IDA

INSTITUTE FOR DEFENSE ANALYSES

WHEN IS IT FEASIBLE (OR DESIRABLE) TO USE THE SOFTWARE ACQUISITION PATHWAY?
(Conference Paper)

David M. Tate, Project Leader
John W. Bailey

April 2022

IDA Document NS D-33047

Log: H 22-000141

The Institute for Defense Analyses is a nonprofit corporation that operates three Federally Funded Research and Development Centers. Its mission is to answer the most challenging U.S. security and science policy questions with objective analysis, leveraging extraordinary scientific, technical, and analytic expertise.

Rigorous Analysis | Trusted Expertise | Service to the Nation

WHEN IS IT FEASIBLE (OR DESIRABLE) TO USE THE SOFTWARE ACQUISITION PATHWAY?

David M. Tate
Cost Analysis and Research Division
Institute for Defense Analyses
730 E. Glebe Rd., Alexandria, VA 22305
703-575-1409 (phone)/703-845-6539 (fax)
Email: dtate@ida.org

John W. Bailey
Cost Analysis and Research Division
Institute for Defense Analyses
730 E. Glebe Rd., Alexandria, VA 22305
703-608-9151 (phone)/703-845-2569 (fax)
Email: jbailey@ida.org

Abstract

DoD Instruction 5000.87 establishes a Software Acquisition Pathway (SWP) "for the efficient and effective acquisition, development, integration, and timely delivery of secure software."[1] Under SWP, programs are required to deliver a Minimum Viable Capability Release (MVCR) deployed to an operational environment within 1 year of initial funding. This MVCR must be secure and suitable for operational deployment and must enhance warfighting capability. This paper discusses the challenge of determining for a software development effort whether the minimum capabilities that meet these criteria and enable ongoing agile development can plausibly be developed, tested, and operationally deployed in less than a year. We use a standard software cost and schedule model to derive bounds on the size of software that can be developed and ready to field in 12 months.

The study concludes that many DoD software acquisitions will require too much development effort for the MVCR to comply with the SWP deadline if SWP is used from program initiation. We propose some criteria the Under Secretary of Defense for Acquisition and Sustainment might use to determine whether the SWP is appropriate for a particular new or existing program or software development project. We also consider development strategies that might improve the chances of success in using the SWP, including how non-SWP programs and projects should be architected if the intent is to later transition to SWP.

A. Executive Summary

Department of Defense Instruction (DoDI) 5000.87 establishes a new Software Acquisition Pathway (SWP) to facilitate streamlined acquisition of software-centric applications. The instruction specifies that programs using the SWP must "demonstrate the viability and effectiveness of capabilities for operational use not later than 1 year after the date on which funds are first obligated to develop the new software capability."[2] The thesis of this brief study is that this requirement may be significantly more restrictive than the drafters intended, due to basic constraints on how much operationally adequate software can be designed, developed, tested, and fielded in a single year. In particular, the need for fielded software to be safe, secure, and easily upgraded imposes significant up-front requirements on the architecture, design, verification, and validation of a minimum viable capability release (MVCR) for the system.

Using calibrated models of past defense software development efforts, we estimate that a military software project MVCR is unlikely to exceed ~28,000 equivalent source lines of code (ESLOC) if accomplished in less than 1 year, producing at most ~250,000 physical source lines of code[3] (SLOC). For programs using the SWP, this first-year

---

[1] DOD Instruction (DoDI) 5000.87, "Operation of the Software Acquisition Pathway," October 2, 2020, https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF, accessed March 19, 2022.

[2] DoDI 5000.87, 1.2(e).

[3] Equivalent SLOC are defined in terms of how many new lines of code could be produced with the same effort. Code reuse, adaptation, and auto-generation all increase the ratio of SLOC to ESLOC in a development effort.

product would need to include implementation of key non-mission software features, such as communications architectures and modular design. It would also need to accomplish verification of the effectiveness and suitability of the MVCR, such as cybersecurity, system safety, and interoperability. These are best-case estimates, using optimistic assumptions regarding developer capabilities, application complexity, off-the-shelf tools, code reuse, automated code generation, and the deployment environment. The achievable capability may be significantly less for more complex applications, such as embedded software, software with extensive interoperability requirements, very high required reliability, software incorporating machine learning, or software for use in extreme safety environments (e.g., space or undersea). Embedded software is particularly noteworthy on this list, in that 5000.87 provides an explicit separate path for embedded software that acknowledges the need to coordinate development of the software and its hosting platform.

Conversely, existing software programs that have already passed their MVCR and are now implementing the iterative phase of an agile or DevSecOps process have a much better chance of meeting the strictures of 5000.87 after transitioning to SWP, both in terms of initial delivery and ongoing capability drops. Post-MVCR transition might also help programs resist pressure to take short cuts with regard to architecture, modularity, and other non-functional features important to future agility.

