. At a minimum, DoD needs to collect information from the workforce to evaluate the relative importance of each competency. This step will require coordination with DCPAS to determine the most appropriate way forward given limitations of DCAT. As described in Chapter Two, DCAT is the official tool for measuring proficiency gaps and proficiency levels in the DoD civilian workforce. However, due to technical requirements coded into the software, DCAT is limited in the number of competencies that can feasibly be fielded at any one time.

Options for Validating Software Acquisition Competencies

Given the current challenges associated with DCAT, we reviewed alternative approaches to gather the necessary data for validation. We considered four options.

1. Aggregate similar competencies to reduce the number of competencies to 12 or fewer (initially recommended by DCPAS). However, this option loses valuable information and specificity that are needed to guide training and education.
2. Update the DCAT software (or use different software) by eliminating the requirement for a match and restructuring the design. For example, a more sophisticated branching could be used in which respondents start by answering one primary question about each competency (i.e., relative importance) and then are provided only with follow-on questions for competencies meeting some threshold in importance (e.g., critical competencies).

Of course, this approach assumes that the branching would help to reduce the number of competencies that a supervisor and employee would need to evaluate.

3. Administer a stratified random sample of competencies to potential respondents. For example, one respondent may evaluate competencies #1 through #12, and another respondent may evaluate competencies #6 through #18. If a sufficient number of responses are received, accurate estimates of the relative importance of competencies and of competency gaps could be made. This approach would require a complex sampling plan to ensure representative responses are received from subgroups (e.g., career fields) and that supervisors and their employees received the same set of competencies.
4. Group competencies into meaningful categories that align with software positions, career paths, or functions. This approach would allow for DCAT to be administered in a much more targeted way using fewer and potentially more relevant competencies for each respondent. However, to implement this option effectively, the workforce would need to be defined and organized into meaningful groups (see Chapter Seven). As an example, we offer a notional grouping of software acquisition competencies based on our professional expertise and observations of the commercial industry in Appendix D.

Balancing Needs and Resource Requirements

Noting current limitations with DCAT and the need to validate competencies, *we recommend either reprogramming DCAT or selecting another software tool. Most importantly, we recommend limiting the number of questions to focus on relative importance of competencies.* Reducing the number of questions will help to minimize "survey fatigue" and facilitate higher response rates. Future analyses that should include assessments of proficiency and workforce competency gaps could then focus on a more limited set of the most critical competencies relevant to targeted software professionals. If DCAT cannot be reprogrammed, the competencies could be administered in chunks using stratified random sampling such that each respondent "sees" only 10 to 12 competencies.

We further recommend consulting a statistician to ensure that the sample of respondents are representative of important perspectives (e.g., service, years of experience, acquisition category (ACAT) level.[1] A well-designed sampling plan is needed so that appropriate statistical analyses can be conducted to address critical questions about the workforce. These questions may include the following:

- How does the importance of competencies vary across different job groups?
  - occupational series
  - acquisition career fields
  - acquisition programs
- Which competencies will be the most important in the future?
- How well do supervisors and job incumbents agree on which competencies are most and least important?
- How does the importance of competencies vary by job grade and acquisition certification level?
- Which competencies are needed on day 1?

Finally, *we recommend planning future validation studies that establish links between performance on competencies and outcome measures*. For example, demonstrating that higher proficiency in a set of competencies is associated with better performance (e.g., fewer errors, faster delivery) provides some evidence on the relative importance of these competencies. These types of criterion-related validation studies can take multiple years and considerable resources to plan and execute. Because appropriate performance measures may not be readily available, they would first need to be developed and evaluated to ensure that the measures reflect true performance. Considering these factors, validation studies should be integrated into a long-term strategy for evaluating and managing the software acquisition workforce.

[1] ACAT is an acquisition category that is based on level of funding provided to a program. For a description of ACAT categories and criteria, see DAU, *Defense Acquisition Guidebook*, undated, Chapter 1, Section 3.2.3.1.

Conclusion

This report presents the methodology for the development of competencies that could be used to train, develop, and manage a software acquisition workforce. We compare these competencies with the training courses provided by DAU to identify potential gaps. Both of these efforts should be viewed as first steps in a long-term strategy to define and manage a software acquisition workforce. Further analysis is required to validate the competencies and to determine who is performing software functions. To gain complete traction on this problem, DoD needs to appoint a senior leader who can implement these recommendations across the services. Without a champion, any improvements will be slow and sporadic.

APPENDIX A

Acquisition Career Fields

Table A.1 lists each of the acquisition career fields and corresponding example activities and duties that are performed. The example activities for each career field were extracted from the Defense Acquisition University website.

Table A.1
Acquisition Career Fields

Acquisition Career Field	Representative Assignments and Activities
Auditing	• Audits financial records, reports, management controls, policies, and practices affecting or reflecting the financial condition and operation of DoD and other federal agency contractors
Business—Cost Estimating	• Relates the processes of life-cycle cost estimating within the context of materiel system acquisition in DoD
Business—Financial Management	• Applies basic concepts of budget and program principles, policies, procedures, concepts, standards, and terminology, as well as a general knowledge of the financial management and business operation systems • Possesses a basic knowledge of acquisition; recognizes the life-cycle process of an acquisition program • Reviews, allocates, or manages acquisition resources and programs
Contracting	• Operational Contracting: contracting functions in support of post, camp, or stations • Research and Development: contracting functions in support of research and development • System Acquisition: contracting functions in support of systems acquisition to include all ACAT programs • Logistics and Sustainment: contracting functions performed by the Defense Logistics Agency or by other offices to sustain weapon systems

Table A.1—Continued

Acquisition Career Field	Representative Assignments and Activities
ENG	• Functional Engineer: 1. plans, organizes, conducts, and/or monitors engineering activities relating to the design, development, fabrication, installation, modification, sustainment, and/or analysis of systems or systems components for a functional specialty (i.e., reliability and maintainability, systems safety, materials, avionics, structures, propulsion, chemical/biological, human systems interfaces, weapons, Computer Engineer/Scientist, etc.); 2. demonstrates how systems engineering technical processes and technical management processes guide engineering activities for a functional specialty. • General Engineer: 1. plans, organizes, conducts, and/or monitors engineering design, development, and sustainment activities for systems or systems components; 2. demonstrates how systems engineering technical processes and technical management processes guide design, development, and sustainment activities
Facilities Engineering (FE)	• Conducts actions that support one or more facet of FE; planning; design; construction; environmental management; base operations, support, and housing; real estate; and real property maintenance • May serve as an IPT member, representing a specific FE functional area
Industrial and Contract Property Management	• Oversees and manages life-cycle processes for government-owned property being utilized by contractors (i.e., government property in the possession of contractors and, in some instances, government-owned contractor-operated plants) • Provides advice and assistance on property-related matters during acquisition planning, contract formation, and contract management • Reviews contractor's purchasing system as it pertains to property management
IT	• Central Design Activity: identifies and describes basic concepts of software engineering and development activities; EA; best practices; IT systems engineering; information assurance/cybersecurity; IT-related technologies; test and evaluation processes; and verification and validation processes • Project Office/Field Activities: Identifies and describes the following: IT program management approaches; emerging IT acquisition strategies; best practices; IT-related performance measures and quality management; acquisition planning, solicitation, and administration; information assurance/cybersecurity; test and evaluation processes; verification and validation processes; and fielding and sustaining IT systems

Table A.1—Continued

Acquisition Career Field	Representative Assignments and Activities
Life-Cycle Logistics	• Design Interface: understand and support the systems engineering process to impact the design from its inception throughout the life cycle, facilitating supportability to maximize the availability, effectiveness, and capability of the system at the lowest total ownership cost • Sustaining ENG: understand, recognize the importance of, and assist in supporting in-service systems in their operational environments • Technical Data: support the identification of, planning for, resourcing, and implementation of management actions to facilitate development and acquisition of information to operate, install, maintain, and train on the equipment to maximize its effectiveness and availability; effectively catalog and acquire spare/repair parts, support equipment, and all classes of supply; define the configuration baseline of the system (hardware and software) to effectively support the warfighter with the best capability at the time it is needed
Production, Quality, and Manufacturing	• Engineer: 1. establishes production planning and control processes and measures the overall effectiveness of the organization, methods, systems, and procedures; 2. builds producibility into designs (tooling, facilities, and products); 3. builds quality characteristics into the designs of products and services; 4. builds quality requirements into technical review criteria and program planning
PM	• Weapon Systems: 1. participates in an IPT delivering a weapon system, C2/network-centric system, or space system; 2. performs financial and status reporting and basic logistic activities; 3. supports pre-award contract activities and workload planning and scheduling • Services: assists in acquisition planning, assessing risk (technical, cost, and schedule), and contract tracking and performance evaluation • Business Management Systems/IT: participates in a business process IPT, fundamentals of enterprise integration, and outcome-based performance measures
Purchasing	• Purchases, rents, or leases supplies, services, and equipment through either simplified acquisition procedures or placement of orders against preestablished contractual instruments to support operational requirements
Science and Technology Management	• Conducts and/or monitors science and technology activities including basic research, applied research, and/or advanced technology development to support acquisition programs

Table A.1—Continued

Acquisition Career Field	Representative Assignments and Activities
Small Business	• Not available on DAU website
Test and Evaluation	• Program Management and Matrix Support: 1. supports the program's T&E working-level IPT; 2. supports development of program's T&E strategy, approach, process, schedule, and resource requirements; 3. supports coordination of cybersecurity T&E IAW DoDI 5000.02 and the DoD Risk Management Framework; 4. supports implementation of an evaluation methodology and framework for product/system under test; 5. supports development of T&E materials/data for technical and progress reviews, to include risk assessment • Range/Lab/Field Supporting Activities: 1. supports identification and scheduling of T&E resources to include workforce, infrastructure, and budgets to support testing at the respective facility; 2. reviews facility T&E tools (IT, video, targets, simulators, stimulators, instrumentation, etc.) and clearly understands their capabilities; 3. supports facility test plan development; 4. supports development of T&E plans and mitigation of safety risks for test plans during test execution; 5. assists in test execution, data collection, analysis, and reporting

APPENDIX B

Trends in Modern Software Development Trends

In this appendix, we provide a detailed summary of industry trends and modern software practices. Because software development practices often outpace traditional peer-reviewed research, our review combines information from several sources including peer-reviewed studies, gray literature (e.g., think tanks, research institutes, DoD government documents), and professional literature (e.g., blog posts, commercial industry white papers).

We have already identified four general, interrelated trends in Chapter Four. These concern changes in the sequencing of the activities used in the production of software, described as an SDLC model; changes in software development architecture from monolithic development to ecosystems; increasing diversity in software deployment architectures; and increasing automation in the practice of software development. Here, we elaborate further on these trends.

Life-Cycle Trends: Waterfall to Agile

Software development practice is often described in relation to an SDLC model that indicates how the practices necessary to produce working software are orchestrated in time. The most commonly referenced SDLCs are *Waterfall* and *Agile*, with almost an infinite variety of life cycles in actual use. Although we describe each SDLC in its idealized form in the sections below, one of the authors of this report has over 40 years in software development and cautions that she has never seen either idealized model used on any project. What is true is that the trend in software over those 40 years has been to shorten the time between idea and working product in an attempt to improve software quality and maximize the potential of software to adapt to changing environments.

Waterfall Software Development Lifecycle

The Waterfall SDLC is an idealized approach largely modeled after hardware development. In this model, development stages are distinct, do not overlap, and happen in sequential order.[1] Progress is one-directional. For a stage to begin, the prior stage must be completed, and once a stage has been completed, it is not revisited.[2] In a Waterfall development model, software is conceived as progressing through distinct stages; the number of stages is not important, but the distinct nature of the stages and the association of each stage with a specific activity/practice are. The literature suggests seven stages: conception, initiation, requirements and analysis, design, implementation (or code), testing, and maintenance (see Figure C.1). Testing, per older DoD software acquisition compliance documents such as MIL-STD-498,[3] is further broken down into unit test, integration test, and final qualification and/or acceptance test.[4] Similarly, the requirements and analysis activity is often broken down hierarchically by system, subsystems, and units to mimic hardware development nomenclature. The Waterfall activities are often depicted in a "V" configuration, with validation occurring on the left branch of the V and verification activities

[1] S. Balaji and M. Sundararajan Murugaiyan, "Waterfall vs. V-Model vs. Agile: A Comparative Study on SDLC," *International Journal of Information Technology and Business Management*, 2012, pp. 26–30; Nayan B. Ruparelia, "Software Development Lifecycle Models," *ACM SIGSOFT Software Engineering Notes*, Vol. 35, No. 3, May 2010, pp. 8–13.

[2] Association of Modern Technologies Professionals, "Software Development Methodologies," 2018; Smartsheet, "What's the Difference? Agile vs Scrum vs Waterfall vs Kanban," 2018.

[3] DoD Military Standard 498, *Software Development and Documentation*, December 5, 1994.

[4] DoD Software Development Plan templates still use this nomenclature to refer to the various levels of testing that software products complete. See, for example, Berton Manning, "Software Development: Software Management Plan," *AcqNotes*, June 15, 2018.

occurring on the right branch.[5] This system engineering V depiction is particularly useful in emphasizing the importance of validation and verification for safety/security critical or high-availability systems.

The trend to automated pipelines has blurred many of the formerly sharp delineations between the activities of the Waterfall model. When everything from code onward has been automated in a set of tooling, the model is less useful as a means for thinking about the organization of the software development process.

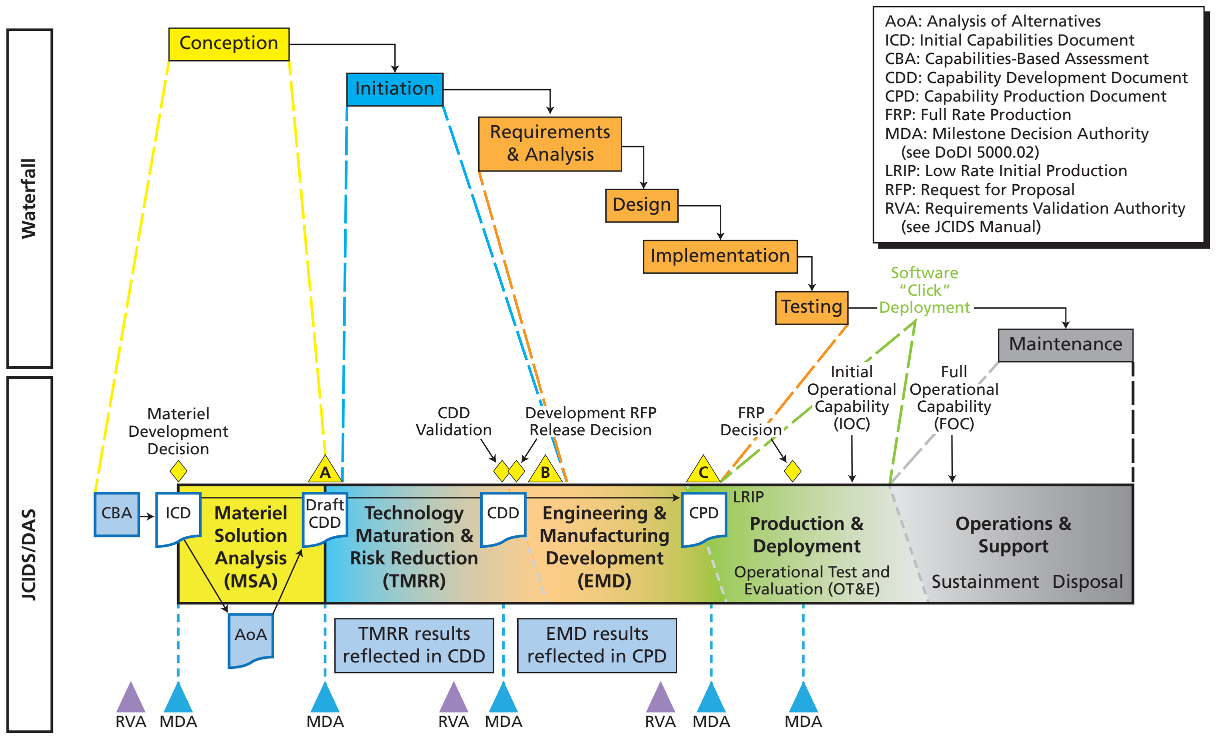
The delineations between early activities of the Waterfall SDLC are also becoming increasingly blurred. Modern software architectural concepts bridge the gap between requirements, design, and implementation in ways that hardware architecture does not. Software architectures are largely abstract and are expressed in multiple dimensions (static versus dynamic, development versus deployment, user capabilities versus quality attributes) that interconnect the requirements, design, and implementation activities.

With those caveats in mind, we offer a brief description of the Waterfall stages as derived from the literature and, to provide perspective, compare the software Waterfall with the Joint Capabilities Integration and Development System (JCIDS)/Defense Acquisition System (DAS) stages used in DoD system development.[6] We caution, however, that our mapping of the software Waterfall to JCIDS/DAS stages in Figure B.1 is highly simplified. The astute reader will notice that although both life-cycle models share the characteristic of being linearly staged, the maturity of software as it passes through its early gates is lower than that required in the JCIDS/DAS and that it passes

[5] For an example of the "V" model, see G. K. Hanssen, B. Haugset, T. Stålhane, T. Myklebust, and I. Kulbrandstad, "Quality Assurance in Scrum Applied to Safety Critical Software," in H. Sharp and T. Hall, eds., *Agile Processes, in Software Engineering, and Extreme Programming. XP 2016.* Lecture Notes in Business Information Processing, Vol. 251, Cham.: Springer, 2016.

[6] Much of the literature on the Waterfall SDLC may be biased in a desire to contrast it unfavorably with Agile SDLCs and emphasizes the rigidity of the Waterfall. However, the experience of SMEs we contacted is that the rigidity of the Waterfall is greatly exaggerated, and what matters is the time span between phases.

Figure B.1
Waterfall Model Activity Flow Compared to Joint Capabilities Integration and Development System/ Defense Acquisition System


Waterfall
JCIDS/DAS
Conception
Initiation
Requirements & Analysis
Design
Implementation
Testing
Software "Click" Deployment
Maintenance
AoA: Analysis of Alternatives
ICD: Initial Capabilities Document
CBA: Capabilities-Based Assessment
CDD: Capability Development Document
CPD: Capability Production Document
FRP: Full Rate Production
MDA: Milestone Decision Authority (see DoDI 5000.02)
LRIP: Low Rate Initial Production
RFP: Request for Proposal
RVA: Requirements Validation Authority (see JCIDS Manual)
Materiel Development Decision
CDD Validation
Development RFP Release Decision
FRP Decision
Initial Operational Capability (IOC)
Full Operational Capability (FOC)
A
B
C
CBA
ICD
Materiel Solution Analysis (MSA)
Draft CDD
Technology Maturation & Risk Reduction (TMRR)
CDD
Engineering & Manufacturing Development (EMD)
CPD
LRIP
Production & Deployment
Operational Test and Evaluation (OT&E)
Operations & Support
Sustainment
Disposal
AoA
TMRR results reflected in CDD
EMD results reflected in CPD
RVA
MDA
MDA
RVA
MDA
RVA
MDA
MDA

SOURCE: DAU, "Figure 3: JCIDS and Defense Acquisition," *Defense Acquisition Guidebook*, undated.

through its later gates with a maturity much higher than that required in the JCIDS/DAS.[7]

During the conception stage, a rough assessment of the project is produced. This includes an assessment of why the project would be beneficial, as well as general goals and scoping of the project. Ideally, an initial cost estimate and rough timeline are also generated.[8] The conception stage ends, and the initiation stage begins with management approval of the basic concept. In DoD acquisitions, this is often signaled by the release of a draft Capabilities Development Document.

Once the project has been approved, the initiation stage begins. During this stage, the project team is hired, and a more detailed project plan is developed. This also includes the clear defining of project scope, objectives, deliverables, and timeline. For DoD programs, this stage is associated with developing an RFP and performing source selection activities.

The requirements and analysis stage may mark the first formal meeting between the project team and the customer/stakeholders. During this stage, needs are identified and a requirements specification document is developed, identifying requirements for each project goal.[9] In some cases, system requirements (i.e., components needed for building the system, including both hardware and software) and software requirements (i.e., the expected level of functionality for the software being developed) are established separately.[10] The aggregated requirements—framed within the initial conception of the software—are then analyzed to determine project feasibility.[11]

[7] In fact, given the highly automated nature of software build, test, and deployment today, software passes from a JCIDS/DAS "Gate C" maturity to an initial operational capability with one click of a button.