B.   Background

As part of the new Adaptable Acquisition Framework (AAF), DoDI 5000.87 establishes a new acquisition pathway for software with the explicit intent of decreasing development lead times and increasing upgrade frequencies for software-intensive defense systems. The SWP defines two distinct paths: one for applications programs operating on commercial hardware, and a second for development of software embedded in defense systems employing military-unique hardware. The instruction also provides for existing acquisition programs to transition all or part of their acquisition strategy to the SWP. In either case, one criterion for entry into the SWP is that the program must demonstrate viability and effectiveness for operational use within 1 year of the program's first software expenditures. This time constraint clearly places limits on the capabilities that can be implemented using this pathway.

To bound the upper limit of the amount of software that can be developed and delivered in 1 year, we define two segments of software development that could be executed in parallel, and we describe the required attributes and qualities that the delivered software must have, regardless of mission domain. The first segment is the mission-specific functions that constitute the outward behavior and capabilities of the application, and the second segment is the core infrastructure software that enables the mission features to execute on the host computing platform. We refer to these segments as the mission software and the infrastructure software, and assume that the interface between these segments can be well-defined so that their developments can be concurrent. Both of these segments must also satisfy what are often called "non-functional" requirements that the software (and its host system) have certain attributes, such as safety, security, reliability, ethics, maintainability, and so forth. These non-functional requirements are not satisfied by a specific body of code but rather must be achieved and supported by all mission and infrastructure software. Although these

requirements add to the size of the software being developed (and the effort to develop it), they cannot be satisfied by adding to or modifying that code after it is written. These are attributes that the developing code must manifest from its inception.

Under DoDI 5000.87, that first operationally effective and viable increment of capability is referred to as the "minimum viable capability release," or MVCR. The instruction defines this as "the initial set of features suitable to be fielded to an operational environment that provides value to the warfighter or end user in a rapid timeline." Any program whose initial feature set of mission software and all necessary infrastructure software cannot plausibly be completed within a year while achieving the mandatory non-functional requirements should not attempt to use the SWP. Furthermore, any program that would have to trade away future agility in order to meet the 1-year deadline[4] should not attempt to use the SWP, since the lack of future agility would defeat the purpose of the pathway.

The Under Secretary of Defense for Acquisition and Sustainment (USD(A&S)) has authority to direct acquisition programs to use other acquisition pathways if the SWP is not appropriate. This research investigates some criteria USD(A&S) might use to determine whether the SWP is appropriate for a particular new or existing program or software development project. It also considers acquisition strategies that might improve the chances of success in using the SWP, including how non-SWP programs and projects should be architected if the intent is to later transition to SWP after an MVCR has been fielded, or at the earliest, during the year prior to fielding an MVCR.

1. What Constitutes a Minimum Viable Capability Release?

Programs executing the SWP are explicitly not subject to the reporting and review requirements of major defense acquisition programs (MDAPs), regardless of expected life-cycle cost. This has the potential to enable SWP programs to begin significantly sooner than if those programs were required to execute the full sequence of major capability acquisition governance processes, from Mission Needs Statement through Analysis of Alternatives to Milestone A/B authority. If the goal is to field new capabilities as quickly as possible, spending less time getting underway is clearly desirable.

While DoDI 5000.87 requires that the MVCR be "viable," it does not specify where authority lies to determine the minimum acceptable infrastructure and operational capabilities (and non-functional requirements) that define the MVCR. The instruction says, "The PM and the sponsor will… define a minimum viable capability release"[5] without constraining who may or should participate in that MVCR definition process. According to agile precepts, the specification of the minimum viable capability should be determined collaboratively by the user community, senior acquisition executives, the commands that will employ the system, the program manager, and importantly, the developers. In practice, this level of collaboration may be hard to achieve.

---

[4] An example of this kind of trade would be to opt for a proprietary, monolithic system architecture rather than an open, modular architecture. Relaxing the requirement for openness and modularity may permit faster initial release, but would also make subsequent upgrades more difficult and more expensive.

[5] DoDI 5000.87, 3.3(b)(5).

The instruction further defines the "sponsor" to be "the individual that holds the authority and advocates for needed end user capabilities and associated resource commitments."[6] The sponsor is also the individual who approves the Capability Needs Statement (CNS) developed by the operational community. The DoD Components are directed to create streamlined requirements processes to develop, coordinate, and approve CNSs, including an expedited joint validation process if the Joint Staff deems it necessary to protect joint equities.

Instruction 5000.87 explicitly states that assurance of system safety, security, effectiveness, and suitability are still very much required and should be integrated and automated to the maximum extent possible. This provides further evidence that the MVCR will generally have mandatory attributes that cannot be deferred or waived; it must be safe, secure, suitable, and effective in accordance with the mission capability priorities established in the CNS. Those priorities might include both operational features like interoperability as well as structural features like modularity or conformance to an architectural standard.