[8] Smartsheet, 2018.

[9] Mark Lotz, "Waterfall vs. Agile: Which Is the Right Development Methodology for Your Project?," *Segue Technologies*, July 5, 2013.

[10] Nabil Mohammed Ali Munassar and A. Govardhan, "A Comparison Between Five Models of Software Engineering," *International Journal of Computer Science Issues*, Vol. 7, No. 5, September 2010, pp. 94–101.

[11] Note that in DoD JCIDS, the feasibility of the program was decided before the project team was formed and before requirements analysis, but in the software Waterfall, the feasibility decision comes after.

Once requirements have been agreed upon and the project has been deemed feasible, the design stage begins. Based on the requirement specification documents created in the prior phase, the project team develops design specifications for both the architectural design (i.e., determines the software framework by defining the major components and their interactions, but not the actual structure of each component), as well as the detailed design (i.e., defines how each component is implemented). These are then converted into models and prototypes, which are evaluated, and subsequently a design is finalized.[12]

The implementation stage uses the documents, models, and evaluations generated over the past four stages to write the code that implements the software. After coding is complete, the testing stages begin. The newly developed software is tested for bugs and defects utilizing a wide range of test tools at various levels of software integration; and user-acceptance tests are conducted, ensuring that the software can execute tasks from real-world scenarios by the users for whom the software was developed. After ample testing has been conducted and all necessary fixes have been made, the final product is released to the customer. The trend toward automation in software development has largely collapsed these implementation and test stages, and there is no significant period of time between when designs are finalized and entry into the maintenance stage.

The final stage—maintenance—is meant to address any issues that may arise from future use of the software. This includes any necessary product updates or patches required for changing needs or shifting environments, and fixes for defects that were not uncovered during the testing stage (or that arose as a result of updates).[13]

It is important to note that documentation is critical when utilizing a Waterfall SDLC on a large program. Because generating and

[12] The need to do a complete validation of the proposed design prior to moving into implementation is based on the fact that it is very expensive to make late changes in a Waterfall model. Agile life-cycle models directly acknowledge and accommodate the inherent difficulty of correctly analyzing and predicting the behavior of complex software and systems. Often, we fully understand which elements are important for modeling only after at least part of the software is built.

[13] Balaji and Sundararajan Murugaiyan, 2012, pp. 26–30.

analyzing requirements may occur years before those requirements are implemented or tested, clarity, correctness, and completeness of documentation are critical to program success. This is especially necessary for large projects with significant personnel turnover over the course of development.

Waterfall Development Teams

A software development team that utilizes the Waterfall approach is often composed of smaller subteams, with each subteam mapped to specific activity-based stages. A common practice is to have a software system engineering team assigned to requirements, analysis, and architectural design; a software development team to take over for detailed design, code, and developer testing; and an independent verification and test team to take over in the later phases leading up to customer delivery. Yet another team will often be assigned to long-term maintenance and sustainment of the software. Handoffs between teams are often formalized in an attempt to limit ambiguity.[14] In DoD programs, it is not uncommon for these teams to work under separate contracts, thereby making the handoffs contractual interfaces.

Waterfall Software Development Life Cycle in Practice

Although the idealized Waterfall has all software capabilities developed in one pass through the model, it has long been recognized that there is value in building software incrementally. In complex systems that have a significant number of dependencies among hardware, software, and external elements, not all requirements and designs mature at the same time. Waiting for "everything" to be set in stone is impractical and unnecessary if the software development team does not have the bandwidth to work on "everything" simultaneously. A variant of the Waterfall, called "Incremental Build," in which detailed design, code, and developer testing are repeated for distinct capability subsets, was the dominant DoD software acquisition life-cycle model in the 1990s and well into the 2000s.

---

[14] Ming Huo, J. Verner, Liming Zhu, and M. A. Baber, "Software Quality and Agile Methods," *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004.

Agile Software Development Life Cycles

The term "Agile" in the context of software development is derived from the 2001 "Manifesto for Agile Software Development"—a short document signed by 17 leaders in the software industry who outlined four values and 12 core principles that they believed were needed if software development practice was to improve.[15] The four values are stated as a need to emphasize:

1. *individuals and interactions* over processes and tools
2. *working software* over comprehensive documentation
3. *customer collaboration* over contract negotiation
4. *responding to change* over following a plan.

Scott Ambler, one of today's leading writers on software development practice, explains these values as: tools and processes are important, but a competent, effective team is more important; comprehensive documentation helps users understand the software's build, but the main point of development is to develop useful software; a contract is important, but cannot replace a close working relationship with customers to discover their needs; and, a strategic plan is important, but should be able to accommodate any changes in customers' priorities, their understanding of the problem, or any changes in the environment (i.e., technological advances).[16]

The 12 core principles of the *Agile Manifesto* reflect these values.

1. 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.

[15] Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, and Brian Marick, "Manifesto for Agile Software Development," 2001.

[16] Scott Ambler, "Examining the Agile Manifesto," Ambysoft, 2014.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople 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.[17]

"Agile," then, began as a philosophy, not a development model or set of practices. Dave Thomas, one of the "Manifesto's" signatories, has observed that "the word 'agile' has been subverted to the point where it is effectively meaningless, and what passes for an agile community seems to be largely an arena for consultants and vendors to hawk services and products."[18] Keeping this caution in mind, we adopted the word "modern" to describe the software competencies needed for DoD acquisitions and distinguish an SDLC model from the practices used to support it.

[17] Beck et al., 2001.

[18] From "Agile Is Dead (Long Live Agility)," quoted in Yvette Francino, "Is the Agile Manifesto Dead? Not by a Longshot," *Tech Beacon*, undated.

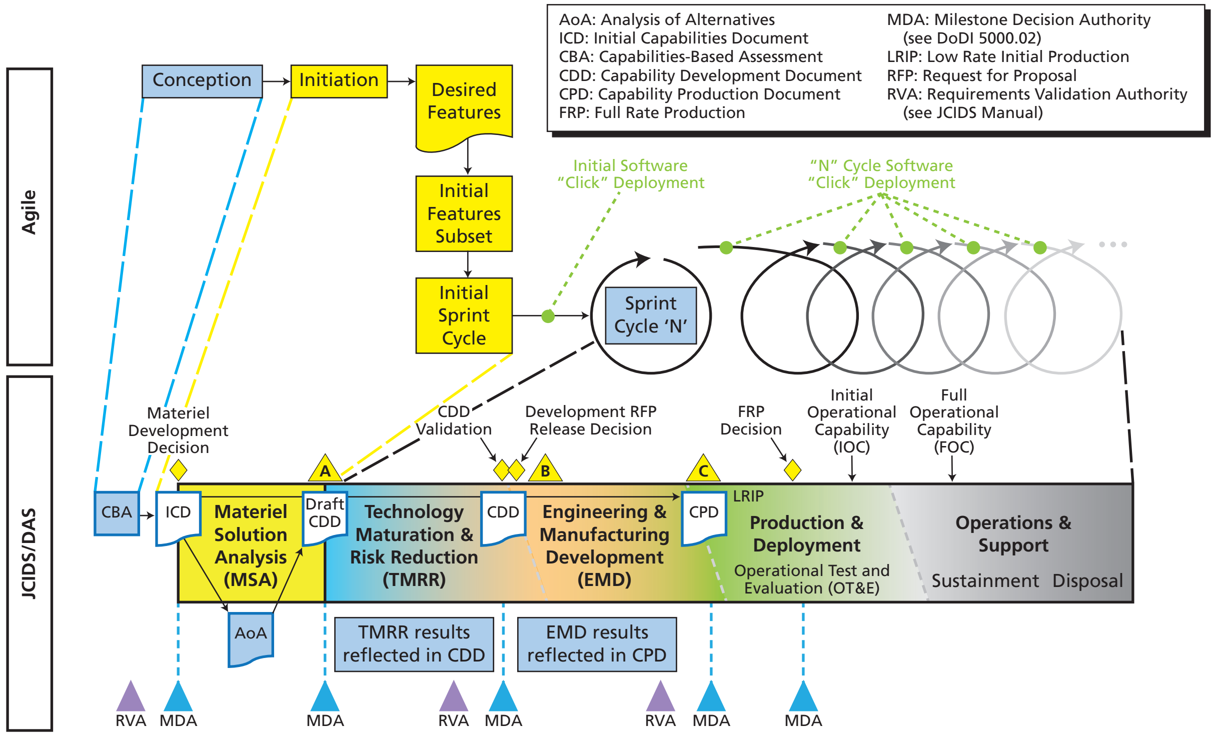
There are many "Agile" SDLC models in the literature. Our description below is derived primarily from Scott Ambler's "High Level Agile" SDLC and reflects what is often referred to as a "DevOps" or "SecDevOps" process in which each iteration ends with working software deployed to operations. We note significant variations in our description.[19]

The initial stages of conception and initiation are much the same as the Waterfall model, but after that, development is partitioned into a series of "sprints," each of which results in "working software" for a set of "features" (see Figure B.2). The partitioning of capabilities for Iterative development is common to all Agile SDLCs. In some Agile SDLCs, the subsets of software developed in each iteration are defined by "user stories," in others by the test cases that must be passed. There is significant variation in the duration of the sprint, with some models using fixed "time-boxed" durations and others using what is called "continuous" iteration. For small projects with few dependencies, sprints are often measured in days or weeks. More complex projects use a longer period, but there is a shared belief among Agile practitioners that shorter sprints improve software quality. Another major element of variation in Agile SDLCs is whether delivery and deployment are included in the iteration. Often, the software is taken through integration only within the sprint, with delivery and deployment performed on "releases" of software that aggregate the results of several sprints.

Although the conception and initiation phases of Agile—often referred to together as "discovery"—are similar to those of Waterfall, a key difference is that Agile SDLCs emphasize the early involvement of the project team. In these early phases, the project team researches the customer's goals, challenges, business climate, and end-users both independently and through face-to-face interactions with the custom-

[19] There have been various attempts to illustrate the different branches of "Agile" software methods that go by names such as Extreme Programming (XP), Lean Development (aka Kanban), Scrum, DevOps, and SecDevOps. One of the more successful attempts to map the various strains to specific variations in practice is the Agile Alliance's "Subway Map to Agile Practices," 2018. See also Scott Ambler, "The Agile System Development Life Cycle (SDLC)," Ambysoft, undated.

Figure B.2
Agile Model Process Flow Compared to Joint Capabilities Integration and Development System/Defense Acquisition System


AoA: Analysis of Alternatives
ICD: Initial Capabilities Document
CBA: Capabilities-Based Assessment
CDD: Capability Development Document
CPD: Capability Production Document
FRP: Full Rate Production
MDA: Milestone Decision Authority (see DoDI 5000.02)
LRIP: Low Rate Initial Production
RFP: Request for Proposal
RVA: Requirements Validation Authority (see JCIDS Manual)
Agile
JCIDS/DAS
Conception
Initiation
Desired Features
Initial Features Subset
Initial Sprint Cycle
Sprint Cycle 'N'
Initial Software "Click" Deployment
"N" Cycle Software "Click" Deployment
Materiel Development Decision
CDD Validation
Development RFP Release Decision
FRP Decision
Initial Operational Capability (IOC)
Full Operational Capability (FOC)
A
B
C
CBA
ICD
Materiel Solution Analysis (MSA)
Draft CDD
Technology Maturation & Risk Reduction (TMRR)
CDD
Engineering & Manufacturing Development (EMD)
CPD
LRIP
Production & Deployment
Operational Test and Evaluation (OT&E)
Operations & Support
Sustainment
Disposal
AoA
TMRR results reflected in CDD
EMD results reflected in CPD
RVA
MDA
MDA
RVA
MDA
RVA
MDA
MDA

SOURCE: DAU, undated.

er.[20] The result of these two phases is the desired capabilities/features/stories set. There is wide variation in the level of detail captured at this stage of development. Some SDLCs advocate that only the minimum set of features and capabilities (the "minimum viable product") be defined during these early phases, while others advocate for a more complete "wish list" of features and capabilities that the customer and their end-users would ideally like incorporated into the software (often referred to as the "backlog"). Both the minimum viable product definition and the complete backlog are of value, and our software competencies include the need for both.

"Iteration Zero" is often referenced separately in Agile SDLCs to acknowledge that going from zero to something is generally much harder than adding incremental functionality to existing software. The selection of the initial subset of features to implement in the initial iteration is made considering overall program risk and may often be quite small in order to allow sufficient time to build team cohesion and gain familiarity with the methods, processes, and tools to be used in subsequent sprints. Well-integrated teams with experience in the methods, processes, and tools selected for use may choose to do an early prototype of a particularly challenging or ill-defined capability during Iteration Zero.

Each "sprint cycle"—also referred to as an "iteration"—essentially contains all of the activities of the Waterfall development cycle,[21] albeit applied to a much smaller capability set and with no fixed sequencing between activities (a key distinguishing characteristic between the Waterfall and Agile/"modern" SDLCs).[22] Architectural designs are

[20] Segue Technologies, "What Is Agile Software Development?" August 24, 2015.

[21] The sequencing of activities within an iteration often does follow the traditional Waterfall, but can also be varied based on need. For example, if the goal of the iteration is to improve the timeliness or efficiency of the product, then the initial activity might be the verification tasks needed to determine the performance of the current code, followed by prototyping of alternative solutions and finally, based on evidence collected, revising the architecture and models.

[22] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta, "Agile Software Development Methods: Review and Analysis," VTT Technical Research Centre of Finland, 2002; Association of Modern Technologies Professionals, 2018.

revised; models are generated and evaluated; documents are generated; capabilities, features, and stories are coded; and software is verified.[23] Once testing is complete, the product is ideally deployed to the customer for feedback and the development team convenes to discuss lessons learned.[24] Subsequent product iterations "add on" capability or sometimes refactor it based on feedback from the customer and/or from lessons learned. In this way, an Agile SDLC allows for early discovery of emergent issues and a more flexible response.[25]

Agile Development Teams

A development team that internalizes the values and principles of the "Agile Manifesto" is often characterized as being cross-functional, self-organizing, and highly collaborative. However, these teams may choose to execute any SDLC, including the Waterfall. In the ideal environment the signatories of the "Agile Manifesto" envisioned, these teams would be small in size, physically work in the same space, and have short daily meetings (the daily stand up, or Scrum) to orient all team members to any changes that may have occurred the previous day.[26] This combination allows for improved communication and product development via increased face-to-face interactions and thus faster information flow and with lower levels of ambiguity.[27] As we will discuss later in this appendix, modern software development tooling often allows these same benefits to be realized by larger and/or geographically dispersed teams.

[23] Smartsheet, 2018.

[24] Balaji and Sundararajan Murugaiyan, 2012, pp. 26–30.

[25] An Agile SDLC does not guarantee these outcomes; it only enables them. An incompetent team that cannot solicit actionable customer feedback, assess and manage risk, or incorporate change may be better off using a more structured SDLC.

[26] Lan Cao, Kannan Mohan, Peng Xu, and Balasubramaniam Ramesh, "A Framework for Adapting Agile Development Methodologies," *European Journal of Information Systems*, 2009; Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu, "Can Distributed Software Development be Agile?," *Communications of the ACM*, Vol. 49, No. 10, October 2006, pp. 41–46.

[27] A. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," *Computer*, Vol. 34, No. 11, November 2001, pp. 131–133.

Modern software developers often *do* have a wider range of skills—individually, as well as team-wide—than in the past. The diversity of software architectures and tooling have expanded the skill sets needed to effectively develop complex products. This wide range of skills plays into the characteristic of self-organization with Agile teams able to reconfigure as needed to facilitate a specific project need. Our analysis of job postings (discussed in Chapter Four) indicates that there is a continuing need for skills specialization as teams hire to fill specific gaps. Achieving the ideal cross-functional individual envisioned by some Agile advocates may not be necessary or even possible, but many software teams regularly reconfigure themselves to ensure that knowledge is not monopolized by select team members.[28]

The project team dynamic is particularly important when applying an Agile SDLC to highly innovative or emerging applications since the pace and tempo requires the ability to make swift group decisions under ambiguous circumstances. To achieve a high level of collaboration, teams must have a healthy balance of personality, talent, and communication skills. If this balance is lacking, the team will not be able to function efficiently and effectively.[29] The implementation of Agile values and principles has been shown to improve communications skills among preexisting as well as newly formed teams.[30]

Agile Software Development Life Cycle in Practice

Both the initial and final phases of the idealized Agile SDLC present challenges to DoD acquisitions. Early engagement with end-users can be limited if personnel cannot be spared from the battlefield or operations to support acquisition. Compliance with federal acquisition

[28] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj, "Challenges of Migrating to Agile Methodologies," *Communications of the ACM*, Vol. 48, No. 5, May 2005, pp. 73–78.

[29] Peter Schuh, *Integrating Agile Development in the Real World*, Rockland, Mass.: Charles River Media, Inc., 2004.

[30] Harald Svensson, and Martin Höst, "Views from an Organization on How Agile Development Affects Its Collaboration with a Software Development Team," in F. Bomarius and S. Komi-Sirviö, eds., *Product Focused Software Process Improvement*, PROFES 2005, Lecture Notes in Computer Science, Vol. 3547, pp. 487–501, Berlin: Springer, 2005.

rules governing competitive contract awards must be considered when designing forums for early pre-award engagements between competing development teams, the acquisition team, and operational teams. Keeping more than one development team in competition through risk-reduction (RR) contracts is one method used to achieve early engagement, as is the growing practice of using other transactional authorities (OTAs) to fund the development of prototype or experiments. However, transitioning from the RR/prototyping stage to full development brings its own risks and is far from the ideal envisioned by Agile advocates.[31] Knowledge transfer from these early engagements must be prioritized if the promise of an Agile SDLC is to be realized.[32]

While delivery to the end customer and deployment into the operational environment are the ideal goals of an Agile SDLC, in practice for DoD programs, delivery to a test and evaluation team and deployment into initial operating test and evaluation (IOT&E) may be the closest DoD can come to this ideal. Even that can be challenging given that the hardware that comprises a developing complex weapons system or vehicle is often not available until quite late in the acquisition life cycle. Identifying the models and test resources needed to support software development is therefore a critical software competency for DoD. Differences in tempo or cadence between the test and evaluation teams and the development team or the operational teams can be accommodated by performing delivery and deployments on specified software releases as opposed to each iteration.[33] Synchronizing the tempos of the different teams is therefore defined as a software competency in our work.

[31] Unfortunately, labeling the work in RR as a prototype sets an expectation that it is not "working software," which may be detrimental to quality.

[32] On the contractor side, a break between RR and SD phases will often mean that a new team of software developers takes over. If continuity of software personnel is a requirement for source selection, this risk can be reduced, but it may not always be obvious to the source selection team which members of a self-organized team are the key to its success.

[33] Alignment of releases with the operational tempo(s) of the battlespace may be key to end-user acceptance and can be particularly challenging.

Considerations in Design of a Software Development Life Cycle

A key software competency for DoD software acquisitions personnel is the ability to design and manage an SDLC in a way that mitigates the specific risks associated with a specific acquisition. For example, when comparing Waterfall and Agile SDLCs as they appear in practice in DoD programs, an incremental-build SDLC and an Agile SDLC that uses incremental releases for delivery and deployment appear remarkably similar. One distinction, however, is in how requirements analysis and architectural design are treated. An Agile approach is designed to allow for more fluidity, experimentation, and prototyping in the requirements and architecture and may be particularly well suited to innovative applications where there is uncertainty in the capabilities desired and/or required. More well-established applications with strong dependencies on evolving hardware may be better suited to the incremental build. We also note that not all software within a program need use the same SDLC, and different SDLCs can be used during different phases. In fact, it is not uncommon for a different SDLC to be used for acquisition and sustainment in DoD programs. The key lies in having DoD software acquisition personnel well versed enough in the theoretical application of both SDLCs such that they can make empowered decisions as to which SDLC (or combination of SDLCs) will best fit a given program and/or phase. With that in mind, we offer a short description of key advantages and disadvantages that appear in the literature,[34] tempered with the experience of the SMEs with whom we interacted over the course of this study.

The largest challenges cited in using Waterfall SDLC include

- *Incomplete/inaccurate requirements*: Waterfall SDLCs perform capabilities elicitation at the beginning of the project, when customers and stakeholders may not have an in-depth understanding of all the needs associated with their desired product. As a result, capabilities may be overlooked, and if added late, may

[34] Lotz, 2013; Balaji and Sundararajan Murugaiyan, 2012, pp. 26–30; Munassar and Govardhan, 2010, pp. 94–101.

require earlier stages of development to be revisited, often at great expense.[35]

- *Inability to accommodate change*: In multiyear developments there are numerous sources of change. Change in the operational threat environment is of particular concern for DoD programs since the "adversary gets a vote." As DIB recently wrote, "The Department of Defense (DoD) must be able to develop and deploy software as fast or faster than its adversaries are able to change tactics."[36]
- *Late customer feedback*: The final product does not begin to take form until late in the Waterfall process. This can pose a challenge if the customer—once having seen the product—decides that the capabilities or features they outlined do not meet their needs. In DoD acquisitions, if IOT&E, safety, or security accreditation teams have not been involved throughout the development period, these late life-cycle deliveries can be particularly problematic.

It is important to acknowledge that while programs with planned updates throughout the development life cycle of a given capability may experience less severe setbacks than programs without planned updates, the challenges outlined above can still manifest and cause significant delays.

Defining requirements early and a linear development approach do, however, have benefits for specific types of projects. Certain advantages associated with the Waterfall approach include

- *Straightforward management across projects*: Utilizing the Waterfall approach across projects can ease and facilitate the management process. Although each project will encounter different issues, the overall structure of the process will be the same across projects. For systems that are predominately comprised of hardware proj-

[35] While it is possible to go back and make adjustments to prior stages, the Waterfall methodology makes this extremely difficult and expensive. Several reputable software development firms report that fixing defects in the later stages of testing is ten to 100 times more expensive than fixing it during the phase where the defect was inserted. One of our SMEs reported collecting data on a DoD project in the 1990s that validated this rule of thumb.

[36] DIB, 2018.

ects, aligning the software development life cycle to the hardware, as the Waterfall model does, may eliminate some miscommunication, but it also may obscure critical differences that management needs to be aware of (such as software dependencies on hardware characteristics that cannot be known until production hardware is deployed into the test or operations environment).
- *Gated progress*: Due to the combination of setting starting and ending criteria for each stage and specific requirements for each stage, progress is easy to track and can be concisely described.
- *Detailed documentation*: Since documentation is an explicit part of each stage, a more thorough picture of the entire process logic may be captured and retained for future reference, if necessary.

The Agile approach to software development evolved to address the disadvantages of the Waterfall approach, but in doing so creates its own disadvantages:

- *Lack of a precise plan*: Lack of a precise plan can make clear communication of project progress with the customer difficult. For DoD projects, communicating software development progress through earned value management has been particularly difficult.[37] The trade-off here, though, is that both the Agile team and the client understand that by accepting to move forward on a less-structured plan, they allow themselves the flexibility to learn and adapt with the project as it develops.
- *Time commitment for the customer*: Agile compensates for the lack of a precise plan through continuous end-user and customer engagement to obtain feedback during the entire project. An Agile SDLC may result in the development of a suboptimal product if end-users are not available to support these interactions. As we noted earlier, freeing up operations or IOT&E personnel on DoD programs to perform the "voice of the customer" role is challenging.

[37] Office of the Secretary of Defense, "Agile and Earned Value Management: A Program Manager's Desk Guide," April 16, 2018.

- *Project success hinges on the team*: Agile may require a higher level of multiskilled personnel than a Waterfall. Also, due to the high level of team collaboration, Agile may not be suitable for very large teams, although modern tooling and techniques such as "Scrum of Scrums" have been used effectively on some large programs.[38]
- *Deficient documentation*: Since Agile does not need to use documentation as a means to transmit information from architecture to code and test, the resultant documentation at the end of the program may not be as comprehensive as that developed in a Waterfall. Determining the appropriate level of documentation needed is a critical software competency for all SDLCs.

The most frequently cited advantages to utilizing an Agile SDLC are

- *Rapid response to change*: Because the Agile SDLC revolves around a series of sprint cycles, any changes—whether in capabilities or the prioritization of features or in the availability of test environments—can be accommodated into subsequent sprint cycles. In some high-priority cases, the desired change can be implemented in weeks, rather than months or years.
- *Deliverable product definition maturation*: Agile's iterative process lends itself to projects with an unclear end goal; these include cases where a customer knows there is a gap, but does not know what the answer looks like, and therefore cannot clearly articulate an end product or requirements. In these instances, the Agile approach can greatly facilitate the discovery process.
- *Improved customer/user acceptance*: Due to the customer's or end-user's close involvement with the development team, their priorities and vision ideally guide the development and influence the final deliverable. They observe product maturation over time as their feedback is received and new features are added, which theoretically improve the ultimate end-user experience. However, as

[38] The Scrum of Scrums approach is described on the Agile Alliance website. Agile Alliance, "Scrum of Scrums," webpage, undated.

we have noted throughout, achieving an effective "voice of the customer" can be challenging on DoD programs.

- *Higher-quality end product*: If each sprint cycle closes with the project team's review of lessons learned, an Agile SDLC may allow for the incorporation of these lessons learned into the next sprint cycle, which will result over time in an overall higher-quality product.

Claims of improved customer acceptance appear to be substantiated with quantitative studies. However, there is little definitive research to defend the claim of higher software quality, given that the fact that Agile teams are often more experienced confounds causal analysis.[39]

Use of Agile Software Development Life Cycles in Government

The use of Agile within government is becoming increasingly common across multiple federal agencies: The Defense Information Systems Agency (DISA), the National Aeronautics and Space Administration, the Federal Bureau of Investigation, the Department of Veterans Affairs, and the Patent and Trademark Office have all documented use of Agile.[40] As the number of federal agencies utilizing Agile increases, information on how the methodology scales to large, complex settings along with best practices for effective implementation are emerging.

A GAO report released in 2012 evaluated the use of Agile across five federal agencies.[41] This report revealed ten key practices that officials from all five agencies identified as effective for successful implementation of the Agile method (see Figure B.3).

[39] A 2008 systematic review of the literature related to Agile software development found that only a handful of studies had looked at software quality, and the results were inconclusive. Studies on customer acceptance were more conclusive. Tore Dybå and Torgeir Dingsøyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information and Software Technology*, Vol. 50, 2008, pp. 833–859.

[40] GAO, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*, GAO-12-681, July 2012; Deloitte, *Agile in Government: A Playbook from the Deloitte Center for Government Insights*, 2017.

[41] GAO, 2012.

Figure B.3
Practices Used and Found Effective by Five Agencies

Practice

1. Start with Agile guidance and an Agile adoption strategy.
2. Enhance migration to Agile concepts using Agile terms and examples.
3. Continuously improve Agile adoption at both project and organization levels.
4. Seek to identify and address impediments at the organization and project levels.
5. Obtain stakeholder/customer feedback frequently and closely.
6. Empower small, cross-functional teams.
7. Include requirements related to security and progress monitoring in your queue of unfinished work (backlog).
8. Gain trust by demonstrating value at the end of each iteration.
9. Track progress using tools and metrics.
10. Track progress daily and visibly.

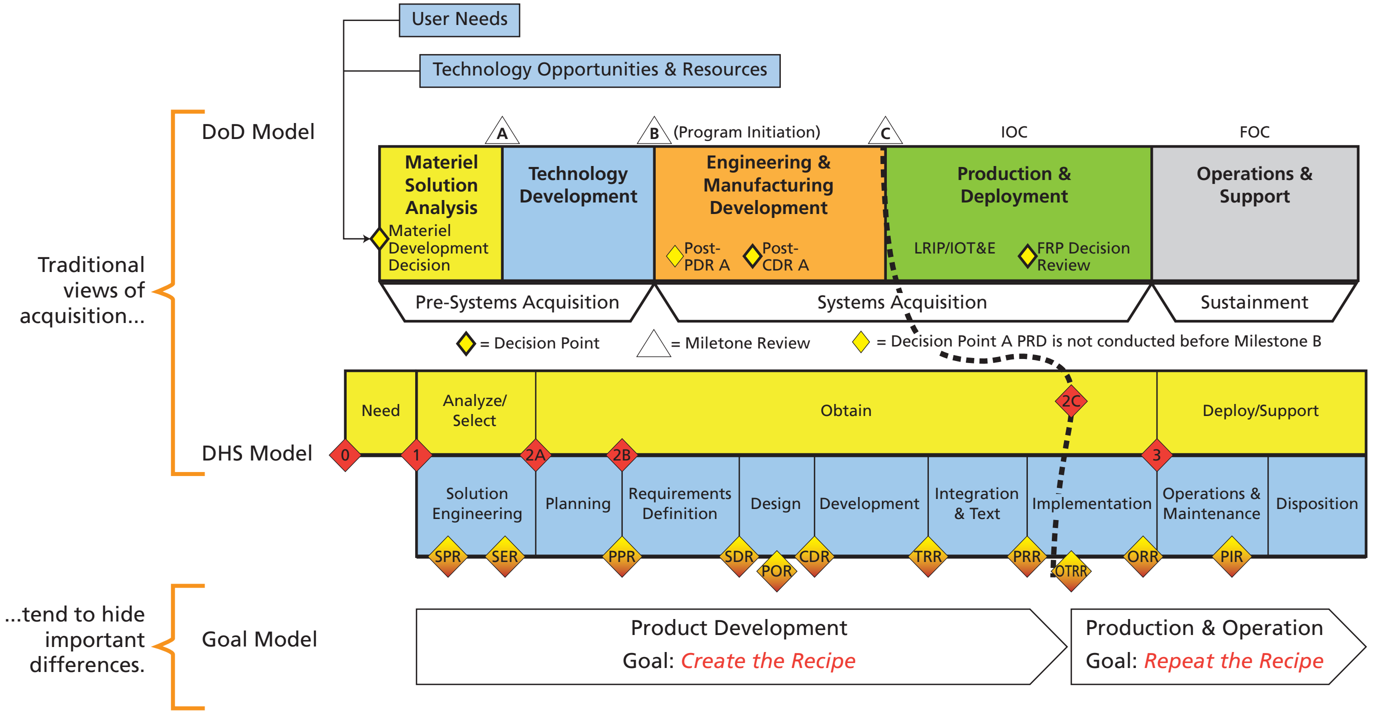
SOURCE: GAO, 2012, Table 1, Practices Used and Found Effective by Five Agencies.

One key challenge to successful adoption of the Agile approach by government agencies is current acquisition practices. Current government acquisition models illustrate the acquisitions process as a singular, unified process composed of various stages. While this view of the process is not untrue, it does not acknowledge the fact that the various stages have different goals and different criteria for success (see Figure B.3). Instead, current models impose the goals and success criteria of the end stages and apply them to the beginning stages.[42] An example of this is controlling scope to limit variability. While this may be a goal of the production stage where uniformity is desired, if applied in the beginning stages of the acquisition process, it may limit the innovations needed to deliver superior solutions.

Another characteristic of current government acquisition models is that they are often very large and complex—a reflection of the agencies they serve. While this is not harmful to all types of acquisition, it is particularly harmful to software acquisition because the capabilities

[42] Troy Mueller, David Harvey, Awais Sheikh, and Scott Johnson, "Making Agile Work in Government," MITRE, May 2015.

Figure B.4
Different Goals, Different Values



SOURCE: Adapted from Mueller et al., 2015.

delivered via software change rapidly and with greater frequency—so much so that current acquisition models cannot adapt effectively and be responsive to such changes.[43]

Switching from a traditional DoD acquisition model to one that utilizes Agile may require a fundamental shift in an agency's organizational culture and mind-set. Changing an organizational culture is a difficult undertaking, requiring both a top-down and a bottom-up approach. A shift to an Agile SDLC may rest on leadership's ability to (1) let go of some decisionmaking power and allow teams to operate organically within the framework of the SDLC; and (2) revise accreditation, certification, test, and evaluations practices to accommodate incremental system delivery and deployment. The organizational culture will not change without leadership taking a visible and proactive role. But by actively engaging in this process, leadership exhibits their support of the change and encourages others in the agency to follow suit.[44] For this reason, we have included competencies associated with organizational change and design thinking in the software competencies developed by this project.

In order for the shift to be enduring, the Agile approach must be incorporated into agency policy in such a way that incentivizes programs to utilize and adhere to this approach. Strategies for accomplishing this would include education and training to ensure workforce competency and extend into on-the-job coaching and mentoring during the adoption period.[45] Education and training issues are addressed in Chapter Six of this report.

[43] Su J. Chang, Angelo Messina, and Peter Modigliani, "How Agile Development Can Transform Defense IT Acquisition," in P. Ciancarini et al., eds., *Proceedings of 4th International Conference in Software Engineering for Defense Applications*, Switzerland: Springer International Publishing, 2016; Mary Ann Lapham, Ray Williams, Charles (Bud) Hammons, Daniel Burton, and Alfred Schenker, "Considerations for Using Agile in DoD Acquisitions," Pittsburgh, Penn.: Software Engineering Institute, Carnegie Mellon University, 2010.

[44] Agile Government Leadership, "Cultural Transformation," *Agile Government Handbook*, 2016.

[45] Harry Levinson, "Helping Large Government Programs Adopt and Adapt to Agile Methods," Pittsburgh, Penn.: Software Engineering Institute, Carnegie Mellon University, 2016.

Development Architecture Trends: Monoliths to Ecosystems

Today's software is increasingly produced by organic ecosystems in which the development teams include commercial entities, open-source foundations, and individual developers. Ecosystems form around shared interface standards, operating systems, or platforms. Participants can contribute at any level of the software "stack" (the computing hardware, operating system, middleware, application interfaces, libraries, and application). Vertical integration, in which a single firm produces all, or even most, of the elements of the software stack are virtually nonexistent today. Even IoT devices include third-party elements in their software.

For the commercial industry, this trend is driven primarily by economics. Assembling executable software from a stack of proven products minimizes time to market and the resources that must be expended to deliver a working product to end-users. Unfortunately, this trend also may decrease the reliability of the software. While sharing elements with other systems means that more defects are found (and presumably fixed), it also increases the total amount of software deployed since elements such as operating systems will contain features that are not needed by all applications. Many firms have no clear insight into the pedigree of the elements they integrate into their products. Furthermore, if many applications use the same stack elements, that common use increases the incentive for cyber attackers to find and exploit vulnerabilities in those elements. DoD programs, many of which are safety- and security-critical, must carefully weigh the advantages and disadvantage of leveraging ecosystems in their products. More than one of the SMEs we talked to in the course of this study wondered if the time is right for DoD to develop its own ecosystem of secure and pedigreed elements of the software stack needed to support DoD application domains such as weapons systems or battle management. The feasibility of doing so is unknown, given that DoD lacks the market mechanisms that give rise to software ecosystems in other domains.

Established and Emerging Ecosystem

One of the first widely popular software ecosystems arose around the Eclipse Platform, an integrated software development environment that was built architected to be highly extensible through third-party developed "plug-ins." The Eclipse ecosystem is fostered through the Eclipse nonprofit foundation and is largely comprised of open-source tools and products.

The "mobile apps" ecosystem is another commonly cited example, encompassing the set of developer tools and support elements (such as app stores) used to develop and deploy literally thousands of applications to our mobile phones. Unlike the Eclipse ecosystem with its emphasis on open-source development, the mobile apps ecosystem is largely proprietary, with Google and Apple setting up "walled gardens" for their third-party developers and partners.

While we are still far from having a widespread autonomous vehicle ecosystem, advancements in technology and the number of companies taking on this challenge are moving society toward that reality at a quicker pace. These advances are changing the way companies view automobiles—from active modes of transportation to moving computational platforms, hosting a wide variety of applications from navigation to entertainment.[46]

Adoption of Open-Source Solutions

Many companies are adopting open source as a potential solution to high-cost software investments. Aside from the lower overhead factor, open-source tools and software offer companies many benefits, including faster development, increased flexibility for customization, and potentially more robust code due to a wide pool of feedback to draw from.[47] Open-source software provides full visibility into the code base, which is a plus for security- or safety-critical applications that need

---

[46] Daniel Eckert, "Three Big Emerging Technology Themes from CES 2016," *PWC*, January 13, 2016.

[47] Carolyn A. Kenwood, "A Business Case Study of Open Source Software," MITRE, July 2001; Testing Whiz, "8 Software Testing Trends Every Tester Should Follow in 2018," January 10, 2018.

access to source code for certification or accreditation. Open-source software products provide relatively fewer intellectual property or data rights constraints than their proprietary counterparts.

Generating Reusable Code

A trend that is being encouraged within the software development field is the generation of reusable code to reduce production time, as well as to enhance security. When done internally, this practice would result in the creation of a code repository with scripts of basic, widely applicable code that can be dragged and dropped into new scripts being developed. This practice requires companies to establish secure internal sharing platforms, code-sharing policies, and training procedures to promote safe individual security practices.[48]

Deployment Architecture Trends: From Stand-Alone to Clouds and Fog

In the design of a software system, the decisions determining what aspects of the software should run on which processing units lead to what is called the *deployment architecture*. Very few applications in today's connected world reside only on a single processor. For example, in a typical bank automatic transaction machine (ATM), the software resident on the ATM itself handles the interface to the user (e.g., card and personal identification number input, service selection and money, check, receipt handling), but the bulk of the software that allows you to make deposits, withdraw cash, or check your balance resides on secure servers, perhaps even in geographically distributed server farms managed by large firms that provide computing resources on demand (i.e., the public cloud). The software resident on the ATM is called *edge computing*, signifying that it is at the edge of the internet, at the user interface. In today's most sophisticated banking systems, the interface

[48] European Center for Security and Privacy by Design, *Emerging Trends in Software Development & Implications for IT Security: An Explorative Study*, Darmstadt, Ger.: Technical University Darmstadt, June 2014.

software that lets you pay for goods and services resides on your phone, interacts with the store's "point of sale" software resident in a device connected to the register, and both systems connect to a much large banking system, resident on servers, to complete the actual accounting that transfers money from your account to the store's. If the software at the edge of the system is embedded in a more specialized device such as your home thermostat, it is termed IoT. The design of a "smart" home thermostat is likely to include a web service to configure the device, as well as software resident on the thermostat itself.

In DoD applications such as the F-35, software resides in multiple computers controlling battle management and fire control systems, radars and other sensors, flight control systems, and communication systems (radios). Off-board computers host maintenance software for diagnostics and other support functions such as mission debrief. In total, the F-35 system (comprising both on-board and off-board processors) is reported to have more than 8 million lines of code spread across multiple processing units ranging from small devices, to real-time embedded core processors, to servers.[49]

Living in the Cloud

Application of the cloud to a wide range of uses continues to mature. As noted earlier, the cloud is simply a system of large server farms linked via fast internet connections to end-users or gateways. These server farms are geographically located where energy is inexpensive, yet are close enough to end-users to provide relatively fast response times, provided users have good internet connectivity. The primary advantage of using a cloud provider for computing is that the user does not have to purchase and maintain the hardware. For unsophisticated users, public cloud services from reputable firms are likely more secure than the typical home computer. Large firms and/or DoD may build their own private cloud systems.

The second advantage of a cloud system is that users can access applications and data stored in the cloud from anywhere an internet

[49] Lockheed Martin, "F-35 Lightning II: A Digital Jet for the Modern Battlespace," webpage, undated.

connection is available. For civilian use in an increasingly wired world, this reliance on an internet connection may be a relatively low-risk dependency. For militaries operating in denied or degraded electronic environments, however, it is a significant limitation.

When only computing services (i.e., processors, memory) are provided to the users, the arrangement is called *Infrastructure as a Service* (IaaS), but there has been rapid growth in providing more elements of the software stack and/or software development tools to provide what is called *Platform as a Service* (PaaS). PaaS is having a profound effect on how software is developed and tested. For large projects or in support of an ecosystem, a core team of "pipeline" engineers will tailor a suite of software development and test tools (i.e., a platform) for the project or ecosystem application that test teams can then access on demand. This ensures uniformity across the project or ecosystem and thus improves interoperability and frees application developers and testers from the need to install, configure, and maintain the lower levels of the software stack or the software development and test tool sets. If one is using a public cloud provider, the flexibility of being able to instantiate a test bed or development environment on demand, only paying for the resources when they are needed, may be cost effective. This flexibility is often cited as a primary factor in decisions to use PaaS.[50]

When a complete software application (such the Microsoft Office suite) is made available to users over the internet from a cloud architecture, it is called *Software as a Service* (SaaS). Typically, SaaS providers charge an annual subscription fee, which is a valuable source of revenue to the provider. SaaS subscriptions are often cost effective for the user in the short term but may not be in the long term. Careful cost analysis over the entire life cycle of a system is required when deciding to incorporate SaaS into a system architecture.