It is worth noting that, while DevSecOps and agile development offer many benefits to the overall efficiency and productivity of development projects, some of those acceleration benefits apply only to the portion of the project that implements non-mandatory features. In particular, a major efficiency benefit of the agile philosophy is to have the freedom to defer or eliminate the development of features that turn out to be low priority. By definition, the MVCR does not contain any low-priority or "optional" features—if it did, it would not be the *minimum* viable capability release. On the other hand, it might include some features that are important in the long run but not yet useful at the time of initial deployment. Implementation of a modular open system architecture (MOSA), for example, has no immediate benefits for MVCR operations. The benefits of MOSA come later, making it easier and faster to add and upgrade capabilities during the subsequent agile phase of the system life cycle. As a result, the effort to develop the MVCR may include necessary work that does not correspond to any explicit functions in MVCR operations. Similarly, the cybersecurity architecture for the full system may be more complex than required for just MVCR operations, but must still be engineered to support the eventual full range of system operations. For these reasons, the infrastructure software for the MVCR including all its non-functional requirements may constitute a disproportionate fraction, even the majority, of the total code effort.

DoDI 5000.87 describes a workflow in which government developmental and operational testing are integrated from inception and throughout the life cycle to support software assurance, cybersecurity, and mission capability.[7] This reinforces the point that the system architecture and design implementing the MVCR must not only be sufficient in terms of infrastructure and mission capabilities to support the MVCR; they must also implement full safety and security requirements to enable it to be fielded in the operational environment, and must be compatible with agile development thereafter.

---

[6] DoDI 5000.87, G.2.

[7] DoDI 5000.87, 3.2.f(2).

2. MVCR Is Not a Prototype

As noted above, the MVCR must implement all the necessary security features, user permissions, encryption, firewalls, etc. to operate safely in the network so as not to introduce flaws or security back doors into the operational environment. This differentiates the MVCR from a *prototype* application. Important non-functional system attributes are seldom implemented in prototypes. The principal exception is in cases where the application technologies are immature and the purpose of the prototype is to verify that a candidate design approach can meet requirements. Any program that is still verifying the feasibility of technical approaches is a poor candidate to be ready to field in less than a year.

It is tempting to say that the MVCR could be implemented as a prototype of the eventual full-up system in order to save time and effort in getting the first release to the users, but this could be dangerous. It is simply not feasible to change the architecture of an application that implements only a prototype solution. Also, applications that do not implement mission-capable cybersecurity would not be granted authority to operate (ATO). The authors have first-hand experience with the acquisition of a very large defense application that had to be rewritten from scratch late in the development process because the program opted to try to save time by enhancing a prototype that had not been architected to provide cybersecurity. The developers found it impossible to achieve security assurance after the fact. The necessary design, architecture, and development processes to ensure the presence of the fundamental qualities must be incorporated from the beginning.

It would be theoretically possible to develop a disposable MVCR just to meet the 1-year deadline and then replace it with the real code at a later date. However, this seems inconsistent with the objectives of the pathway. Any disposable MVCR would still need full cybersecurity and a complete operational test and evaluation, and fielding it could introduce version control and interoperability issues between the MVCR and the eventual fully compliant application that would replace it. Further, even if a disposable MVCR was thought to be cost-effective in order to maintain conformance to the SWP, it would be antithetical to the DevSecOps philosophy of early testing and integration of the actual application as it will be delivered.

3. MVCR Is Not MVP

DoDI 5000.87 also defines a minimum viable product (MVP) in the context of SWP acquisition. The MVP is defined as an early version of the software that allows users to evaluate and provide feedback on basic capabilities and design features, helping to shape scope, requirements, and design. In practice, an MVP could be a mock-up or storyboard that enables the developers and users, as well as other stakeholders, to agree on how the final application should look and behave. Agile developments use such prototypes to make both the application's requirements and the proposed solutions visible and understandable to all parties.

Language in 5000.87 suggests that the MVP delivery could not only be a waypoint along the path to producing an MVCR, it might even be identical to the MVCR if it is operationally deployable. This suggestion is also dangerous in many respects. The MVP is meant only to illustrate design and function so that developers can show users different

options and ideas.[8] The goal of getting the MVP in front of users and commanders as quickly as possible actually mitigates *against* trying to make it suitable for eventual operational use. Building it correctly, with all of the mandatory structural attributes and security, would take too long, defeating the purpose of eliciting early feedback. The MVP can (and indeed *should*) be devoid of most internal functions and lack many required features but still be useful for generating important requirements feedback. Requiring the MVP to have a secure and modular architecture would miss the point of the MVP. At the same time, basing the MVCR on an MVP that was neither secure nor modular would be an extremely inefficient, and potentially disastrous, way to code. As valuable as the MVP is, there is little chance (and no need) for that product to factor into the deployable application. It should be considered little more than an interactive requirements demonstrator, not production code. Remember also that the MVP and MVCR necessarily compete for resources during early development. Putting too much effort into the MVP could shortchange the development of the MVCR. The two can (and should) be simultaneously developed, but the MVP should be as simple as possible to achieve its purpose—information collection regarding stakeholder acceptance and priorities.

4.   Operational Testing and ATO

Before any changes or additions are made in the operational environment, software must be granted ATO and must pass rigorous operational test and evaluation (OT&E). Consistent with a DevSecOps development process, the required analysis to achieve ATO can be performed continuously and iteratively during development, an approach referred to as continuous ATO (c-ATO). Successful c-ATO can eliminate the dedicated ATO process that confronts programs that treat ATO as a test to be passed, rather than as a state to be maintained.