Trends in Internet of Things and Edge Computing

IoT is evolving from the "*internet* of things" to the "*interoperability* of things." This evolution means that rather than users simply being

[50] Capgemini, *The Changing Dynamics of the Global High Tech Industry*, 2011.

able to control certain devices from others (i.e., turning on your lights from your phone), these devices can now automatically interact with one another based on a specific set of characteristics and preferences (i.e., all the IoT devices in a room will alter their settings—such as turning up lighting and changing the music—as a person enters a room).[51]

In addition to the increasing proliferation and interoperability of IoT devices, these devices' ability to compute is also increasing. The IoT devices at the edge of the network are increasingly able to perform data processing and analytics. This means that end-users experience faster real-time analytics, without having to transfer data to a central data center for analytics processing.[52] This capability has given rise to a new type of architecture called *fog computing*, in which a collection of IoT devices at the edge of the internet provides low-latency data processing without reliance on a permanent connection to the wider internet.[53] Fog computing may be of special value to DoD forward units when communications links back to secure servers are unreliable.[54]

The improved performance of edge processors and improvements in web service delivery have also enabled the development of what are called *progressive web apps* (PWAs)—webpages that appear and interact with the user almost as if they were native applications hosted on the edge device itself. PWAs enable continued use of many of the app functions even when disconnected from the internet.[55]

---

[51] Eckert, 2016.

[52] Ben Putano, "6 Software Development Trends for 2018: Developers Needed," *Stackify*, November 24, 2017.

[53] Fog computing is named to evoke the concept of a cloud close to the ground. It applies cloud computing concepts to the IoT processors at the edge of the network. See Christopher Mims, "Forget 'the Cloud'; 'The Fog' Is Tech's Future," *Wall Street Journal*, May 18, 2014.

[54] Divya Lanka, Ch. Lakshmi, and D. Suryanarayana, "Application of Fog Computing in Military Operations," *International Journal of Computer Applications*, Vol. 164, 2017, pp. 10–15.

[55] Kerry B., "5 Software Development Trends to Watch for in 2018," April 23, 2018.

Automation in Software Development: From "Quality Assurance" to "Quality Engineering"

Traditional software quality assurance (QA) practices were largely reliant on human inspection, analysis, and testing. As software development efforts became larger and more complex, teams quickly realized that repeating these steps each time a change was made to the software was time consuming and that variations in the process led to uncertainty regarding the quality of the software. They also realized that postponing these steps led to increased cost and scheduling; defects are most efficiently removed when they are found early. Common practice is, therefore, to automate these steps to the extent practical. Twenty years ago, this automation would largely be custom built into test beds, supported by scripting languages to provide inputs to and collect outputs from the software under test. Analysis tools were also often custom built, and inspections continued to be done by humans. Today, tools have been developed to automate not only inspections, but also commonly used analysis techniques, along with portions of software build, integration, and testing.[56] This shift in automation has given rise to the term *quality engineering* (QE). QE allows development teams to maintain high quality at higher speeds, reducing time to delivery by optimizing functional testing and enabling teams to build automated "fitness functions" that continually evaluate the architecturally important quality attributes of the software.[57] The terms *continuous integration*, *continuous delivery*, and *continuous deployment* are used to designate which activities in an SDLC are automated.

Continuous Integration

Continuous integration of software automates the process of software merge, build, and integration to ensure that newly committed code is compatible with existing code, meets quality standards for safety and security, and has not "broken" existing functionality or performance. It usually includes the use of static and dynamic analysis tools and the

[56] Testing Whiz, 2018.

[57] Gifographics Creative Team, *6 Emerging Software Testing Trends That Will Rule 2018*, infographic, June 8, 2018.

execution of a suite of regression tests. In the 2000s, it was common to run this automation nightly (i.e., "the nightly build") so that developers arriving the next morning would have notifications in their inboxes of items to be fixed. With improvements in computing capability, these steps now execute so quickly that the common practice is to trigger the automation whenever a developer "submits" code to provide feedback in nearly real time.

Continuous Delivery

Continuous delivery of software automates the complete integration and test process and includes all steps necessary to package the software for delivery into the operational environment. This practice can include audits needed for safety and security certification or accreditations. Often the steps that comprise the continuous integration and delivery process are termed *the pipeline* to emphasize the continuous flow of product through these processes.

Continuous Deployment

Continuous deployment takes automation all the way to the operational environment. For teams that have implemented continuous deployment, the updated code is automatically pushed out to users, sometimes as often as every 12 seconds.[58] While this level of speed is not necessary for every software deployment, continuous deployment may be beneficial in some cases. The risk of deploying new software directly into operations can be mitigated using techniques such as A/B testing, in which some users of a website are selected to use the new software (B) while the bulk of the users remain on the prior software baseline (A). This approach can help teams obtain immediate feedback from actual, as opposed to simulated, operations with relatively low risk. Unfortunately, it can also make users unwitting testers of unproven functionality if the integration and delivery pipelines are not engineered with an appropriate emphasis on software quality.

[58] Putano, 2017.

APPENDIX C

Tracing Initial Competencies to Other Competency Models

We traced the initial draft competencies to five existing models:

1. IT career field model updated by IT FIPT, which was provided to RAND by the executive secretary of the IT FIPT
2. IEEE's SWECOM
3. IT career field's AWQI model
4. ENG career field's AWQI model, pulled from the AWQI website in October 2017
5. PM career field's AWQI model, along with an updated PM career field model.

Although other models and information (e.g., SEI reports) were consulted to develop the initial draft competencies, the five models listed above were the primary sources used. Also, because the competency model evolved over the course of the study, the mapping in this appendix reflects the initial competencies but does not necessarily reflect the final competencies.

Table C.1
Mapping of Initial Draft Competencies to Five Existing Competency Models

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Problem Specification	Identify and specify the problems that must be overcome to enable a desired capability, based on consultation with key stakeholders.	• Identify stakeholders, elicit capability objectives, and negotiate conflicts among stakeholders as required • Evaluate stakeholder objectives to identify and specify system operational requirements and capability needs (includes gap analysis vs. existing capabilities) • Identify key performance parameters (KPPs) and other performance specifications for inclusion in capabilities documents	IT: 7, 14, 35 ITa: 7, 13, 26 ENGa: 2-3, 12, 16, 20-21 PMa: S: 1-5, 15, 27-30, 39, 56, 58
Solution Identification	Identify and specify a desired solution approach to the problem based on utilizing alternative analysis, market research, trade-off analyses, and business drivers of cost, schedule, capability, and risk. This includes initial implementation and integration efforts (e.g., prototyping).	• Apply or conduct an analysis of alternatives to ensure data-based decisions for meeting critical objectives • Apply methods to assist solution identification which may include context definition, prototyping, and dependency analyses • Explore options for reuse of existing GOTS and COTS capabilities • Identify cost and schedule drivers associated with key performance parameters/key system attributes decision elements	IT: 3, 4, 5, 27, 39 ITa: 4-5, 15, 22 ENGa: 1, 20-21 PMa: S: 27-29, 31, 58

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Development Planning	Identify and implement methods, processes, and lifecycle management approach to be used for system development (and/or purchase), and deployment. This includes project planning from initial concept development through implementation, integration, deployment, and transitions to operations.	• Select methods, processes, and a life-cycle approach (such as Agile, Iterative, Waterfall, etc.) that are appropriate to the development needs. This includes cost and schedule management, team communication, requirements management, mission and quality assurance, change management, corrective action, configuration management, and release management, among other processes • Select metrics and measures appropriate to managing software scope, cost, schedule, and quality. This may include quantitative methods to assess and track software progress against a baseline (planned vs. actual) • For each acquisition phase, determine the appropriate entrance and exit criteria to minimize program risk • Develop detailed plans for installation, acceptance testing, and accreditation of the operational system within a larger system environment • Select appropriate languages, tools, frameworks, platforms, and environments that will be needed during software design, code, validation, verification, and sustainment and identify how the configuration of these items will be managed	IT: 17, 18, 2, 19, 20, 29, 32, 10, 22, 39 ITa: 1-2, 10, 14-18, 20 ENGa: 5, 10-11, 13, 15, 21 PMa: S: 5, 7, 10-12, 14-17, 22-24, 26, 33-34, 38, 51-55

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Transition and Sustainment Planning	Identify and specify the accountabilities and dependencies needed to successfully transition the software from initial development (and/or purchase) to sustainment, and the methods, processes, and life-cycle management approach to be used during sustainment of the system until its termination.	• Develop detailed plans for transitioning accountability of the software requirements, design, code, and verification artifacts to the sustainment organization • Plan for and manage future modernizations to meet emerging requirements and/or relationships with other systems • Develop detailed plans for sustainment of the software through system termination	IT: 11, 30, 39 ITa: 11 ENGa: 9 PMa: S: 5, 12, 16, 19-21, 32, 34, 51-53
System Architecture Design	Specify the system architecture at various levels of implementation. This includes specification of where the system fits within the context of the broader DoD ecosystem down to implementation on end-user hardware.	• Develop, review, and evaluate alternative system architectural designs. (Architectures may be based on distributed components or rely on external dependencies. This task may include architectures within the DoD Information Enterprise Architecture) • Perform or utilize enabling techniques such as abstraction, coupling/cohesion, and information hiding, as appropriate • Ensure aspects of quality attributes (e.g., performance, interoperability, sustainability) and risk mitigation techniques (e.g., system safety, security, and usability) are integrated into architecture specifications as appropriate • Specify a final design architecture based on reviews of alternative designs	IT: 13, 8, 14, 27, 28, 37, 38 ITa: 8-9 ENGa: 4, 13, 19 PMa: S: 2, 6-8, 12, 32, 46

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Validation Modeling	Specify the software models that comprise the software system components. This may include implementation details about database design, object-oriented design, interface design, among other characteristics.	• Develop, integrate, employ, and evolve the authoritative model of the system under development • Employ models appropriate to the requirements which may include formal logic, state machines, and process models • Use models to explore quality attributes and other design considerations such as managing concurrency, event handling, data persistence, or distributed software • Interpret modeling or simulation results to explore concepts, refine system characteristics/designs, assess overall system performance, and inform acquisition program decisions	IT: 13, 22 ITa: 9 ENGa: 4, 11 PMa: S: 6-9, 12, 31-32, 40, 46, 57
System Attribute Analyses	Explore and specify how the system is meeting the key attributes that the software solution must satisfy. Examples of key attributes include availability, integrity, and performance scalability.	• Apply and execute the software security practices and analyses necessary to meet system requirements • Apply and execute the software safety practices and analyses necessary to ensure the resultant system will meet system requirements • Apply and execute the software reliability and maintainability (R&M) practices and analyses necessary to ensure the resultant system will meet system requirements • Conduct appropriate analyses necessary to ensure the resultant system will meet all other specified quality attributes	IT: 13, 14, 22, 26 ITa: 8, 25-27 ENGa: 4, 6, 10-11, 19 PMa: S: 2, 9, 15, 17, 32, 39-50, 58-60

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Software Construction Management	Implement plans for development (and/or purchase or sustainment), and manage objective and threshold requirements and qualities throughout the acquisition against constraints in technology, cost, schedule, and policy.	• Continually engage with stakeholders to a) surface discrepancies between user-defined needs and specifications, and b) recommend trade-offs for affordability and schedule feasibility • Continually engage with ongoing mission and quality assurance activities to elicit corrective action recommendations (e.g., bug fixes, relief of performance bottlenecks, changes to software library dependencies, correction for method or process misalignments) • Evaluate change recommendations from stakeholders or for corrective action for impacts to technology, performance, cost, schedule, and policy • Approve recommended changes to the software development plans and/or features of the software solution within the constraints of technology, cost, schedule and policy • Implement approved changes to the software development plans and/or features of the software solution • Perform analyses to confirm resolution of approved changes	IT: 13, 12, 14, 15, 19, 20, 25, 32, 27, 28, 30, 36, 38, 40, 41, 42 ITa: 6, 14, 18, 20 ENGa: 5, 10-11, 15 PMa: S: 5, 11-12, 18, 24, 26, 28, 33, 38, 52

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Cost Management	Implement plans for development (and/or purchase or sustainment), and manage the cost of the acquisition against constraints in scope, schedule, and policy.	• Conduct a decomposition of the system into its key elements and cost drivers using work breakdown structures (WBS) aligned to program plans and the software architecture • Estimate software system cost using methods that account for software size, complexity and required attributes, expected changes, the need for future corrective actions, and program risks • Establish a software cost baseline • Plan and implement execution year adjustments to the cost baseline or make contingency plans in response to program progress (vs. plan), anticipated requirements changes, or external resource adjustments (Congressional/OMB/service or agency) • Implement cost estimation and monitoring processes to assess and track software-reliant program progress	IT: 6, 21, 23, 34, 41 ITa: 3, 6, 18 ENGa: 10-11 PMa: S: 11, 24, 26, 28

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Schedule Management	Implement plans for development (and/or purchase or sustainment), and manage the schedule associated with feature development and release against scope, cost, and policy.	• Develop a schedule for executing the planned software activities, including schedule buffer to accommodate expected change, future corrective actions, and program risks • Establish a software schedule baseline • As needed, assess the impact to schedule from changes in the requirements, staffing levels, and internal or external dependencies • As needed, manage schedule buffers to minimize the risk of cascading effects from critical ordered dependencies • Implement quantitative methods and measures (such as an integrated master schedule) to assess and track software progress against the baseline	IT: 6, 21, 41 ITa: 6, 18 ENGa: n/a PMa: S: 11, 24, 26
Policy Management	Implement plans for development (and/or purchase or sustainment), while considering and adhering to relevant laws, regulations, and policies (e.g., data and property rights, ownership) and managing against constraints in scope, cost, and schedule.	• Identify and review organizational policy regarding use of standard processes, methods, tools, metrics, and measures • Identify and review current laws, policies, regulations, directives, and guidance applicable to management and acquisition of DoD IT programs • Tailor governing policy, as appropriate, to establish an initial program baseline that is compliant with current laws, policies, regulations, directives, and guidance (to include Title 10 direction) for the acquisition effort • Use quantitative and qualitative methods and metrics to ensure the developer's implementation is compliant with the approved program baseline	IT: 1, 9, 16, 28, 41 ITa: 1, 27 ENGa: 10 PMa: S: 11, 24, 26, 36-37

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Mission Assurance	Identify, specify, and execute strategies for managing mission risks and meeting validation, certification, and accreditation needs. This includes operational test, evaluation, and audit support.	• Establish, specify, and manage an integrated risk and opportunity management process • Identify mission risks and propose appropriate mitigation activities • Conduct a crosswalk to assess technical, financial, and contract documents are consistent with the proposed technical solution and program planning • Continually assess the software design (e.g., boundaries, interfaces, standards, available production process capabilities, performance and behavior characteristics) to validate the ability of the resultant product to meet mission requirements, including that it interfaces properly with the rest of the system • Continually assess mission areas end-to-end, across system and platform boundaries, to identify and close integration and interoperability (I&I) gaps in mission critical capabilities • Develop operationally representative test plans and test management plans to ensure that all expected deliverables are met and that those deliverables are fully functional • Conduct accreditation management (e.g., assess results from operational test and evaluation, traceability from requirements to test plan, metric management ([KPP, KSA]) to ensure products meet their intended use and can operate within intended environments and dependent systems	IT: 19, 20, 22, 27, 33, 37, 39, 42 ITa: 19 ENGa: 1, 4, 8 PMa: S: 2, 4-5, 9, 15-18, 21, 33, 35, 37, 45-46, 50, 54-55

Table C.1—Continued

Competency	Definition	Tasks	Trace to Model and Competency Number[a]
Quality Assurance	Identify, specify, and execute strategies for managing project risks, corrective analyses, and meeting verification needs. This includes development and integration test and evaluation.	• Continuously conduct corrective action assessments, (i.e., monitor metrics such as bug reports, static analysis results, peer review processes, and testing processes) to identify adverse trends. This may include independent audits • If adverse trends are identified, conduct root cause corrective action to identify recommended process or product improvements • Identify verification plans and procedures to be included in the software planning • Conduct verification activities (e.g., verification test planning and execution, software design reviews, static analyses, coding standards, unit test and code coverage) and verify the system elements against their defined requirements (build-to specifications) • Trace verification activities to link modeling and simulation, developmental test and evaluation and operational test and evaluation together, as needed to document system capabilities, limitations, and risks against the system requirements	IT: 12, 19, 20, 24, 22, 41 ITa: 12, 19, 21, 23-25 ENGa: 6-7, 10, 14, 17-18, 21 PMa: S: 9, 13-19, 25, 35-36, 38-51, 54-55, 59-60

NOTE: [a] The following notation is used to represent the five competency models: IT = IT career field model updated by IT FIPT, S = IEEE's SWECOM, ITa = IT AWQI model, ENGa = Engineering AWQI model, PMa = PM AWQI model.

APPENDIX D

Notional Example of Software Careers

Because the software acquisition workforce has not yet been defined, we draw on our research team's software expertise and review of commercial industry practices (Chapter Four and Appendix B) to propose possible career paths for software professionals. These and any other career paths should be more fully evaluated once the workforce has been identified. Table B.1 provides a crosswalk of five potential career paths with the software competencies described in this report (Chapter Five). In Table D.1, we use a "P" to indicate a primary competency and an "S" to indicate a supporting competency that is useful but not critical. Below is a brief description of each possible career path.

- *Program Managers* and *System Engineers* have primary accountability for stakeholder relationship management and overall accountability for program technical quality, cost, and schedule.
- *Enterprise and Software Architects* are primarily accountable for making architectural-level trades and ensuring the product quality attributes are appropriately balanced such that delivered product meets stakeholder needs.
- *Software Project Managers* are the Scrum leaders and release managers. They have detailed accountability for synchronizing the development and release of their individual products. They perform the detailed management of configuration, cost, and schedule.
- *Software Integration Managers* are focused on integration, test, and delivery. They manage the DevOps pipeline and have the detailed accountability to deliver products to specific groups of users, external test environments, and other stakeholders.

- *Software Technologists* are the technical experts in one or more software specializations. These are the people the architects and system engineers go to when they need deep expertise to build models or perform analyses (safety, cyber, performance). The project and integration managers go to them when there are tough problems to be solved.

In terms of career paths, software project managers could grow to be program managers and system engineers, and software technologists could develop into enterprise and software architects. Software integration managers have generally been around only since the mid-2000s and have less defined career paths.

Table D.1
Example of Possible Software Careers and Corresponding Competencies

Program Managers and System Engineers	Enterprise and Software Architects	Software Project Managers	Software Integration Managers	Software Technologists	DRAFT Software Competencies
S	P				Capabilities Elicitation
P	S				Business Case Development
P	S		S		Strategic Risk/ Reward Analysis
P	S			S	Cloud Computing
S	P			S	Software Ecosystems
S	S			P	Model-Based Engineering
S	S	P	S		Development Tempo
S	S	P	S		Release Planning
		P	S		Software Development Planning

Table D.1—Continued

Program Managers and System Engineers	Enterprise and Software Architects	Software Project Managers	Software Integration Managers	Software Technologists	DRAFT Software Competencies
		S	P		Planning for Continuous Delivery
S		S	P		Planning for Continuous Deployment
P	S				System Engineering Planning
S		P			Software Metrics
		S	P		Configuration and Version Control
	S	P	S	S	Software Documentation
P		S			Contracting for Software Development
P			S		Data and Proprietary Rights Management
	P		S	S	Architectural Design Approach
	S			P	Software Orchestration and Choreography Patterns
	P		S		Software Deployment Patterns
	P			S	Artificial Intelligence and Machine Learning Applications
	P			S	Augmented and Virtual Reality Applications

Table D.1—Continued

Program Managers and System Engineers	Enterprise and Software Architects	Software Project Managers	Software Integration Managers	Software Technologists	DRAFT Software Competencies
	P			S	Embedded Systems
S	P			S	Balancing Quality Attributes
S	P			S	Emerging Technologies
P	S				Use/Abuse Case Modeling
S	S			P	Validation of Performance Requirements
S	S		S	P	Validation of Sustainability Requirements
S				P	High Fidelity System Modeling
S		P	S		Software Assurance
	P		S	S	Cybersecurity
	S	S	S	P	Safety Critical Systems
	P	S	S	S	High-Availability Systems
S	S	P			Life-Cycle Management
S		P			Detailed Backlog Management
S		S	P		Release Management
P		S			Change Management
		S	P	S	Automated Test and Continuous Integration

Table D.1—Continued

Program Managers and System Engineers	Enterprise and Software Architects	Software Project Managers	Software Integration Managers	Software Technologists	DRAFT Software Competencies
S		P			Effort Estimation
S	S		P		Product Roadmap and Schedule Management
S			P		Cost Management
P	S				Legal Policy and Regulatory Environment Management
P	S	S			Risk, Issues, and Opportunity Management
P		S			Quality Assurance
S		P			Root Cause, Corrective Action
S	S		P		System Integration and Testing
P	S				Strategic Planning and Change Management
S	P			S	Innovation and Entrepreneurship
12	12	10	8	6	Number of primary competencies for this career path
22	19	11	13	13	Number of secondary competencies for this career path
34	31	21	21	19	Total

NOTE: "P" indicates a primary competency, and "S" indicates a supporting competency that can be useful but is not critical.

APPENDIX E

Existing Competency Models

Table E.1 provides a summary of the key competency models that we reviewed as part of the competency development process. It includes information about the hierarchical structure (i.e., number of levels) of the competency model, the number of competencies, and example competencies.

Table E.1
Competency Models Reviewed

Model	Number of Levels	Number of Competencies	Competency Example 1	Competency Example 2	Competency Example 3
DoD Career Field Functional Competencies—PM	4	70	**Configuration Management** (Basic)—Understand the configuration management process and how it can be used to provide technical insight into the program.	**Technology Management**—Use current/require science/technology as trade space to cover user needs recognizing that there will be gaps in coverage.	**Technical Data Management**—Ensure the application of the principles, procedures, and tools of data management and associated data rights.
DoD Career Field Functional Competencies—ENG	3	75	**Requirements Analysis**—Evaluate stakeholder and derived requirements (including constraints) and transform those requirements into a functional and technical view of a system capable of meeting the stakeholders' needs. Decompose needs and constraints into clear, achievable, and verifiable high-level requirements. As the system design evolves, allocate and derive requirements to the system elements and enabling system elements (hardware and software) to be designed and developed.	**Verification**—Generate evidence that the system or system element (hardware or software) performs its intended functions and meets all performance requirements listed in the system performance specification and functional and allocated baselines. Apply methods to verify performance, which may include the use of modeling and simulation, and developmental test, including Integrated Testing.	**Data Management**—Apply policies, procedures and information technology to plan for, acquire, access, manage, protect, and use data of a technical nature to support the total life cycle of the system.

Table E.1—Continued

Model	Number of Levels	Number of Competencies	Competency Example 1	Competency Example 2	Competency Example 3
DoD Career Field Functional Competencies—IT	2	42	**Contracting for IT**—Knowledge of IT specific areas of emphasis for acquisition IAW FAR and DFAR processes to develop and execute an acquisition strategy. (Includes TECHFAR)	**Enterprise Architecture—** Applies and/or assesses enterprise architectures (EA) and develops EA products (e.g. DODAF) to ensure compliance with DoD EA strategic goals. (Includes Information Support Plan (ISP))	**Cybersecurity**—Develops and applies Cybersecurity requirements for adequacy and effectiveness of security measures, continuity of operations, and protection of systems and system content.
Acquisition Workforce Qualification Initiative (AWQI)—PM	5	50	**Business Case Development**—Evaluate the merits and associated trade space of two or more potential solutions	**Acquisition Policy and Best Practices**—Apply current acquisition policy and best practices to products and processes in each phase of the Defense Acquisition Management System to enable sound acquisition management decisions	**Configuration Management**—Articulate the program technical insights provided by the configuration management process. Employ Configuration Management methods and best practices to establish and maintain consistency of a product's attributes with its requirements and product configuration information
Acquisition Workforce Qualification Initiative (AWQI)—ENG	5	21	**Validation**—Evaluate the requirements, functional and physical architectures, and the implementation to determine the right solution for the problem in an operationally-representative environment.	**Configuration Management**—Apply sound program practices to establish and maintain consistency of a product or system's attributes with its requirements and evolving technical baseline over its life cycle.	**Data Management**—Apply policies, procedures and information technology to plan for, acquire, access, manage, protect, and use data of a technical nature to support the total life cycle of the system.

Table E.1—Continued

Model	Number of Levels	Number of Competencies	Competency Example 1	Competency Example 2	Competency Example 3
Acquisition Workforce Qualification Initiative (AWQI)—IT	5	27	**Business Case Analysis**—Applies and/or assesses the rationale and key parts of building a business case to support achievement of critical business objectives.	**Enterprise Architecture**—Applies and/or assesses enterprise architectures (EA) and develops EA products (e.g., DODAF) to ensure compliance with DoD EA strategic goals.	**Contracting for Systems**—Knowledge of contracting stages and requirements for IT acquisitions IAW the FARS and DFARS processes to provide a clear and correctly informed acquisition process.
Software Engineering Competency Model (SWECOM)	2	60	**Software Requirements Verification and Validation**— • Checks requirements for accuracy, lack of ambiguity, completeness, consistency, traceability, and other desired attributes. • Constructs and analyzes prototypes. • Negotiates conflicts among stakeholders during verification.	**Software Maintenance**— • Establishes software maintenance processes and plans. • Obtains and maintains baseline software artifacts. • Performs problem identification and technical impact analysis. • Makes and assures changes to software (corrective, adaptive, perfective). • Performs preventative maintenance and software re-engineering. • Monitors and analyzes software maintenance activities.	**System Requirements Engineering**— • Establish the system development environment and identify technology constraints. • Identify system-level traceability requirements and tools. • Identify system requirements. • Develop the system requirements specification. • Develop plans, procedures, and scenarios for system integration, verification, validation, and deployment.

Table E.1—Continued

Model	Number of Levels	Number of Competencies	Competency Example 1	Competency Example 2	Competency Example 3
Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)	2–3	102 *topics*	**Software Requirements Fundamentals**	**Software Design**	**Software Construction**
Software Acquisitions Training and Education Working Group (SATEWG)	2	29	**Software Architecture**—The software structure of the system, including the definition of software components, and the relationships between them, the system, and the operational architectures.	**Software Configuration and Data Management**—An umbrella activity to manage evolving software baselines and related products that continuously ensure traceability, proper implementation and overall system/software compatibility across the lifecycle.	**Sustainment**—The post-delivery activities to support corrective, adaptive, perfective, and preventative software and system changes.

APPENDIX F

RAND-Developed Software Acquisition Competencies

This appendix contains the final draft set of competencies and associated definitions and tasks for the software acquisition workforce. These competencies were developed using several inputs:

- review of existing DoD and commercial industry competency models (Chapter Six and Appendix C and E),
- industry trends based on a literature review (Appendix B),
- discussions with industry SMEs (Chapter Four), and
- exploratory text analyses comparing software acquisition position announcements in commercial industry with those used by the Army and course descriptions in civilian software courses with those of DoD software course descriptions (Chapter Four).

After the initial model development, the competencies and definitions were revised following feedback from the sponsor's office and a series of SME panel workshops. The final set of competencies is designed to augment (but not replace) DoD-defined competencies for acquisitions (e.g., contract management, program/project management, system engineering, mission assurance, and so on) with software-specific detail. Because the software acquisition workforce has not yet been identified, the competencies still need to be validated by DoD following scientific and professional best practices, as described in Chapter Eight.

Each competency is described in terms of general work activities and tasks. In some cases, additional context and clarification are also provided. Because these competencies are technical, some terms may not be widely shared by all communities or may have different meanings depending on the context. To facilitate interpretation of these terms, we provide a glossary at the end of the competencies list.

Software Acquisition Competencies

Capabilities Elicitation

- Engage with stakeholders (to include representative end-user organizations, owners, developers, integrators, certification authorities, independent validation and verification personnel, and operators) to elicit capability objectives (i.e., functional requirements) and quality attributes (i.e., nonfunctional requirements) for the proposed system.
- Negotiate among stakeholders to prioritize needs and establish the product roadmap by identifying the minimum viable product and its possible evolutions.

  Context: Elicitation techniques vary in formality and may include solicitations of user stories (through observation of current tasking or through discussions) or completion criteria (e.g., asking stakeholders to define what "done" looks like), systematic exploration of scenarios and critical mission threads, or more formal use case modeling and quality attribute workshops.

Business Case Development

- Explore the problem space and identify focal areas for acquisition. Perform an analysis of alternatives using methods such as story mapping, success case methods, IT value perception models, and risk/reward trade-offs.
- Based on the analysis, recommend an acquisition solution and obtain management support for the proposed acquisition strategy.

  Context: Acquisition strategies may leverage prototypes, cloud computing, and/or existing or emerging software ecosystems. Acquisition strategies may be iterative or evolutionary.

Strategic Risk/Reward Analysis

- Evaluate risk/reward from various stakeholder perspectives, including the sponsoring organization, end-users, test and evaluation teams, cybersecurity compliance officers, and data rights managers.
- Balance system rewards when evaluating acquisition strategies that depend on externalities, such as commercial cloud computing or software ecosystems, against risks impacting overall system capability or system sustainment.

  Context: Solutions that rely on external dependencies such as commercial cloud computing or existing software ecosystems may offer rewards such as earlier deployment of capability to operations or lower resource expenditures (i.e., avoiding the "cost of delay"). However, the risks of depending on externalities may include the inability to modify the products to meet key requirements such as operations in denied environments, susceptibility to external system failures or malicious cyber behavior. Risks to system sustainment, such as vendor lock-in, may also exist.

Cloud Computing

- Identify risks (such as multilevel cybersecurity, vendor lock-in, or resources needed to operate and sustain DoD unique cloud platforms) and rewards (such as scale and elasticity for end-users and/or for test and evaluation teams) of cloud-based services.
- Evaluate options to acquire SaaS, PaaS, and/or IaaS.

Software Ecosystems

- Identify the risks and rewards from leveraging existing or emerging DoD, open-source, or third-party innovative technologies to support a shared end-user community. Consider how the ecosystem might evolve over time (including both planned and unplanned end of life), its responsiveness to emergent threats, its compatibility with existing capabilities and emerging software systems across DoD, and data/proprietary rights issues.
- Identify and evaluate alternative approaches to sustain the stability and fitness of the ecosystem over time (such as standardization of component interfaces or required regression testing).

- Proactively manage the expectations of the community the ecosystem supports.

Model-Based Engineering

- Create a digital engineering environment that uses models, prototyping, visualization, simulation, and dependency analyses to support the acquisition.
- Ensure that all models, prototypes, and analyses are fit for their intended purpose and not used for purposes for which they were not intended.

  Context: Digital engineering environments can be used to (1) engage end-users or other stakeholders in exploring alternate acquisition solutions, (2) explore and define the context (operational boundaries, key attributes and external dependencies) of proposed acquisition solutions, (3) mature key technologies, and/or (4) reduce system risk.

Development Tempo

- Determine the life-cycle approach to be used and the tempo of software construction, release, and deployment to operations.
- Synchronize the software construction, release, and deployment tempo(s) with system, hardware, and other environmental constraints (such as the stability of requirements; availability of developmental, test, or certification/accreditation resources; or disruption of end-user activities).
- Consider benefits and risks of adapting best practices from the Agile development community, such as continuous development (e.g., Kanban) or time-boxed iterations (e.g., Scrum or Agile Release Train), feature-driven development, test-first development, and pair programming.
- Plan for the transition of all artifacts, including data rights, from the development to the sustainment life cycle (e.g., staged versus continuous delivery) and environment (e.g., contractor to organic staff).
- Obtain support for the selected approach from program managers, system engineers, and software development leadership.

Release Planning

- Determine the minimum viable content of and completion criteria for (i.e., define "done") each release and/or software development iteration within the planning period.
- Consider the prioritized user needs and operational improvements (the product roadmap) while reserving resource margin to accommodate (1) expected and unexpected changes in the dependencies of the operational environment, including emergent security threats or supply chain issues, and (2) addressing technical debt buildup.
- Obtain support for the planned development and release approach from end-users as well as program management, system engineering, software development, test and evaluation (including certification authorities), and operations management.
- Reprioritize (i.e., groom) backlog(s) as required to ensure that high-risk/highly needed capabilities are developed early in the development cycle and to identify capabilities that are no longer needed.

Software Development Planning

- Develop methods, processes, and training needed for software construction (to include design, code, test, build, integration, and release). Identify tools and methods for backlog management, continuous integration, automated regression testing, and release management.
- Identify methods and processes for managing the software supply chain, to include periodic reevaluation of supplier viability and managing for COTS obsolescence.
- Assess the effectiveness of methods, processes, and training, and update as appropriate.

Planning for Continuous Delivery

- Identify cases where it may benefit the program to maintain software in a continual state of readiness for deployment into higher-level test facilities or into operations.
- Identity the methods (e.g., SecDevOps), processes, and training plans for automating the software release process (including fitness checks, integration tests, and acceptance tests) that

package the software for deployment into the operational or test environment(s).

Planning for Continuous Deployment

- Identify software items that could benefit from rapid delivery into operations (e.g., websites or application program interface updates).
- Consider the adverse impact incorrect or unstable software could have on the warfighter before deciding to implement a continuous deployment approach.
- Collaborate with the operations team to identify the methods, processes, and training plans for automating the delivery, including plans for "rolling wave" release, which minimize disruption of ongoing operations.
- Work with operations, accreditation, and certification teams to ensure the methods, processes, and training plans satisfy the conditions to grant authority to operate.
- Develop methods and processes to monitor newly deployed features and to rapidly roll back or otherwise mitigate unintended consequences of the deployment.

System Engineering Planning

- Develop methods, processes, and training needed to develop and evolve modern EAs and to perform dependency management (for both internal and external dependencies), validation, and verification activities (including accreditations or certifications) that are aligned to the software development life cycle, tempo, release planning, methods, and processes.
- Assess the effectiveness of methods, processes, and training, and update as appropriate.

  Context: Methods and processes may include (1) Iterative model-based engineering, analysis and visualization tools (i.e., digital engineering environments), (2) simulations, test beds, and software or system integration labs (with special consideration for how the minimum viable product will evolve), and (3) tools for issue tracking and version control.

Software Metrics

- Select metrics and measures to monitor software scope, cost, schedule, and quality (including performance) as needed to ensure program success, which may include quantitative methods (e.g., statistical control) to assess progress against quality goals (e.g., throughput, latency, availability, safety, security) or development baselines (planned versus actual features deployed to users or verified by test and evaluation teams).
- Select appropriate metrics at the team, program, and stakeholder levels.
- Reevaluate the timeliness and use of the selected metric(s), and eliminate those that are not timely and/or useful.

  Context: Team-level metrics may include measures of the velocity of requirement validation (e.g., requirement decision time) or velocity of software construction (e.g., story points/sprint, sprints/month). Program- and stakeholder-level metrics focus on progress against delivered stakeholder value (e.g., customer value streams, backlog burn down, and technical debt buildup).

Configuration and Version Control

- Develop strategies for identifying and managing the configuration of the system and software development and test environment(s), design, test, and analysis artifacts (including documentation) and the software itself (including external dependencies such as associated systems, the underlying hardware and software stack) throughout the life cycle.
- Develop methods, coding or naming standards, processes, and training plans for maintaining configuration and version control, including support of code repository branch and merge procedures, builds, and automated test suites (i.e., continuous integration), support for staging to test and production environments (i.e., continuous delivery) and, if feasible, delivery into operations (i.e., continuous deployment).

Software Documentation

- Document software planning, requirements, design, code, validation, verification, and sustainment needs in the program planning (e.g., Software Development Plans and Software Measurement Plans).

- Maintain versioned repositories for design, code, and test artifacts at all levels and across all phases of the life cycle.
- Use self-documentation tools for design, code, and test artifacts where practical.
- Continuously update and revise documentation standards to ensure the produced artifacts support their intended users and eliminate any unneeded artifacts.
- Use collaborative communication tools to ensure that planning and artifacts remain current and are easily accessible (e.g., wikis for team-wide communication and integrated development environments that link the product backlog to team member specific tasks and to artifact repositories).

Contracting for Software Development

- Ensure that contract requirements, constraints, end items, and data deliverables are compatible with the selected tempo, release planning, software and system development planning, metrics, and documentation requirements.
- If applicable, evaluate the use of alternate contracting mechanisms, such as time and materials, indefinite delivery/indefinite quantity, or OTAs, to provide needed flexibility in acquisition of the software or system.
- Identify appropriate incentives to manage technical debt buildup during iterative development.

Data and Proprietary Rights Management

- Negotiate data rights up front if elements of the software or system will be acquired from DoD-external sources (i.e., open-source repositories, COTs software, GOTs software, or from private entities) to ensure DoD will have assured access to all mission-critical software throughout the life of the supported system.
- Ensure that all software licenses are in compliance with federal regulations and compatible with program needs.

  Context: Program needs may include a perpetual and nonrevocable right to use the software over mission life; the right to inspect source code, design, analysis, and test artifacts to demonstrate security or safety compliance; and/or the right to create a derivative work as new system capabilities are added.

Architectural Design Approach

- Determine "how much" architectural design effort is needed to ensure a successful acquisition.
- Consider benefits and risks of adapting practices from modern architectural design methods such as Artifact Driven, Use/Abuse Case Driven, Attribute Driven, Domain Driven (i.e., Manage by Architecture), or Human-Centered Design when selecting an architectural design approach.
- Develop methods, processes, and training for the selected architectural design approach.
- Implement fitness functions (policies, procedures, tests) to periodically reassess the architecture's ability to satisfy system goals and constraints and to evaluate the architectural compliance of the acquired software.
- Develop and implement procedures to manage deviations and to effectively evolve the architecture over the program life cycle.

Context: The level of detail in architectural design will vary based on the proposed use of the architecture. These purposes may include (1) develop a common understanding between stakeholders and the development team of goals and objectives and the design approach to meeting them, (2) provide a narrative for current and future developers of the rationale used in making critical design decisions, and/or (3) provide a formal model used to validate the software's ability to satisfy system goals and constraints (such as modularity, open interfaces, safety, or multilevel security constraints).

Software Orchestration and Choreography Patterns

- Determine the patterns the software will use to manage the interactions between components, modules, or services.
- Consider common orchestration and choreography patterns (e.g., client/server, publish/subscribe, peer-to-peer, and services/microservices) that balance quality attributes for timing perfor-

mance (latency, throughput), evolvability, safety, and security in a manner consistent with the product vision.

- Evaluate how selected patterns may impact system resilience to naturally occurring failures or malicious attack for mission-critical systems.

Software Deployment Patterns

- Determine how the software will be deployed onto the computing infrastructure in the operational system.
- Determine which capabilities are required to be mobile (e.g., accessible to the warfighter even when on the move), which must be present in embedded systems or on local servers (e.g., accessible to the warfighter when disconnected from wide area networks), and which can be hosted on remote servers (e.g., cloud architecture, edge computing).
- Consider key attributes such as availability, maintainability, time performance (latency and throughput), safety, and security when making deployment decisions to ensure consistency with the product vision.

Artificial Intelligence and Machine-Learning Applications

- Identify and implement architectural components (such as "clear box" artificial intelligence), methods, processes, and training plans to mitigate the risks of incorporating artificial intelligence and machine-learning techniques to create autonomous cyber-physical systems, automated or augmented decision support tools, or other emerging AI-based systems.
- Balance correctness, transparency, explainability, privacy, availability, safety, and security when developing artificial intelligence and machine-learning applications.

Augmented and Virtual Reality Applications

- Identify and implement architectural components, methods, processes, and training plans that support the development and acquisition of augmented and virtual reality applications.
- Balance accessibility against safety when evaluating augmented and virtual reality applications.

- Consider human factors such as color blindness or other sensory factors, navigational perceptions, and general fitness for use.