OT&E has customarily been treated as an end-of-development hurdle to be overcome by an independent team after developers consider a product ready to field, introducing another sequential activity that must be completed before live deployment can occur. IDA experts in conducting OT have concurred that, like c-ATO, continuous OT&E can be integrated into and performed concurrently with development. Although this integration would not entirely eliminate a final evaluation before deployment, it is intended to make that final check routine and efficient. Our discussions with OT&E experts led us to conclude that less than 2 months of OT&E can be sufficient if continuous verifications have been conducted during development. The ability to run developmental and operational testing concurrently during development greatly reduces the late discovery of defects, incompatibilities, and other surprises.

---

[8]   The definition of MVP used in the commercial world is somewhat different from that in DoDI 5000.87. The commercial usage is more akin to the definition of MVCR in the instruction. This may be a source of confusion regarding the potential for an MVP to also be an MVCR.

C. How Much Capability Can You Deliver in a Year?

1. Some Optimistic Assumptions

To assess how much capability could plausibly be developed in 1 year, we will start by making some optimistic assumptions. In particular, we assume that:

- The requirements (both functional and non-functional) for the MVCR are well-defined and fixed.
- A single version of the software is to be fielded.
- The development process takes advantage of all reasonable measures to allow as much concurrency of effort (i.e., parallel development) as possible.
- The development team employs best-practice agile and DevSecOps methods, including continuous testing and early user feedback.
- The development team uses as much automation as possible.

A recent source on software development with agile teams estimates that these conditions could improve development speed by 15% to 20% relative to traditional development methods.[9]

The MVCR is required only to be mission-capable and to provide operational value to users and stakeholders. As noted above, the most efficient way to achieve this would be to divide the project into two parallel developments that run concurrently but independently: 1) mission software that can deliver a modest but useful subset of capabilities, and 2) core application infrastructure providing all the necessary operating system connections, messaging, user privileges, encryption, external interfaces, and essentially anything that is not strictly mission-specific from the user's viewpoint. We need to make some assumptions regarding the relative effort between implementing core infrastructure versus implementing functional mission capabilities, and the implications of non-functional requirements for both infrastructure and mission development effort.

We assume that such a partition of effort is feasible, so that the time to produce a deployable MVCR is determined by the longer of 1) the time to implement the mandatory infrastructure, and 2) the time to implement the mission capabilities. We also assume that development of the core software can adapt and re-use existing, commercial, and open-source code to a much larger extent than the mission software. Finally, we assume that the MVCR must instantiate the full set of non-functional attributes, such as cybersecurity or modularity. The next section explores the maximum amount of code the MVCR could conceivably deliver in a 1-year development effort.

2. Modeling with COCOMO

To understand how much software could be developed and deployed in 1 year, we turned to the current COCOMO software cost model, now formally named COCOMO-

---

[9] Sarah Elk, Steve Berez, and Darrell K. Rigby, *Doing Agile Right*, Harvard Business Review Press, 2020, 76.

II.[10] This version incorporates updates to the cost-estimating relationships published in the original 1981 text by Barry Boehm.[11]

The COCOMO cost equations accept software size estimates and yield estimates of the required number of staff months of effort required to produce that much software. A second COCOMO equation estimates the schedule over which that amount of effort can be accomplished. Schedule does not vary linearly with size, since a larger team can be applied to a larger project, so COCOMO first yields an estimate of total effort and then uses that to provide an estimate of the schedule. Since our constraint is the schedule, not the effort or cost, we used the model to reverse-engineer the largest code size that could be developed and delivered within 1 year under our optimistic assumptions from the previous section.

COCOMO accepts 5 scaling factors and 17 effort adjustment factors to accommodate the differences among software development projects that were observed to have an effect on software development productivity. All 22 factors have nominal (default) values reflecting typical software projects. Each can be tuned to reflect atypical aspects of a given project or development environment. The COCOMO models have been calibrated against historical outcomes to predict the influence of these factors on project outcomes.[12]

Of the 22 factors that can affect productivity and schedule, we left all but 4 at their default values. The non-default values we applied were as follows:

1. *Required software reliability* was set to "high," one level above nominal, to reflect the demands of operational defense mission software.

2. *Use of software tools* was set to its maximum value, reflecting our optimistic assumption about use of automation by the development team.

3. *Required development schedule* was set to "maximum compression of schedule" to maximize the delivered content within the fixed 1-year period.

4. *Process maturity* was set to its maximum value, CMM Level 5, assuming a highly capable development team and organization.

We did not alter the "volatility of requirements" setting because the default is "no volatility," which was one of our optimistic assumptions.

The effects of these parameter settings vary. High reliability adds 10% to the effort and about 3% to the schedule.[13] Setting the tool use factor to Very High reduces effort by 22% and also reduces schedule by about 8%. Tool use is particularly important for the development of the core software since reuse, code generation, and off-the-shelf software are all likely to be extremely useful for that portion of the development.

Setting the Required Development Schedule driver to Very Low results in a 25% reduction in schedule but a 43% increase in effort, trading a 30% drop in productivity for

---

[10] We will adopt the common practice of simply using COCOMO to refer to the more recent COCOMO-II version of the model, an update based on 20 additional years of software project data.