Embedded Systems

- Employ explicit strategies for incremental realization of capabilities within the constraints of the hardware supply chain.
- Identify and implement architectural components (such as application program interfaces or other mediating elements), methods, processes, and tools (such as simulators or models) to mitigate the risks of hardware/software co-development and/or obsolescence.
- Consider potential impacts of asynchronous hardware and software development life cycles when acquiring embedded applications and tools.

Balancing Quality Attributes

- Evaluate alternative design solutions to effectively balance the quality attributes for critical mission threads or other identified scenarios.
- Identify architecturally significant quality attributes (e.g., throughput, latency, evolvability, safety) using scenario-based elicitation techniques (such as quality attribute workshops) to engage critical stakeholders (including owners, operators, and users), and develop a common understanding of the relative importance of each quality attribute to the mission drivers and product vision.
- Define fitness functions that provide an objective measure of the key attributes, and consider building an executable architecture or digital engineering environment to evaluate design alternatives.

Emerging Technologies

- Maintain an understanding of emerging technologies, the implications these technologies may have on a given organizational need and solution space, and their technical maturity/readiness or certification.
- Evaluate both the possible benefits and risks of incorporating emerging tech, including its ability to disrupt opponents, the U.S. warfighters and our allies, external systems, and the product architecture itself.

Use/Abuse Case Modeling

- Use static and dynamic views to model the software components that implement the required capabilities of the software to identify the use cases (i.e., the required uses of the system over the entire life cycle, including operations and sustainment) and abuse cases (e.g., critical system failures, malicious user inputs, response to attacks) for the system to be acquired.
- Develop and evolve models as needed to explore boundary conditions and interactions among domains, capabilities, quality factors (such as robustness and fault tolerance), activities, and entities that comprise the acquired software.

Validation of Performance Efficiency Requirements

- Implement models of the software that reflect the concurrency management, event handling, and data persistence approaches used in the software.
- Evolve the models as needed to track the "as-built" software design and implementation.
- Validate the capability to meet performance efficiency requirements (with margin as appropriate to the life-cycle phase) under realizable nominal, best, and worst-case conditions for each mission-critical thread.

Validation of Sustainability Requirements

- Implement models to explore sustainability features of the software architecture with consideration for specific needs associated with high availability or safety-critical systems.
- Consider modeling use cases for updates of software, data, and operating procedures, and write test cases to verify those capabilities.

High-Fidelity System Modeling

- Create a digital, high-fidelity representation of the as-built system that reflects lessons learned in test or operations to support the analysis of critical quality attributes.
- Use the model to verify critical attributes such as safety, security, reliability, sustainability, and/or performance efficiencies (e.g.,

system recovery time) under worst-case conditions that cannot be verified via test or demonstration.

Context: Digital twins is one method for gathering pertinent information from operations for use in the model.

Software Assurance

- Specify and implement methods, processes, and tools needed to assure the integrity of the acquired software.
- Determine appropriate coding standards, static and dynamic analysis rules, test code coverage, and fuzz testing standards.
- Determine the corrective actions to be taken when code or test standards are not met, and/or when analysis rules are violated. Track all actions to closure.
- Obtain support for the selected approach(es) from all external accreditors (e.g., NSA, Army Cross-Domain Management Office), program quality assurance managers, system safety or security organizations, and software development leadership.

Cybersecurity

- Specify and implement methods, processes, tools, and models to improve the cybersecurity of the acquired software, including procedures to assess the pedigree of reuse, open-source, and commercial software.
- Evaluate cybersecurity attributes to include confidentiality, integrity, availability, and nonrepudiation. Determine the importance of auditability, redundancy, and resilience attributes to respond to potential cyberattacks.
- Identify the key security components of the architecture (such as whitelists, audit traces, and multilevel security guards), and specify the methods and processes that will be used to assure their integrity throughout the program life cycle.
- Develop and execute verification activities such as penetration and/or red-team testing.

Safety Critical Systems

- Specify and implement methods, processes, tools, and models (the elements of a safety culture) to identify, mitigate, and/or remove hazards from the acquired software and the systems it supports.

- Apply Military Standard 882E,[1] Hazard Identification and Probability vs. Severity Risk Assessment method on all potential safety hazards.
- Use available best practices and/or required standards (such as Document-178C[2] for aircraft) to increase the safety of the operational software and its supported systems.

  Context: Relevant to safety-critical systems (e.g., aircraft, nuclear systems, ground combat systems, missile systems, space systems) or portions of systems (e.g., deployment mechanisms that interface with live ordnance).

High-Availability Systems

- Specify and implement methods, processes, tools, and models (attributes of a high-reliability culture) to improve (1) the integrity, defect tolerance, and resilience of the acquired software and, (2) the integrity, fault tolerance, and resilience of the systems it supports.
- Establish service-level indicators and objectives to measure reliability/stability of the software and system from the user perspective over time (this includes identifying user-defined mission-critical threads, stressing test cases such as max load) in off-nominal conditions, and having actual users demonstrate their standard operating procedures and field deviations.
- Use available best practices (such as Service-Level Agreements) to monitor and improve the availability of the system.

Life-Cycle Management

- Evaluate the effectiveness of the selected life cycle, methods, processes, and tools against program outcomes using both quantitative and qualitative measures.
- Identify timing, content, and stakeholders for retrospective reviews.

[1] DoD Military Standard 882E. *System Safety*, May 11, 2012.

[2] Radio Technical Committee for Aeronautics Document 178C, *Software Considerations in Airborne Systems and Equipment Certification*, December 2011.

- Update plans as necessary to address obsolete or emerging technologies, methods, processes, and tools.

  Context: Retrospective reviews are team-driven events that include all stakeholders to evaluate the effectiveness of the acquisition against program outcomes. They are typically synchronized to the development tempo and program schedule (e.g., continuous development would schedule reviews by calendar, while Scrum would perform retrospectives at the conclusion of each sprint).

Detailed Backlog Management

- Maintain a list of capabilities to be developed (aka, the product backlog) and the tasks that are required to realize those capabilities mapped to the release plan.
- Add tasks as they are identified throughout the life cycle, including tasks to resolve defects found during design, code, test, and operations.
- Define and clearly communicate the "ready" and "done" criteria for each task, developing a shared understanding with all stakeholders.

  Context: Fine-grained tasking, linked to capabilities, may be required to synchronize activities across development teams (e.g., "Scrum of Scrums" concept), validation and verification teams (e.g., software integration or certification facilities), or customer support teams.

Release Management

- Use the "done" criteria from the release planning to identify the required verification steps (inspection, analysis, unit, integration, or acceptance test) for each release to higher levels of integration testing, certification activities, and/or operations.
- Synchronize software releases with the development of models, simulations, test beds, and operations environment(s) as needed to ensure compatibility.
- Review and revise the release planning in response to changes in program needs.

Change Management

- Implement methods, processes, and tools to manage changes to program planning, requirements, architectural design decisions, code, as well as validation and verification artifacts.
- Implement mechanisms to ensure that decisions regarding proposed and approved changes are communicated clearly and in a timely fashion to all stakeholders (i.e., at the speed required by the development tempo).
- Implement a closed-loop process to track changes to closure.

Automated Test and Continuous Integration

- Implement a continuous integration process to ensure that newly written code is compatible with existing code.
- Consider implementing methods, processes, and tools as part of the continuous integration process to write and execute test cases that reflect the completion criteria for each release, capability, microservice, or component of the software being acquired.
- Automate the tests (from unit tests to system integration tests) when feasible to allow for rapid discovery of integration issues. Identify a subset of the test to function as a "smoke test" for daily or on-demand builds of the software.
- Ensure the integrity of the automated test environment, and validate that it performs as expected.

Effort Estimation

- Create and maintain an estimate of the total software acquisition effort (labor and material), accounting for software size, complexity, precedent, team cohesion, and the development team's direct experience with the application, methods, and processes to be used.
- Use parametric, historical comparisons (analogies) and bottom-up effort estimates from the development team, as appropriate, to support business case development and acquisition strategy refinement.
- Revise the acquisition strategy (this may include program cancellation or a redefinition of the minimum viable product) if effort estimates exceed defined thresholds.

Product Roadmap and Schedule Management

- Implement plans for development (and/or purchase or sustainment); and manage the schedule associated with capability/feature development and release (i.e., the product roadmap) against scope, cost, and policy.
- Monitor the velocity of actual software and warfighter value production.
- Monitor early indicators of schedule risks, such as schedule buffer (i.e., distributed schedule float) consumption and cumulative flow metrics.
- Update effort estimates, cost, and schedule baselines as appropriate.

Cost Management

- Implement plans for development (and/or purchase or sustainment); and manage the cost of the acquisition against constraints in scope, schedule, and policy.
- Monitor actual software production metrics versus labor and material expenditures, and update effort estimates and cost baselines as needed. Utilize cost management reserve to mitigate development risks.
- Ensure that the formulation of "value earned" for software development tasks accounts for (1) the accumulation of technical debt and (2) the evolution of the product backlog when using earned value measurement as a tool for monitoring program cost and schedule.

Legal Policy and Regulatory Environment Management

- Implement plans for development (and/or purchase or sustainment), while considering and adhering to relevant laws, congressional budgets (fiscal year funding constraints), regulations and certification requirements, and policies (e.g., data and property rights, ownership, export rules) and managing against constraints in scope, cost, and schedule.
- Update program plans in response to changes in legal policy and regulatory environments.

Context: Examples of relevant laws includes Buy America Act[3] and Title 10.[4]

Risk, Issues, and Opportunity Management

- Identify, specify, and execute strategies for actively managing project risks, to include risk identification, quantization, and implementation of approved mitigation strategies (e.g., active management of the system accreditation process for both operation and test/evaluation; synchronization of software, hardware, and system development).
- Implement a closed-loop process to actively track issues as they arise, identify opportunities for improving products and processes that add to customer value, and continuously reassess program plans to mitigate risks and realize opportunities.

Quality Assurance

- Review/audit the system, software, and supply chain's processes and products; and team, program, and stakeholder metrics to assess ability to meet the acceptance criteria for the delivered product and provide an independent assessment of quality to program management.
- Establish criteria for reviewing and auditing the software supply chain across all subtiers as necessary to ensure program success.

  Context: Team-level metrics may include measures of software construction (e.g., story points/sprint, sprints/month), integration testing, or software quality. Program- and stakeholder-level metrics focus on progress against delivered stakeholder value (e.g., customer value streams, backlog burn down, and technical debt buildup).

Root Cause, Corrective Action

- Monitor the program and software metrics to identify early indicators of adverse trends and determine root causes.

[3] United States Code, Title 41, Chapter 83, "Buy American."

[4] United States Code, Title 10, Armed Forces.

- Use statistical control or other methods to proactively propose changes to methods, processes, and software to correct those trends.

System Integration and Testing

- Implement and execute system integration test plans and procedures, to include both informal and formal validation, that the software meets the needs of its users and verification that the software and/or system meets its requirements.
- Document the test results.
- Automate integration and test activities to the extent practical, and build them into the software release process.
- Analyze the results to identify early indicators of adverse trends; and implement root cause, corrective action.

Strategic Planning and Change Management

- Take a long-term view and build a shared vision with others; act as a catalyst for organizational change.
- Influence others to translate strategic planning into action.
- Apply change management principles, strategies, and techniques needed to effectively plan, implement, and evaluate change in the organization.
- Develop an understanding of the impacts that change may have on people, processes, procedures, leadership, and organizational culture and the impacts that organizational culture and generational conflict may have on the ability to achieve change.

Innovation and Entrepreneurship

- Provide transformational solution-based approaches to problem-solving and building products by employing an Iterative process of empathize, define, ideate, build/prototype, and test (i.e., design thinking); and institute a culture that encourages early and continuous learning (i.e., fail early/learn early).

Glossary

These definitions started from standard terms of art in the software industry. Where formal standards exist for specific terms, those standards are referenced. The definitions are part of the competency model and were refined using the same processes described in this report.

artificial intelligence: The theory and development of computer systems able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, optimization, and decisionmaking. *Explainability* is a key quality attribute for artificial intelligence applications and is a prerequisite to building trust between humans and machines.

backlog: A prioritized list of tasks that teams need to work on within the scope of an acquisition strategy and life-cycle approach. It is derived from an analysis of users' stories, quality attributes, and feedback from actual users and testers. To adapt to emerging needs, the backlog is periodically reviewed to add new items, reprioritize, and drop unneeded or low-priority items. This process is termed *backlog grooming*.

cloud computing: A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Clouds can be private or public and generally take the physical form of large server farms connected by high-speed digital networks.

continuous delivery: An automated set of audits and tests to ensure software is in a continual state of readiness for deployment into external testing or operations. This includes packaging the software for deployment in a production-like environment (e.g., containers). In Agile development, the steps that comprise continuous integration and delivery are referred to as a *pipeline*.

continuous deployment: A process that automatically deploys the results of continuous delivery into the final production environment, usually hundreds of times per day.

continuous integration: An automated software merge, build, and integration cycle to ensure that newly submitted code is compatible with existing code. Can include automated static and dynamic analyses and regression tests. At minimum, continuous integration is performed once per day (i.e., the nightly build). The cycle may also be triggered automatically for each code commit.

cumulative flow: A metric that tracks the growth (or shrinkage) of work as it progresses through different states over time. For software, cumulative flow diagrams provide an immediate visual indicator of lead time and cycle time for work in progress and are an effective technique for identifying bottlenecks in the software construction process, such as the limited availability of specific skill sets, integration facilities, or certification authorities.

decision support tools: Interactive software-based systems intended to help decisionmakers compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make business decisions. A well-designed decision support tool provides intelligently filtered data at the appropriate time in the human decisionmaking process.

design thinking: An Iterative method to engage in solution-focused thinking with the intent of producing a constructive future result. Especially useful when both the problem and the solution are unknown. The goal of design thinking is to discover the parameters of the problem that matter.

digital engineering: An integrated digital approach that uses authoritative sources of system data and models as a continuum across disciplines to support life-cycle activities from concept through end of life.

digital twin: An integrated model of an as-built/as-deployed system that is intended to reflect all defects and failures encountered in operations and to capture the evolving operational environment. It is a sensor-enabled digital model that simulates the deployed system in a live setting. The sensors collect cumulative, real-time, real-world data across an array of dimensions to create an evolving digital profile of the historical and current behavior of the system.

development tempo: The frequency and timing of software development iterations, releases to test or certification organizations, and deployments into hardware or operations. Tempo must be tailored to the needs of the program and to the different levels of integration required—no one size fits all.

ecosystem: A set of entities functioning as a unit and interacting with a shared end-user constituency for software and services, together with relationships among them. Ecosystems form when a set of core components (the keystone) are complemented by peripheral components (e.g., apps or services) developed by autonomous entities (i.e., organizationally independent of the core developer) to address specific user needs. Ecosystems are characterized by interoperability and co-innovation enabled through common interfaces and shared knowledge.

executable architecture: A description of a system architecture in a formal notation together with the tools that allow the generation of artifacts from that notation, which are then used in the analysis, refinement, and/or the implementation of the architecture described.

fitness functions: An objective integrity assessment of a quality attribute of the software. The integrity assessment should be conducted continuously or triggered periodically to ensure that the software architecture evolves in a direction that meets the system's key objectives. Fitness functions can be manual, such as a quarterly review by the architect to review for modularity and open interfaces, or automated. Automated fitness functions are often built into the continuous integration and/or delivery pipeline. Static analysis for cyclomatic complexity or directionality of package imports in Java are examples of fitness functions that evaluate modularity and coupling. Testing for system resilience using chaos engineering principles is another example of a fitness function.

fuzz testing: An automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a software item and verifying that the resultant behavior of the software is consistent with the assumptions used in the design, validation, and verification of safety-critical, security-critical, or high-availability systems.

IaaS: Provides flexible computing resources for data storage, memory, or computing power.

ideate: An "idea generation" process characterized by alternating between divergent and convergent thinking. Divergent, or "out of the box," thinking creates a wide variety of proposed problems and solutions, while convergent thinking seeks to narrow the proposed problems/solution space to a best or acceptable alternative.

integrity: The quality of an information system reflecting the logical correctness and reliability of the operating system, the logical completeness of the hardware and software implementing the protection mechanisms, and the consistency of the data structures and occurrence of the stored data. When used in this report in the specific context of cybersecurity, integrity is interpreted more narrowly to mean protection against unauthorized modification or destruction of information (i.e., data integrity). This definition, including the narrower one used for cybersecurity, are per the Committee on National Security Systems (CNSS) Instruction 4009.[5]

life-cycle approach: The sequencing of activities within each iteration or for each software release. Activities may include requirements analysis, design (architectural and detailed), validation analysis and modeling, implementation, static and dynamic analysis of the code, verification (via inspections, analysis, or test at various levels of integration), build and integration, delivery, and deployment of the software. Life cycle approaches should evolve as constraints change (e.g., the life cycle for an initial greenfield development may differ significantly from that used when sustaining the mature product).

machine learning: A category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. These algorithms receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available. Commonly used machine-learning algorithms (in order of increasing sophistication) include: correlators, optimizations using decision trees, K-means clustering, and neural networks.

[5] CNSS Instruction 4009, April 26, 2010.

minimum viable product: The minimum set of features that provides value to a user. It is a core artifact in an Iterative process of idea generation, prototyping, presentation, data collection, analysis, and learning. The process is iterated until a desirable product/market fit is obtained, or until the product is deemed nonviable.

PaaS: Provides the underlying software that makes cloud infrastructure accessible to individual applications. PaaS differs from IaaS in that it provides more of the software application stack, and from SaaS in that it does not provide the complete software stack. A wide range of PaaS constructs is used in support of DoD programs.

performance efficiency requirements: Per International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 25010:2011,[6] includes timing characteristics (e.g., responsiveness and latency), capacity and resource usage (e.g., memory and throughput).

product roadmap: A plan that matches short-term and long-term business goals with specific technology solutions to help meet those goals. It is a strategic mapping of how the enterprise will evolve over time.

quality attributes: Nonfunctional requirements used to evaluate the performance of a system. These are sometimes named "ilities" after the suffix many of the words share. They are usually architecturally significant requirements that require architects' attention.

resilience: A measure of how quickly the system recovers essential services after failure or attack.

risk/reward analysis: Used to select among different acquisition strategies when actual costs and benefits cannot be determined. Plots risk versus a range of possible reward over the product life cycle to select the strategy that has the greatest potential to provide rewards while minimizing risks.

[6] International Standard ISO, *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, ISO/IEC 25010–2011, International Organization for Standardization ISO, 2011.

root cause analysis: Tools that promote an understanding of the underlying (as opposed to surface) causes of systemic issues. Two common techniques for understanding the root cause of a series of related issues are the "5 Whys" and the "Fishbone Diagram."

SaaS: Provides access to software applications without having to purchase and install those application on individual devices.

schedule buffer: An allocation of development resources (labor and time) to accommodate changes in system requirements, resolution of software defects, and reductions in technical debt. Schedule buffers should be placed prior to all critical integration points to mitigate the risk of late software deliveries to embedded hardware, external interface testing, and end-to-end verification activities.

software construction: A software engineering discipline to create working, meaningful software through a combination of user-focused validation and modeling, architecture, design, coding, analysis, unit testing, integration testing, and debugging. Higher quality can be achieved when software construction is structured to minimize the calendar time from initial start of a capability to delivery of usable software.

software development environment: The suite of tools that are used to aid in software development. These include tools used to enhance team collaboration; modeling tools that often link directly to code generators; syntax checkers, compilers, linkers, and debuggers that provide feedback in real time to human coders; change management tools; and continuous integration and delivery tools. As these latter tools have expanded in scope and complexity, the software development environment has come to be increasingly referred to as the *software development ecosystem*. Another term coming into usage to describe both the tools and the processes that/humans who use them is the *software factory*.

software quality: This document uses the term *software quality* as defined in International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC) 25010:2011, which includes all nonfunctional attributes, including performance efficiency (timing, resource usage), reliability, usability, maintainability, security, safety, etc.

success case method: Involves identifying the most and least successful cases in a program and examining them in detail. It is a useful approach to document stories of impact and to develop an understanding of the factors that enhance or impede impact.

technical debt: A concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer. Its buildup is used as an indicator and warning for system development risk.

technologies: Includes architectures, algorithms, software, methods, processes, tools, and techniques that have the potential to transform warfighting or the velocity and quality of deployment of DoD systems.

validation: An assessment of whether the delivered system will or does meet user needs (i.e., "Are we building/did we build the right thing?" "Did we specify the capabilities and quality attributes that the warfighter actually needs?"). Validation can be conducted through user stories, quality attribute workshops, modeling and simulation, integration testing, and feedback from users regarding the operation of prototypes or early releases in operational tests and the operational environment itself. Systems and the services they deliver to the warfighter must continually evolve as warfighter needs evolve.

value perception model: A conceptual model that defines and relates price, perceived quality, and perceived value. Price is used as a proxy to value-proposed acquisition strategies.

verification: An assessment of whether the system implementation satisfies the identified capabilities and quality attributes (i.e., "Did we build it right?"). Verification activities include inspection, analysis, demonstration, and test and are conducted at various levels of integration. To the extent practical, verification activities should be automated within continuous integration, delivery, and deployment pipelines to provide rapid feedback to the development and sustainment teams.

APPENDIX G

List of Courses Reviewed

This appendix lists the courses reviewed for this study and discussed in Chapter Six. Tables G.1, G.2, and G.3 detail the 394 courses provided by DAU, other DoD institutions, and civilian institutions, respectively.

Table G.1
Courses Provided by the Defense Acquisition University

Fundamentals of Systems Acquisition Management	Capability Maturity Model-Integration (CMMI)	Applied Systems Engineering in Defense Acquisition, Part II
Fundamentals of Technology Security/ Transfer (FTS/T)	Intellectual Property and Data Rights	Fundamentals of Earned Value Management
Intermediate Systems Acquisition, Part A	Cybersecurity Throughout DoD Acquisition	Basic Information Systems Acquisition[a]
Intermediate Systems Acquisition, Part B	Introduction to DoD Cloud Computing	Intermediate Information Systems Acquisition[a]
Mission-Focused Services Acquisition (R)	Introduction to Agile Software Acquisition	Advanced Enterprise Information Systems Acquisition[a]
Understanding Industry (Business Acumen) (R)	Reliability and Maintainability	Advanced Program Information Systems Acquisition[a]
Advanced Technology Security/Control Workshop	Supportability Test and Evaluation	Acquisition Logistics Fundamentals
Acquisition Law	Life-Cycle Logistics for the Rest of Us	Fundamentals of System Sustainment Management
Forging Stakeholder Relationships (R)	Developing a Life-Cycle Sustainment Plan (LCSP)	Reliability, Availability, and Maintainability (RAM)
Fundamentals of Cost Analysis	Designing for Supporta-bility in DoD Systems	Product Support Strategy Development, Part A
Acquisition Reporting for MDAPs and MAIS (R)	Supportability Analysis	Configuration Management
Operating and Support Cost Analysis (R)	Product Support Business Case Analysis (BCA)	Performance-Based Logistics
Applied Software Cost Estimating	Sustainment of Software Intensive Systems	Enterprise Life-Cycle Logistics Management
Cost Analysis	Diminishing Manufacturing Sources and Material Shortages (DMSMS) Fundamentals	Program Management Tools Course, Part I
Program Execution	Cost Estimating	Program Management Tools Course, Part II
Planning, Programming, Budgeting, and Execution and Budget Exhibits	Risk Management	Program Manager's Skills Course

Table G.1—Continued

Software Cost Estimating	Commercial-Off-the-Shelf (COTS) Acquisition for Program Managers	Production, Quality, and Manufacturing Fundamentals
Value Engineering	Environmental Safety and Occupational Health	Intermediate Production, Quality, and Manufacturing, Part A
Technical Reviews	Introduction to Data Management	Intermediate Production, Quality, and Manufacturing, Part B
Introduction to Lean Enterprise Concepts	Data Management Strategy Development	Core Concepts for Requirements Management
Environment, Safety, and Occupational Health in Systems Engineering	Data Management Planning System	Introduction to Science and Technology Management
DoD Open Systems Architecture	Technical Data and Computer Software Rights	Intermediate Science and Technology Management
Continuous Process Improvement Familiarization	Data Management Protection Storage	Fundamentals of Test and Evaluation
Technical Planning	Quality Assurance Auditing	Intermediate Test and Evaluation
Technology Readiness Assessments	Introduction to the Joint Capabilities Integration and Development System	Advanced Test and Evaluation
Program Manager Introduction to Anti-Tamper	Analysis of Alternatives	DISA Information Systems Engineering Workshop (ISEW)
Modeling and Simulation in Test and Evaluation	Introduction to Earned Value Management	Systems Engineering Plan (SEP)
Engineering Change Proposals for Engineers	Software Acquisition Management (SAM) Policy and Procedures	Engineering Management Workshop (EMW)
Software Reuse	Software Acquisition Management (SAM) Policy Implementation	Reliability and Maintainability (R&M) for Engineers
Practical Software and Systems Measurement	Fundamentals of Systems Engineering	Intellectual Property (IP) Workshop
Human Systems Integration	Applied Systems Engineering in Defense Acquisition, Part I	Cybersecurity Awareness Workshop

NOTE: These are the ISA courses used to identify gaps in the DAU software acquisition curriculum.

Table G.2
Courses Provided by Department of Defense Institutions

Air Force Institute of Technology		
System Software Engineering	Introduction to Software Engineering	Software Architecture and Design Methods
Project Management	Software Project Management	Software Implementation Techniques
System Architecture	Software Requirements Management	Software Test Engineering
Human-Computer Interaction	Software Architecture and Design Management	Software Deployment and Sustainment Techniques
Decision Analysis	Software Implementation Management	Current Software Technology Topics
Data Security	Software Test Management	Introduction to Configuration Management
Secure Software Design and Development	Managing Software Deployment and Sustainment	Introduction to Systems Engineering
Software Evolution	Current Software Acquisition and Management Topics	Technology Readiness Assessment (TRA)
Information Visualization	Software Requirements Engineering	Management of the Systems Engineering Process
Advanced Topics in Software Engineering		
Air University		
Principles of Computer Operation	Introduction to Computer Networks	Cyber Surety Management
Data Processing, Inquiry, and Retrieval Systems	Software Engineering	Data Retrieval Systems
Operational Systems Utilities	Principles of Database Applications	Computer System Administrator
Introduction to System Software	Introduction to Logistics Automated Data Systems	Advanced Data Inquiry and Retrieval
Computer System Familiarization	Software Engineering II	Database Applications Programming
Contracting Computer Applications	Cyber Surety	Computer Systems Security

Table G.2—Continued

National Defense University		
Information Technology Program Management	Enterprise Architectures for Leaders	
Strategic Information Technology Acquisition	Decisionmaking for Government Leaders	
Capital Planning and Portfolio Management	Data Management Strategies and Technologies: A Managerial Perspective	
Strategies for Assuring Cyber Supply Chain Security	Information Assurance and Critical infrastructure Protection	
Emerging Technologies	Enterprise Information Security and Risk Management	
Naval Postgraduate School		
Software Methodology	Introduction to Computer Security	Intermediate Programming
Software Reliability	Advanced Software Engineering	Human-Computer Systems Interaction
Acquisition of Defense Systems	Principles of Software Design	Computer Architecture
Project Management for Enterprise Systems	Software Testing	Computer Communications and Networks
Introduction to Formal Methods in Software Engineering	Software Architecture	Enterprise Architecture
Software Engineering Research and Development in DoD	Introduction to Programming	Enterprise Systems Analysis and Design
Requirements Engineering	Fundamentals of Computing Systems	Enterprise Database Management Systems

Table G.3
Courses Provided by Civilian Institutions

California Institute of Technology		
Managing the Software Component of Projects	Agile Project Management Certificate Program	Software Engineering and Management
Carnegie Mellon University		
System Architectures for Managers	Distributed Embedded Systems	
Architectures for Software Systems	Real-Time Embedded Systems	
Agile Software Development Frameworks: Theory	Models of Software Systems	
Agile Software Development Frameworks: Practice	Communication for Software Engineers I	
Deciding What to Design	Communication for Software Engineers II	
Management of Software Development for Technology Executives	Embedded System Software Engineering	
Business for Engineers	Engineering Run-Time Malware Detection	
Managing Software Development	Analysis of Software Artifacts	
Cornell University		
Computer System Organization and Programming	Object-Oriented Programming and Data Structures	
Operating Systems	Programming Languages and Logics	
Data Structures and Functional Programming	System Security	
Introduction to Database Systems	Software Engineering	
Embedded Systems	Open-Source Software Engineering	
George Mason University		
Reusable Software Architectures	Secure Software Design and Programming	Security Policy
Service-Oriented Architecture	Software Requirements Analysis and Specification	Distributed Software Engineering
Quality of Service for Software Architectures	Component-Based Software Development	Software Project Laboratory

Table G.3—Continued

Database Management	Management Science	Software Analysis and Design of Real-Time Systems
Database Systems	Software Project Management	Software Engineering for the World Wide Web
Object-Oriented Software Specification and Construction	Information Security Theory and Practice	Software Testing
Software Modeling and Architectural Design	Operating Systems Security	Advanced Software Testing
User Interface Design and Development		
Georgia Institute of Technology		
Advanced Computer Architecture	Software Fundamentals for Engineering Systems	
Engineering Software Design	Programming for Hardware/Software Systems	
Massachusetts Institute of Technology		
Computation Structures	Automata, Computability, and Complexity	Elements of Software Construction
Database Systems	Introduction to Algorithms	Computer System Engineering
Artificial Intelligence	Fundamentals of Programming	Computer Language Engineering
Introduction to Machine Learning	Design and Analysis of Algorithms	Performance Engineering of Software Systems
Princeton University		
Operating Systems	Human-Computer Interface Technology	Functional Programming
Computer Architecture and Organization	Machine Learning and Artificial Intelligence	Programming Languages (COS 441)
Database and Information Management Systems	Compiling Techniques	Programming Languages (COS 510)

Table G.3—Continued

Purdue University		
Operating Systems (CS 35400)	Database Systems	Algorithm Design, Analysis, and Implementation
Operating Systems (CS 50300)	Compilers: Principles and Practice	Software Engineering I
Information Systems	Compiling and Programming Systems	Software Engineering II
Data Communication and Computer Networks	Programming Languages	Software Testing
San Jose State University		
Computer Organization and Architecture	Enterprise Distributed Systems	Introduction to Programming
Cloud Technologies	Enterprise Application Development	Assembly Language Programming
Introduction to Data Structures	Global and Social Issues in Engineering (ENGR 195A)	Information Security
Data Structures and Algorithms	Global and Social Issues in Engineering (ENGR 195B)	Introduction to Engineering
Introduction to Database Management Systems	Virtualization Technologies	Software Engineering I
Data Mining	Computer Networks I	Software Engineering II
Large Scale Analytics	Enterprise Software Platforms	Software Engineering Process Management
Senior Design Project I	Computer Network Design	Software Engineering Processes
Senior Design Project II	Network Programming and Applications	Software Engineering Management
Operating Systems	Cloud Services	Software Quality Engineering
Object-Oriented Design	Network Architecture and Protocols	Software Quality Assurance and Testing
Computer and Human Interaction	Network Security	

Table G.3—Continued

University of California, Berkeley		
Computer Architecture and Engineering	Database Management	Operating Systems and System Programming
Graduate Computer Architecture	User Interface Design and Development (COMPSCI 160)	Programming Languages and Compliers
Introduction to Database Systems	Introduction to Embedded Systems	Design of Programming Languages
Principles and Techniques of Data Science	User Interface Design and Development (COMPSCI 260A)	Computer Security
Introduction to Database Systems	Human-Computer interaction Research	Security in Computer Systems
Implementation of Database Systems	Software Prototyping for Data Science and Information Management	Software Engineering
Information Organization and Retrieval	Needs and Usability Assessment	
University of Illinois, Urbana-Champaign		
Embedded Systems	Distributed Systems	Programming Language Semantics
Computer System Organization	Formal Software Development Methods	Software Engineering I
Computer Architecture	System Programming	Software Engineering II
Database Systems	Programming Languages and Compilers	Topics in Software Engineering
Data Structures	Programming Language Design	Program Verification
University of Michigan, Ann Arbor		
Computer Networks	Advanced Operating Systems	Principles of Real-Time Computing
Database Management Systems	Foundations of Artificial Intelligence	Computer and Network Security
Introduction to Operating Systems	Compiler Construction	Software Engineering
Web Systems	Programming Languages	

Table G.3—Continued

University of Texas at Austin		
Software Architecture	Communication Networks: Tech/Arch/Protocol	Algorithmic Foundations for Software Systems
Data Engineering	Distributed Systems	Introduction to Optimization
Data Mining	Parallel Algorithms	Requirements Engineering: Acquisition and Modeling
Formal Methods in Distributed Systems	Systems Programming	Distributed Information System Security
Mobile Computing	Advanced Programming tools	Middleware
Social Computing	Multicore Computing	Verification and Validation
Advanced Embedded Microcontroller Systems	Computer Graphics	Software Testing
System Engineering Program Management and Evaluation		
University of Washington		
Computer Architecture	Computer-Aided Reasoning for Software	Advanced Topics in Programming Languages
Advanced Topics in Software Systems	Advanced Topics in Human-Computer Interaction	Programming Language Analysis and Implementation
Computer Operating Systems	Software Development for Data Scientists	Principles of Programming Languages
Computer Systems Architecture	Machine Learning/Data Mining	Advanced Topics in Programming Languages
High-Performance Computer Architectures	Applications of Artificial Intelligence	Computer Security and Privacy
Computer Systems	Machine Learning	Principles of Software Engineering
Operating Systems	Artificial Intelligence	Software Engineering
Database Management Systems	Artificial Intelligence II	Advanced Topics in Software Engineering
Scalable Data Systems and Algorithms	Compiler Construction	Performance Analysis (CSEP 597)
Principles of Database Systems	Programming Languages	Performance Analysis (CSE 597)
Human-Computer Interaction		

APPENDIX H

Software Curriculum–Competency Mapping

Table H.1 shows potential overlap between information systems acquisition (ISA) courses and each competency. ISA courses were evaluated by each course's manager. There following four ISA courses were included in the review:

- ISA 101—Basic Information Systems Acquisition
- ISA 201—Intermediate Information Systems Acquisition
- ISA 301—Advanced Enterprise Information Systems Acquisition
- ISA 320—Advanced Program Information Systems Acquisition.

Green indicates that there is full coverage of the competency. Yellow indicates that there is partial coverage, and red indicates no coverage. We also provide a column to sum the total number of ISA courses that provide full coverage of the competency. A RAND SME also evaluated other DoD and civilian courses to identify potential courses that could be used to address gaps in DAU's ISA curriculum. For each competency, we provide a count of the number of other DoD courses and civilian courses that appear to provide good coverage of competency content. This analysis should be viewed as exploratory and reevaluated once the competencies have been validated and a workforce analysis conducted. These steps are necessary to identify the relative importance of competencies, the level of proficiency required on each competency, and potential proficiency gaps in the workforce.

Table H.1
Competencies Covered by Defense Acquisition University Information Systems Acquisition Courses and Other Department of Defense and Civilian Courses

Competency	ISA 101	ISA 201	ISA 301	ISA 320	# Green	Other DoD Courses	Civilian Courses
Legal Policy and Regulatory Environment Management	🟢	🟢	🟢	🟢	4	2	1
Risk, Issues, and Opportunity Management	🟢	🟢	🟢	🟢	4	3	0
Cybersecurity	🔴	🟢	🟢	🟢	3	3	2
Business Case Development	🔴	🟡	🟢	🟢	2	9	12
Contracting for Software Development	🔴	🟡	🟢	🟢	2	3	0
Life-Cycle Management	🔴	🟡	🟢	🟢	2	0	3
Release Planning	🔴	🟢	🔴	🟢	2	1	2
Strategic Planning and Change Management	🔴	🟢	🟢	🔴	2	2	1
Cloud Computing	🔴	🟢	🟢	🔴	2	0	3
Data and Proprietary Rights Management	🔴	🟢	🔴	🟢	2	2	1
Quality Assurance	🟢	🟢	🔴	🔴	2	0	3
Release Management	🔴	🟢	🔴	🟢	2	0	3
Strategic Risk/Reward Analysis	🔴	🔴	🟢	🟢	2	9	3
System Integration and Testing	🔴	🟢	🔴	🟢	2	1	2
Architectural Design Approach	🔴	🟡	🟢	🔴	1	0	3
Configuration and Version Control	🔴	🟡	🔴	🟢	1	1	2
Planning for Continuous Delivery	🔴	🟡	🔴	🟢	1	3	1
Planning for Continuous Deployment	🔴	🟡	🔴	🟢	1	3	3

Table H.1—Continued

Competency	ISA 101	ISA 201	ISA 301	ISA 320	# Green	Other DoD Courses	Civilian Courses
Root Cause, Corrective Action	🔴	🟢	🔴	🔴	1	2	1
Software Development Planning	🔴	🟡	🔴	🟢	1	0	5
Software Documentation	🔴	🟡	🔴	🟢	1	0	4
Software Metrics	🔴	🟡	🔴	🟢	1	3	0
Validation of Sustainability Requirements	🔴	🟡	🔴	🟢	1	2	4
Artificial Intelligence and Machine-Learning Applications	🔴	🔴	🟢	🔴	1	0	9
Augmented and Virtual Reality Applications	🔴	🔴	🟢	🔴	1	1	4
Balancing Quality Attributes	🔴	🟢	🔴	🔴	1	0	3
Detailed Backlog Management	🔴	🟢	🔴	🔴	1	3	2
Development Tempo	🔴	🟢	🔴	🔴	1	0	4
Effort Estimation	🔴	🟢	🔴	🔴	1	1	2
Emerging Technologies	🔴	🟢	🔴	🔴	1	1	2
Product Roadmap and Schedule Management	🔴	🟢	🔴	🔴	1	0	0
Safety Critical Systems	🔴	🟢	🔴	🔴	1	0	5
Software Assurance	🔴	🟢	🔴	🔴	1	3	1
Automated Test and Continuous Integration	🔴	🟡	🔴	🔴	0	2	3
Capabilities Elicitation	🔴	🟡	🔴	🔴	0	6	11
Change Management	🔴	🟡	🔴	🔴	0	3	0
Cost Management	🔴	🟡	🔴	🔴	0	1	2
Embedded Systems	🔴	🟡	🔴	🔴	0	0	7

Table H.1—Continued

Competency	ISA 101	ISA 201	ISA 301	ISA 320	# Green	Other DoD Courses	Civilian Courses
High-Availability Systems	🟠	🟡	🟠	🟠	0	2	1
Software Deployment Patterns	🟠	🟡	🟠	🟠	0	1	3
System Engineering Planning	🟠	🟡	🟠	🟠	0	2	1
High Fidelity System Modeling	🟠	🟠	🟠	🟠	0	2	1
Innovation and Entrepreneurship	🟠	🟠	🟠	🟠	0	0	3
Model-Based Engineering	🟠	🟠	🟠	🟠	0	2	8
Software Ecosystems	🟠	🟠	🟠	🟠	0	7	5
Software Orchestration and Choreography Patterns	🟠	🟠	🟠	🟠	0	1	2
Use/Abuse Case Modeling	🟠	🟠	🟠	🟠	0	1	2
Validation of Performance Efficiency Requirements	🟠	🟠	🟠	🟠	0	1	5

References

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen, and Juhani Warsta, "Agile Software Development Methods: Review and Analysis," VTT Technical Research Centre of Finland, 2002. As of September 17, 2018:
https://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

Agile Alliance, "Scrum of Scrums," webpage, undated. As of January 2019:
https://www.agilealliance.org/glossary/scrum-of-scrums/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'scrum*20of*20scrums))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

———, "Subway Map to Agile Practices," 2018. As of September 17, 2018:
https://www.agilealliance.org/agile101/subway-map-to-agile-practices/

Agile Government Leadership, "Cultural Transformation," *Agile Government Handbook*, 2016. As of September 17, 2018:
https://handbook.agilegovleaders.org/#cultural-transformation

Ambler, Scott, "The Agile System Development Life Cycle (SDLC)," Ambysoft, undated. As of August 27, 2019:
http://www.ambysoft.com/essays/agileLifecycle.html

———, "Examining the Agile Manifesto," Ambysoft, 2014. As of September 17, 2018:
http://www.ambysoft.com/essays/agileManifesto.html

Association of Modern Technologies Professionals, "Software Development Methodologies," 2018. As of September 17, 2018:
http://www.itinfo.am/eng/software-development-methodologies/#chapter14

B., Kerry, "5 Software Development Trends to Watch for in 2018," April 23, 2018. As of November 20, 2018:
https://dzone.com/articles/5-software-development-trends-to-watch-for-in-2018

Balaji, S., and M. Sundararajan Murugaiyan, "Waterfall vs. V-Model vs. Agile: A Comparative Study on SDLC," *International Journal of Information Technology and Business Management*, 2012.