[11] Barry W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[12] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Ellis Horowitz, Ray Madachy, et al., *Software Cost Estimation with COCOMO-II*, Prentice Hall, August 2000.

[13] In COCOMO, schedule estimation involves spreading the estimated effort over a feasible schedule. Since the relationship between effort and schedule is not linear, it means that the effect on schedule from a given effort adjustment factor varies depending on the size of the project.

faster execution. The COCOMO development research determined that no further schedule compression is possible beyond this since the calibration data did not contain any programs that successfully completed in less than 75% of a nominal schedule.

In addition to setting these four adjustment factors, we eliminated the earliest development phase estimated by COCOMO, called the inception phase.[14] This choice reflects our assumption that the requirements for the MVCR are well-defined and have been finalized before commencement of the SWP. COCOMO also provides a final operational verification effort and schedule, which the model calls the transition phase that occurs after development is complete. For developments that complete in close to 1 year, the transition phase in COCOMO is between 1 and 2 months. This phase begins at IOC and ends with product release, making it analogous to our estimates of ATO and final OT&E.

3. Measuring Size

a. ESLOC

COCOMO uses "equivalent source lines of code" (ESLOC) to measure the size of software development efforts. ESLOC is a derived value that considers how many new lines of code must be written, how many preexisting lines of code are re-used unmodified, how many preexisting lines of code will be modified and adapted for use by the project, and how many lines will be generated by software tools. For a project that requires all new code, ESLOC is the same as the number of SLOC. When code is re-used, adapted, or generated by tools, the ESLOC count will be less than the physical SLOC count. It is a theoretical measure of the number of new lines that could have been written with the same effort as that required to adapt and integrate the re-used or generated code. This allows the cost and schedule modeling to work with a single, normalized measure of program size.

For our estimate of the largest MVCR that could be achieved in 1 year, we assumed that all required mission software would be newly developed. However, we assumed that the infrastructure software to support this new mission software would either be highly re-used and adapted from preexisting software, or would be generated by software tools with much less human effort than required to develop those lines of code from scratch. In other words, the ESLOC measure of the infrastructure software is expected to be much less than its delivered software size.

b. Function Points

Function points (FPs) are an alternative measure of code size based on functionality provided rather than volume of code required. COCOMO allows unadjusted function points (UFPs) to be estimated as a size input to the effort estimation and the corresponding schedule projection. This involves prior conversion of UFPs to SLOC and

---

[14] Through collaboration with Rational, Inc. in 1999, USC parsed the COCOMO-II effort into four phases: Inception, Elaboration, Construction, and Transition. The middle two estimate the main software design and development activity, and the final phase is analogous to the operational deployment, discussed in Part B.

running the model as before. Conversion tables are provided in the COCOMO documentation to determine the SLOC that would be comparable to one UFP in various computer languages. These conversions range from 300 lines of assembly language per function point to as few as 10 or fewer lines per function point for fourth- and fifth-generation languages. Third-generation languages, such as C and Ada (and older languages such as Fortran), range from 70 to 120 source lines per function point. Object-oriented languages like C++ and Ada95 can implement a function point in about 50 lines.

Although it may be possible to use a single language for all of the mission software development, this is unlikely to be true for the system and support software due to the various tools and operating system software that we expect will have to be integrated to complete the implementation. Thus, any conversion to UFP will be crude for the non-mission portion of the MVCR development.

4.   COCOMO Calculations

a. Modeling Results

COCOMO modeling with the settings described above indicates that 28,000 ESLOC could be developed and made ready for deployment in 12 months, including either the COCOMO estimate for transition or our independently derived estimate of between 1 and 2 months for OT&E. This estimate assumes that the mission software and the non-mission infrastructure software can be developed independently and concurrently. Continuous verification of the compatibility between the two parallel tracks would be necessary to prevent any incorrect assumptions about their interfaces from delaying their eventual integration. DevSecOps processes would help to achieve the integration of verification with developmental testing cycles.

We assume that the MVCR mission software will be almost entirely new code, with little reuse from prior applications. Thus, the maximum delivered size of the mission code in the MVCR would remain close to 28,000 SLOC.

For the non-mission infrastructure software, we assume significant opportunities for reuse. The literature suggests various overhead factors to re-use existing software, based on how much study is required to understand the software, whether it has to be modified, and whether it has to be tested and verified or can be assumed to function as specified. For our bounding exercise, we optimistically assume that the infrastructure software is well-understood and well-supported by tools and existing software. Based on discussions with software development experts at our company who are familiar with the development of infrastructure software through the use of tools and reuse, we assume that the composite of all the reuse effort factors for the infrastructure software could be as low as 5% to 15% of the cost of new code development.[15]

Thus, for the infrastructure segment, our modeling estimates that as many as 250,000 physical lines of software could be produced with 28,000 ESLOC of effort. To