Beck, Kent, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, and Brian Marick, "Manifesto for Agile Software Development," 2001. As of September 17, 2018:
http://agilemanifesto.org/

Belanich, J., F. L. Moses, and P. Lall, *Review and Assessment of Personnel Competencies and Job Description Models and Methods*, Alexandria, Va.: Institute for Defense Analyses, 2016. As of April 8, 2019:
https://apps.dtic.mil/docs/citations/AD1021552

"Best Computer Engineering Programs," *U.S. News & World Report*, 2018. As of February 2018:
https://www.usnews.com/best-graduate-schools/top-engineering-schools/computer-engineering-rankings

Bourque, P., and R. E. Fairley, eds., *SWEBOK V3.0: Guide to the Software Engineering Body of Knowledge*, Piscataway, N.J.: IEEE Computer Society, 2014.

Bureau of Labor Statistics, U.S. Department of Labor, "Software Developers," *Occupational Outlook Handbook*, April 13, 2018. As of March 05, 2019:
https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

Campion, Michael A., Alexis A. Fink, Brian J. Ruggeberg, Linda Carr, Geneva M. Phillips, and Ronald B. Odman, "Doing Competencies Well: Best Practices in Competency Modeling," *Personnel Psychology*, Vol. 64, No. 1, 2011, pp. 225–262, doi:10.1111/j.1744-6570.2010.01207.x.

Cao, Lan, Kannan Mohan, Peng Xu, and Balasubramaniam Ramesh, "A Framework for Adapting Agile Development Methodologies," *European Journal of Information Systems*, Vol. 18, No. 4, 2009, pp. 332–343.

Capgemini, "The Changing Dynamics of the Global High Tech Industry," 2011. As of November 20, 2018:
https://www.capgemini.com/wp-content/uploads/2017/07/The_Changing_Dynamics_of_the_Global_High_Tech_Industry_____An_Analysis_of_Key_Segments_and_Trends.pdf

Chang, Su J., Angelo Messina, and Peter Modigliani, "How Agile Development Can Transform Defense IT Acquisition," in P. Ciancarini et al., eds., *Proceedings of 4th International Conference in Software Engineering for Defense Applications*, Switzerland: Springer International Publishing, 2016.

Cockburn, A., and J. Highsmith, "Agile Software Development: The People Factor," *Computer*, Vol. 34, No. 11, November 2001, pp. 131–133. As of September 17, 2018:
https://ieeexplore.ieee.org/document/963450/#full-text-section

Committee on National Security Systems Instruction 4009, *National Information Assurance (IA) Glossary*, April 26, 2010.

Congressional Research Service, "The Department of Defense Acquisition Workforce: Background, Analysis, and Questions for Congress," July 29, 2016.

DAU—*See* Defense Acquisition University.

Defense Acquisition University, "About AWQI," webpage, undated. As of August 27, 2019:
https://www.dau.edu/tools/awqi/p/About-AWQI

———, "Acquisition Workforce Qualification Initiative," undated. As of August 27, 2019:
https://www.dau.edu/tools/awqi/

———, "Career Fields," webpage, undated. As of August 27, 2019:
https://icatalog.dau.edu/onlinecatalog/CareerLvl.aspx

———, *Defense Acquisition Guidebook*, undated. As of August 27, 2019:
https://www.dau.edu/guidebooks/Shared%20Documents%20HTML/Chapter%201%20Program%20Management.aspx

Defense Innovation Board, *Ten Commandments of Software*, Version 0.14, April 15, 2018. As of January 2018:
https://media.defense.gov/2018/Apr/22/2001906836/-1/-1/0/DEFENSEINNOVATIONBOARD_TEN_COMMANDMENTS_OF_SOFTWARE_2018.04.20.PDF

Defense Science Board, *Design and Acquisition of Software for Defense Systems*, February 2018. As of October 22, 2018:
https://apps.dtic.mil/dtic/tr/fulltext/u2/1048883.pdf

Deloitte, *Agile in Government: A Playbook from the Deloitte Center for Government Insights*, 2017. As of September 17, 2018:
https://www2.deloitte.com/content/dam/insights/us/articles/3897_Agile-in-government/DUP_Agile-in-Government-series.pdf

Department of Defense Instruction 1400.25, "DoD Civilian Personnel Management System: Volume 250, Civilian Strategic Human Capital Planning (SHCP)," Washington, D.C.: Under Secretary of Defense for Personnel and Readiness, November 18, 2008. As of April 8, 2019:
https://prhome.defense.gov/Portals/52/Documents/RFM/TFPRQ/docs/1400.25-v250.pdf

——— 5000.66, "Defense Acquisition Workforce Education, Training, Experience, and Career Development Program," Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, July 27, 2017. As of April 8, 2019:
https://www.esd.whs.mil/portals/54/documents/dd/issuances/dodi/500066_dodi_2017.pdf

——— 5000.66, “Operation of the Defense Acquisition, Technology, and Logistics Workforce Education, Training, and Career Development Program,” Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, December 21, 2005. As of April 8, 2019:
https://pdf4pro.com/fullscreen/department-of-defense-instruction-50a603.html

DoD Military Standard 498, *Software Development and Documentation*, December 5, 1994.

DoD Military Standard 882E, *System Safety*, May 11, 2012.

DIB—*See* Defense Innovation Board.

DoD—*See* U.S. Department of Defense.

DoDI—*See* Department of Defense Instruction.

Eckert, Daniel, “Three Big Emerging Technology Themes from CES 2016,” *PWC*, January 13, 2016. As of November 20, 2018:
http://usblogs.pwc.com/emerging-technology/3-big-emerging-technology-themes-from-ces-2016/

Erwin, Sandra, “Pentagon Advisory Panel: DoD Could Take a Page from SpaceX on Software Development,” *Space News*, April 10, 2018. As of November 20, 2018:
https://spacenews.com/pentagon-advisory-panel-dod-could-take-a-page-from-spacex-on-software-development/

Equal Employment Opportunity Commission, Civil Service Commission, Department of Labor, and Department of Justice, *Uniform Guidelines on Employee Selection Procedures*, August 25, 1978.

European Center for Security and Privacy by Design, *Emerging Trends in Software Development & Implications for IT Security: An Explorative Study*, Darmstadt, Ger.: Technical University Darmstadt, June 2014. As of November 20, 2018:
https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/SoftwareDevelopment-Fraunhofer_SIT.pdf

Finley, J. L., “Establishment of Software Acquisition Training and Education Working Group,” memorandum, Washington, D.C.: Under Secretary of Defense for Acquisition, Technology, and Logistics, February 19, 2008.

Francino, Yvette, “Is the Agile Manifesto Dead? Not by a Longshot,” *Tech Beacon*, undated. As of August 27, 2019:
https://techbeacon.com/agile-manifesto-dead-not-long-shot

GAO—*See* U.S. Government Accountability Office.

Gates, Susan M., Brian Phillips, Michael H. Powell, Elizabeth Roth, and Joyce S. Marks, *Analyses of the Department of Defense Acquisition Workforce: Update to Methods and Results Through FY 2017*, Santa Monica, Calif.: RAND Corporation, RR-2492-OSD, 2018. As of March 18, 2019:
https://www.rand.org/pubs/research_reports/RR2492.html

Gifographics Creative Team, *6 Emerging Software Testing Trends That Will Rule 2018*, infographic, June 8, 2018. As of November 20, 2018: https://gifographics.co/emerging-software-testing-trends-2018-infographic/

Hanssen, G. K., B. Haugset, T. Stålhane, T. Myklebust, and I. Kulbrandstad, "Quality Assurance in Scrum Applied to Safety Critical Software," in H. Sharp and T. Hall, eds., *Agile Processes, in Software Engineering, and Extreme Programming. XP 2016.* Lecture Notes in Business Information Processing, Vol. 251, Cham.: Springer, 2016.

Headquarters Air Force Personnel Center, *Air Force Enlisted Classification Directory (AFECD),* Randolph Air Force Base, Tex.: HQ AFPC/DPSIDC, October 31, 2018. As of October 22, 2019: https://www.afpc.af.mil/Portals/70/documents/07_CLASSIFICATION/20191031%20AFECD.pdf?ver=2019-10-02-093958-540

Huo, Ming, J. Verner, Liming Zhu, and M. A. Baber, "Software Quality and Agile Methods," *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004. As of September 17, 2018: https://ieeexplore.ieee.org/document/1342889/

Institute of Electrical and Electronics Engineers (IEEE) Computer Society, *Software Engineering Body of Knowledge (SWEBOK)*, undated. As of March 21, 2019: https://www.computer.org/education/bodies-of-knowledge/software-engineering

International Standard ISO: ISO/IEC 25010–2011, *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, International Organization for Standardization ISO, 2011.

Joint Task Force on Computing Curricula, *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, New York: IEEE Computer Society and Association for Computing Machinery, 2015.

Kendall, Frank, Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, "Better Buying Power 2.0: Continuing the Pursuit for Greater Efficiency and Productivity in Defense Spending," memorandum, Washington, D.C.: Department of Defense, November 13, 2012. As of April 8, 2019: https://www.acq.osd.mil/fo/docs/USD(ATL)%20Signed%20Memo%20to%20Workforce%20BBP%202%200%20(13%20Nov%2012)%20with%20attachments.pdf

Kenwood, Carolyn A., "A Business Case Study of Open Source Software," MITRE, July 2001. As of November 20, 2018: https://www.mitre.org/sites/default/files/pdf/kenwood_software.pdf

Lanka, Divya, Ch Lakshmi, and D. Suryanarayana, "Application of Fog Computing in Military Operations," *International Journal of Computer Applications*, Vol. 164, 2017, pp. 10–15, 10.5120/ijca2017913653.

Lapham, Mary Ann, Ray Williams, Charles (Bud) Hammons, Daniel Burton, and Alfred Schenker, "Considerations for Using Agile in DoD Acquisitions," Pittsburgh, Penn.: Software Engineering Institute, Carnegie Mellon University, 2010.

Levinson, Harry, "Helping Large Government Programs Adopt and Adapt to Agile Methods," Pittsburgh, Penn.: Software Engineering Institute, Carnegie Mellon University, 2016.

Lockheed Martin, "F-35 Lightning II: A Digital Jet for the Modern Battlespace," webpage, undated. As of April 9, 2019:
https://www.f35.com/about/life-cycle/software

Lotz, Mark, "Waterfall vs. Agile: Which Is the Right Development Methodology for Your Project?," *Segue Technologies*, July 5, 2013. As of September 17, 2018:
https://www.seguetech.com/waterfall-vs-agile-methodology/

Lucero, D. S., "Influencing Software Competencies Across the DoD Acquisition Workforce," *Journal of Defense Software Engineering*, 2010, pp. 4–7.

Manning, Berton, "Software Development: Software Management Plan," *AcqNotes*, June 15, 2018. As of August 27, 2019:
http://acqnotes.com/acqnote/careerfields/software-development-plan

Mansfield, Richard S., "Building Competency Models: Approaches for HR Professionals," *Human Resource Management*, Vol. 35, 1996, pp. 7–18.

Mims, Christopher, "Forget 'the Cloud'; 'The Fog' Is Tech's Future," *Wall Street Journal*, May 18, 2014. As of January 2019:
https://www.wsj.com/articles/SB10001424052702304908304579566662320279406

Mueller, Troy, David Harvey, Awais Sheikh, and Scott Johnson, "Making Agile Work in Government," MITRE , May 2015.

Munassar, Nabil Mohammed Ali, and A. Govardhan, "A Comparison Between Five Models of Software Engineering," *International Journal of Computer Science Issues*, Vol. 7, No. 5, September 2010, pp. 94–101.

Naval Postgraduate School, Computer Science—Curriculum 368 (Resident), Curriculum 376 (Distance Learning), Academic Catalog, 2019. As of April 10, 2019:
https://nps.smartcatalogiq.com/en/Current/Academic-Catalog/Graduate
-School-of-Operational-and-Information-Sciences-GSOIS/Department-of
-Computer-Sciences/Computer-Science-Curriculum-368-Resident-Curriculum
-376-Distance-Learning

Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj, "Challenges of Migrating to Agile Methodologies," *Communications of the ACM*, Vol. 48, No. 5, May 2005, pp. 73–78.

Office of the Secretary of Defense, "Agile and Earned Value Management: A Program Manager's Desk Guide," April 16, 2018. As of January 2019:
https://www.acq.osd.mil/evm/assets/docs/PARCA_Agile_and_EVM_PM_Desk_Guide.pdf

O*Net Online, webpage, undated. As of August 27, 2019:
https://www.onetonline.org

OPM—*See* U.S. Office of Personnel Management.

Public Law 115-91, National Defense Authorization Act for Fiscal Year 2018, December 12, 2017. As of May 21, 2018:
https://www.congress.gov/bill/115th-congress/house-bill/2810/text?q=%7B%22search%22:%5B%22congressId:115+AND+billStatus:/%22Introduced/%22%22%5D%7D&r=126

Putano, Ben, "6 Software Development Trends for 2018: Developers Needed," Stackify, November 24, 2017. As of November 20, 2018:
https://stackify.com/software-development-trends-2018/

Pyster, A., ed., *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering*, Integrated Software & Systems Engineering Curriculum Project, Hoboken, N.J.: Stevens Institute of Technology, September 30, 2009

Radio Technical Committee for Aeronautics Document 178C, *Software Considerations in Airborne Systems and Equipment Certification*, December 2011.

Ramesh, Balasubramaniam, Lan Cao, Kannan Mohan, and Peng Xu, "Can Distributed Software Development be Agile?" *Communications of the ACM*, Vol. 49, No. 10, October 2006.

Ruparelia, Nayan B., "Software Development Lifecycle Models," *ACM SIGSOFT Software Engineering Notes*, Vol. 35, No. 3, May 2010, pp. 8–13.

Schuh, Peter, *Integrating Agile Development in the Real World*, Rockland, Mass.: Charles River Media, Inc., 2004.

Segue Technologies, "What Is Agile Software Development?" August 24, 2015. As of September 17, 2018:
https://www.seguetech.com/what-is-agile-software-development/

Shippmann, J. S., R. A. Ash, M. Battista, L. Carr, L. D. Eyde, B. Hesketh, J. Kehoe, K. Pearlman, E. P. Prien, and J. I. Sanchez, "The Practice of Competency Modeling, *Personnel Psychology*, Vol. 53, 2000, pp. 703–740.

SHRM—*See* Society for Human Resource Management.

Society for Human Resource Management, *Content Validation Study of the SHRM Competency Model*, undated. As of April 8, 2019:
https://www.shrm.org/LearningAndCareer/competency-model/Documents/14-0705%20Content%20Validation%20Study%203.pdf

Society for Industrial and Organizational Psychology, "Principles for the Validation and Use of Personnel Selection Procedures," *Industrial and Organizational Psychology*, Vol. 11, No. S1, 2018, pp. 1–97, doi:10.1017/iop.2018.195.

Smartsheet, “What’s the Difference? Agile vs Scrum vs Waterfall vs Kanban,” 2018. As of September 17, 2018:
https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

Svensson, Harald, and Martin Höst, “Views from an Organization on How Agile Development Affects Its Collaboration with a Software Development Team,” in F. Bomarius and S. Komi-Sirviö, eds., *Product Focused Software Process Improvement*, PROFES 2005, Lecture Notes in Computer Science, Vol. 3547, pp. 487–501, Berlin: Springer, 2005.

Testing Whiz, “8 Software Testing Trends Every Tester Should Follow in 2018,” January 10, 2018. As of November 20, 2018:
https://www.testing-whiz.com/blog/8-software-testing-trends-every-tester-should-follow-in-2018

Tore, Dybå, and Torgeir Dingsøyr, “Empirical Studies of Agile Software Development: A Systematic Review,” *Information and Software Technology*, Vol. 50, Iss. 9–10, 2008, pp. 833–859, 10.1016/j.infsof.2008.01.006.

U.S. Code, Title 5, Section 5105, Standards for Classification of Positions.

U.S. Code, Title 10, Armed Forces.

U.S. Code, Title 41, Chapter 83, Buy American.

U.S. Department of Defense, *Defense Acquisition Workforce Program Desk Guide*, Washington, D.C., July 20, 2017a. As of March 14, 2018:
http://www.hci.mil/docs/Policy/Guidance%20Memoranda/DoDI_5000_66_Desk_Guide_Signed_20_July_2017.pdf

———, *Report to Congress: Restructuring the Department of Defense Acquisition, Technology and Logistics Organization and Chief Management Officer Organization*, August 1, 2017b. As of May 21, 2018:
https://dod.defense.gov/Portals/1/Documents/pubs/Section-901-FY-2017-NDAA-Report.pdf

———, Civilian Personnel Advisory Services, *Strategic Workforce Planning Guide*, November 23, 2016. As of March 14, 2019:
https://www.dcpas.osd.mil/Content/documents/OD/2_A_Strategic_Workforce_Planning_Guide.pdf

U.S. Department of Defense Instruction 1312.01, *Department of Defense Occupational Information Collection and Reporting*, January 28, 2013.

U.S. Government Accountability Office, “DOD Space Acquisitions: Including Users Early and Often in Software Development Could Benefit Programs,” GAO-19-136, March 18, 2019.

———, *High-Risk Series: Progress on Many High-Risk Areas, While Substantial Efforts Needed on Others*, Washington, D.C.: GAO-17-317, February 2017.

———, *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*, Washington, D.C.: GAO-12-681, July 2012.

U.S. Office of Personnel Management, *Competency Model for Cybersecurity*, February 16, 2011. As of June 25, 2019:
https://www.chcoc.gov/content/competency-model-cybersecurity

———, *Interpretative Guidance for Cybersecurity Positions: Attracting, Hiring and Retaining a Federal Cybersecurity Workforce*, October 11, 2018.

———, "Introduction to the Position Classification Standards," 2009. As of June 10, 2019:
https://www.opm.gov/policy-data-oversight/classification-qualifications/classifying-general-schedule-positions/positionclassificationintro.pdf

———, "Our Mission, Role & History," 2019. As of April 10, 2019:
https://www.opm.gov/about-us/our-mission-role-history/what-we-do/

Wijeratne, David, Gagan Oberoi, and Shashank Tripathi, "The New Ways to Win in Emerging Markets," strategy+business, April 24, 2017. As of November 20, 2018:
https://www.strategy-business.com/article/The-New-Ways-to-Win-in-Emerging-Markets?gko=7f566

Wolf, Michael, and Dalton Terrell, "The High-Tech Industry, What Is It and Why It Matters to Our Economic Future," in U.S. Bureau of Labor and Statistics, *Beyond the Numbers*, Vol. 5, No. 8, May 2016. As of November 20, 2018:
https://www.bls.gov/opub/btn/volume-5/pdf/the-high-tech-industry-what-is-it-and-why-it-matters-to-our-economic-future.pdf

RAND NATIONAL DEFENSE RESEARCH INSTITUTE

The U.S. Department of Defense (DoD) seeks to advance the ability of its software acquisition workforce to rapidly and reliably deliver complex software-dependent capabilities through an enhanced understanding of technical competencies, improvements to education and training, and guidance for workforce management and assessment. Focusing on three primary acquisition career fields—information technology, engineering, and program management—the authors review existing competency models used by DoD and commercial industry, along with industry trends and modern software practices, and gather feedback from stakeholders and subject-matter experts to develop a model consisting of 48 competencies organized by topic: problem identification, solution identification, development planning, transition and sustainment planning, system architecture design, software construction management, software program management, mission assurance, and professional competencies. They also review existing courses offered by the Defense Acquisition University, other DoD institutions, and private and public universities to determine whether and to what extent the courses offer software training and education that corresponds with these competencies, and to identify ways to address potential gaps. Although there is no currently accepted government job title or occupational series for software professionals, and although the competency model thus affords limited utility for assessing current workforce capability, the authors present options for tracking and managing the software acquisition workforce, as well as further steps toward validating the competency model.

$35.00

ISBN-10 1-9774-0384-0
ISBN-13 978-1-9774-0384-1

53500

9 781977 403841

www.rand.org

RR-3145-OSD