---

[15] A 15% cost relative to new code development was also observed by one of the authors in *A Component Factory for Software Source Code Re-engineering*, University of Maryland, 1992. Bailey measured the rate of near-verbatim mission software reuse at the Software Engineering Laboratory at the University of Maryland and NASA Goddard. Our expectation is that the general case of re-using operating system and support software should be even more efficient.

arrive at this total code size, we used optimistic estimates for the efficiency of reuse due to the high tool use adjustment factor and the wide availability of construction tools for this more general-purpose software.[16] If the infrastructure software were unprecedented (e.g., if it had to be hosted on novel hardware), the availability of tools and reusable code would be much more limited. As a result, the same amount of effort would not be able to deliver nearly as much code. A case study we used for other insights into relative sizes and efforts for mission and infrastructure software is described in section 5.

b. COCOMO Modeling Details

According to COCOMO, after applying the effort adjustment factors of high reliability, high tool use, and shortest schedule, and adopting the highest process maturity, 28,000 ESLOC would take 10.8 months to develop. The transition phase (corresponding to OT&E) is estimated to take an additional 1.4 months, which we optimistically reduce to 1.2 months due to continuous verification during development. Since we assume mission software is nearly all new code, this bounds the mission software that can be delivered as part of the MVCR.

Potentially more infrastructure software (in terms of SLOC) can be included in the MVCR, due to available tools, reuse, and adaptation of existing software. If we assume that 28,000 ESLOC of infrastructure software would require only 10% of the resulting SLOC to be newly developed, and the remainder would require 50% of the integration effort of new code, it would be possible to deliver about 150,000 SLOC of software. If the integration effort were only 10% that of new code, COCOMO modeling suggests that over a half million SLOC could be delivered. As a compromise, we estimate that the upper limit for delivered infrastructure software from a highly automated 28,000 ESLOC effort is probably somewhere in the range of 200,000 to 300,000 SLOC under these highly optimistic assumptions. Figure 1 shows the maximum ESLOC as a function of available time.

---

[16] Examples from one of our colleagues include COTS RTOS run-time operating system tools, such as those from VxWorks or QNX, and for appropriate applications, cloud-native functions associated with platform as a service (PaaS) and software as a service (SaaS). Development with open-source software (OSS), adoption of middleware, virtualization, or containers were also offered as common approaches.
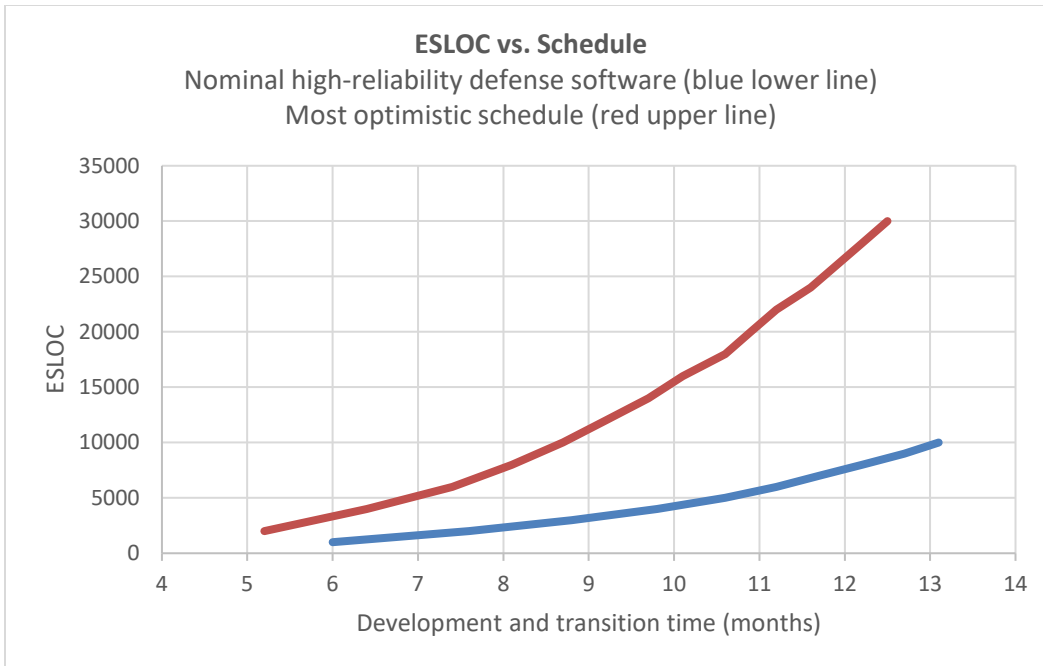
Figure 1. Development schedule vs. software size for typical defense software

5. Case Study Data

Now that we have estimates of the maximum ESLOC that can be delivered and deployed in 1 year, the remaining questions are 1) how much mission function can be delivered with a maximum of 28,000 new lines of software, and 2) would 250,000 lines of system and support software, including the requisite cybersecurity, be enough to host the MVCR mission software in the operational environment?

Capers Jones summarized 265 new and enhancement military software projects prior to 2000.[17] Application sizes were expressed in function points to eliminate the effect of language in the comparison. To compare this to our COCOMO modeling, we converted to ESLOC by assuming implementation in a third-generation language such as Ada, C, or Fortran.

The new projects in Jones average 2,800 function points, implying they probably average between 200,000 and 300,000 SLOC. Further, a graphic in Jones shows some much smaller completed projects, on the order of 100 function points. Although many of these could be enhancement projects, we note that software enhancements can also qualify for SWP. A 100 UFP development using C would amount to about 12,000 source lines. In fact, at 120 lines per UFP, up to a 233 UFP project in C could be developed within the 28,000 ESLOC limit for a 1-year MVCR deployment. This implies that some of the projects in that data set could have been entirely completed in a 1-year development effort, according to our modeling. However, most of the projects clearly would have required several years, even under our optimistic assumptions. Jones does not address the relative effort needed for infrastructure versus minimally viable mission software for those systems; it is probable that some of those larger projects could have

---

[17] Jones, *Software Assessments, Benchmarks, and Best Practices,* Addison-Wesley, 2000.

demonstrated viable and useful operational capability in just 1 year. The authors are unaware of any statistics on the typical size of the MVCR for military applications relative to the mature application after iterative improvements.

As a case study, we examined data on a military software project that involved the development of multiple radio waveform applications that would run on a core of hardware-specific infrastructure software. Each waveform corresponded to a legacy radio family that the new system would be able to communicate with. The infrastructure software would implement signal processing, cryptography, and other common radio functions, while the waveform software would enable the radio to communicate with various legacy radio systems. Although the program office estimated a need for more than 4 million lines of waveform software in total, the individual waveforms were estimated to each need anywhere from 2,500 to 200,000 SLOC. Our modeling confirmed, therefore, that a small but operationally useful number of those waveforms could have been developed within the 1-year MVCR deployment constraint. (This was consistent with the outcome of earlier prototyping efforts in a similar application.) Unfortunately, because of the more limited opportunities for low-cost reuse and available tools than in our optimistic scenario, the infrastructure software was estimated to require more than 1 million ESLOC to deliver 2.1 million SLOC, a net gain from reuse and tool-generated code of only 50% of the cost of new code. This suggests that this program would not have been a viable SWP candidate from its inception. A more realistic plan would have been to implement the infrastructure functions under a different pathway—perhaps Middle Tier Acquisition[18]—and then to transition the balance of the development to the SWP after or within 1 year of the successful completion of the infrastructure portion of the MVCR plus an operationally useful subset of waveforms.

Some projects implementing DevSecOps have reported higher software development productivity than was observed in the projects used to calibrate COCOMO. There is some evidence that agile and DevSecOps approaches can improve even our optimistic MVCR productivity rates through the benefits of early and continuous testing and user feedback. These improvements, if real, would not be captured in COCOMO-II, which attempted to be forward-looking with some of its adjustment factors but was last calibrated in 2001. Similarly, the examples from Jones are also more than 20 years old. Novel tools and code generators to help build operating systems and infrastructure software continue to appear in the marketplace. Although these advances generally show up first in commercial software development, applicable ones eventually appear in defense acquisitions. It is conceivable that our best-case estimates of productivity should be bumped up by an additional 10% to 20% to account for this. However, even with that additional headroom, many defense software projects—and particularly those associated with major capabilities—still appear unlikely to be executable to MVCR in 1 year of development. A significant impediment to the delivery of operationally deployable software is the stringent non-functional requirements associated with deployed operational systems, and the extensive infrastructure effort needed to support operational viability and long-term maintainability of the applications.

---

[18] DoDI 5000.02, section 4.2(b), *Operation of the Adaptive Acquisition Framework*, January 23, 2020.

D. Transitioning into SWP

DoDI 5000.87 provides for programs that are currently being executed under some other acquisition pathway to transition to the SWP when they can plausibly meet the specified timelines. Our analysis provides a template for how program managers could assess whether they are within 1 year of achieving the needed software maturity. First, separate the remaining development into two segments: the architectural and infrastructure requirements and the minimum mission capability requirements. Second, estimate the remaining development time (including OT&E) to complete the MVCR set of requirements for each of the two segments, including verification of all non-functional requirements that are mandatory for actual operations. When the larger of those two estimates is less than 1 year, the program may be ready for transition to SWP.

Since the infrastructure requirements of the application will often require more development effort than the minimal set of mission capabilities, it is important that programs not be tempted to skimp on non-functional attributes such as cybersecurity, reliability, or modular design in an attempt to accelerate delivery of the MVCR. Missing non-functional attributes at MVCR are very unlikely to be satisfied later, short of a complete rewrite of the application. Where the complexity of infrastructure requirements or the non-functional demands of future capability (or both) are high, transitioning to SWP after implementation of the infrastructure functions may be the more effective process in the long run.

Our research suggests that programs that cannot expect to deliver a fieldable capability that provides operational value by the end of the first year of development should be conducted under a different acquisition strategy, such as Middle Tier Acquisition, at least until an MVCR can be completed in 1 additional year. At that time, transitioning to SWP would be feasible as long as continuous ATO and embedded OT&E verification of required non-functional attributes had been practiced, and care had been taken in architecting the solution such that annual upgrades could be delivered after every subsequent year of development. The time prior to transition to SWP could also be used to develop automated test environments to support rapid capability upgrades post-transition.

E. Summary

This exploratory study examines the implications of the DoD policy that acquisition programs using the software acquisition pathway (SWP) must have produced viable and effective code suitable for operational deployment within 1 year of initial funding. We estimate the maximum amount of completed code that could be produced under ideal conditions within that time span, and use those results to bound the feasible attributes of the minimum viable capability release (MVCR). To do this, we distinguish three drivers of MVCR effort: implementation of core infrastructure code that mission capabilities will rely on; implementation of an operationally useful set of mission capabilities; and assuring the mandatory non-functional attributes that the application must possess prior to operational use and maintain throughout its life. We note that the infrastructure code effort typically generates the more binding constraint, especially when assurance of the non-functional requirements is considered.

Using the COCOMO-II software cost estimation model, we estimate that 28,000 equivalent source lines of code (ESLOC) is the most optimistic limit on the size of either

the non-mission infrastructure software or the mission package that could be fielded in 1 year. Comparison against historical DoD software development efforts suggests that many past systems exceed 28,000 ESLOC of mandatory infrastructure software, and would thus not have been good candidates for SWP execution under the new pathway. Even though we find that useful mission software subsets can often be completed in under a year, many DoD software applications are likely too complex to complete and field enough of the required infrastructure software to supply that mission subset with the required core system services, external interfaces, data management, and other non-mission-specific functions while maintaining required levels of safety, information assurance, and other non-functional requirements. This is even more true for applications planning to support future agility by using modular open-systems architecture (MOSA), or for embedded applications on new or existing defense-specific platforms.

Since the SWP also allows projects that are already underway to transition to the SWP development model, that pathway is available to any number of DoD software projects as long as they are also able to adopt an annual, agile release cycle for deployment upgrades. This may be a heavy lift for legacy applications that were not planned from the start to be agile. Programs intending to transition to SWP at some point should therefore devote early attention to architectures and design choices that will allow them to achieve and maintain continuous authority to operate and regression testing of effectiveness and suitability. This attention should also include explicit verification and tracking of non-functional requirements from very early in the program life cycle.

All of the estimates developed in this study were based on excessively optimistic estimates of the effectiveness of the development team, the ease of software reuse, and the benefits of agile and DevSecOps methodologies when implementing a fixed set of requirements. In the authors' opinions, it is more likely that two concurrent segments of 10,000 to 15,000 ESLOC is the effective upper limit on a 1-year development of a nontrivial new-start application to be used in combat or intelligence environments. It is not clear that the drafters of DoDI 5000.87 intended the 1-year restriction to be this binding, but as currently promulgated it would prevent many of the Department's highest-profile software efforts from starting on the SWP.

References

Bailey, John Winston. *A Component Factory for Software Source Code Re-engineering*. Ph.D. dissertation. University of Maryland, 1992.

Boehm, Barry W. *Software Engineering Economics*. Prentice Hall, 1981.

Boehm, Barry W., Abst, Chris, Brown, A. Winsor, Chulani, Sunita, Horowitz, Ellis, Madachy, Ray, et al. *Software Cost Estimation with COCOMO-II*. (2020). Prentice Hall.

DoD Instruction (DoDI) 5000.87. "Operation of the Software Acquisition Pathway." October 2, 2020. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF. Accessed March 19, 2022.

_____. 1.2(e).

_____. 3.3.(b)(5).

_____. G.2.

Elk, Sarah, Berez, Steve, and Rigby, Darrell K. *Doing Agile Right*. Harvard Business Review Press, 2020, 76.

Jones, Capers. (2020). *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Publishing Co., Inc.

Biographical Information

David M. Tate

David Tate joined the research staff of the Institute for Defense Analyses (IDA) Cost Analysis and Research Division in 2000. Prior to that, he was an Assistant Professor of Industrial Engineering at the University of Pittsburgh, and the Senior Operations Research Analyst (Telecom) for Decision-Science Applications, Inc. At IDA, he has worked on a wide variety of resource analysis and quantitative modeling projects related to national security. These include an independent cost estimate of Future Combat Systems development costs, investigation of apparent inequities in Veterans' Disability Benefit adjudications, and modeling and optimization of resource-constrained acquisition portfolios. Dr. Tate holds bachelor's degrees in Philosophy and Mathematical Sciences from the Johns Hopkins University, and M.S. and Ph.D. degrees in Operations Research from Cornell University.

John W. Bailey

John Bailey has worked for both the former Computer Science Division and the Cost Analysis and Research Division of the Institute for Defense Analyses (IDA) since the 1980s. Prior to that, he conducted software programming research for General Electric and co-founded Software Metrics, Inc. While completing his Ph.D. at the University of Maryland in software reuse methodology, he helped Rational establish its Ada development environment, which was later acquired by IBM. Dr. Bailey has overseen numerous cost and schedule analyses for Department of Defense software programs, as well as root cause analyses of cost and schedule growth.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE

4. TITLE AND SUBTITLE

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	c. THIS PAGE		

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER