SYM-AM-17-080

Proceedings
of the
Fourteenth Annual Acquisition Research Symposium

Thursday Sessions
Volume II

Acquisition Research:
Creating Synergy for Informed Change

April 26–27, 2017

Published March 31, 2017

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

Acquisition Research Program
Graduate School of Business & Public Policy
Naval Postgraduate School

Software Productivity Trends and Issues

David M. Tate—joined the research staff of the Institute for Defense Analyses' Cost Analysis and Research Division in 2000. Since then, 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 Johns Hopkins University, and MS and PhD degrees in operations research from Cornell University. [dtate@ida.org]

Abstract

The Department of Defense is experiencing an explosive increase in its demand for software-implemented features in weapon systems. The combination of exponential increases in computing power and similar advances in memory density and speed has made software-mediated implementation of system features increasingly attractive. In the meantime, defense software productivity and industrial base capacity have not been growing as quickly. Do we have an impending bottleneck? If so, what are the management implications?

Malthus on Software

The Scottish cleric and economist Thomas Robert Malthus famously noted that, when there is enough food to go around, population growth is exponential. Since Malthus could not envision any means whereby food production could also grow exponentially, given the constraints of arable land and property ownership, he predicted that the inevitable result would be a population limited by recurring poverty and starvation.

Malthus was wrong about food, at least so far, but could he be right about national security software? Any time you have an exponential growth in demand without a commensurate exponential growth in supply, demand will soon be frustrated. Rapidly growing demand for new software, combined with the need to maintain the code going forward, places considerable stress on the productive capacity of the defense software industrial base. The ability to keep up will depend on just how fast demand is growing, how quickly the Department of Defense (DoD) can grow the industrial base, and how quickly the productivity of individual software developers improves over time. To know whether we should worry, we need to look at each of those factors.

How Fast Is Defense Demand for Software Growing?

It is surprisingly difficult to find historical and current data on the demand for software in defense systems. However, there are some strong indicators available:

- The National Research Council (2010) wrote that "the extent of the DoD code in service has been increasing by more than an order of magnitude every decade, and a similar growth pattern has been exhibited within individual, long-lived military systems." One order of magnitude per decade is approximately 25% annual growth.
- The Aerospace Vehicle Systems Institute (2017) states that source lines of code (SLOC) in aircraft (both military and commercial) has been doubling approximately every four years. This corresponds to an annual growth rate of ~18%.
- The Army (2011) estimated that the volume of code under Army depot maintenance (either post-deployment or post-production support) had

increased from 5 million to 240 million SLOC between 1980 and 2009. This corresponds to ~15% annual growth.

- Dvorak (2009) stated that National Aeronautics and Space Administration unmanned space systems SLOC have also increased by an order of magnitude every 10 years, with manned systems SLOC growing even faster.

Taken together, these suggest an annual growth rate of at least 15% for the amount of software being developed and maintained for defense purposes, with 25% or more annual growth possible. Annual growth of 15–25% means doubling every three to five years, on top of which is the added workload of maintaining the growing base of deployed code.

In order to forecast future demands for new code and software maintenance, we also need to know the current size of the code base and the current annual demand. The most recent demand estimate we were able to find (Chao, 2006) concluded that the 2006 requirement for national security software was about 35 million lines of new code and 25 million lines of maintenance code. We can apply the "20% per year" rule of thumb for maintenance effort to infer a deployed 2006 base of about 125 million lines of code. We will base our analysis on those assumptions: 125 million source lines of code (MSLOC) under maintenance in 2006, 35 MSLOC of new code required in 2006, and annual demand for new code growing at 15% annually from that time forward. For maintenance effort, we assume that annual maintenance effort on the installed base is equivalent to 20% of the development effort of the base, and that half of the maintenance effort results in more new code to be maintained.[1] In addition, some fraction of the installed code base is retired every year. We will assume that 10% of the installed base is retired each year, exactly offsetting the new code generated by maintenance. As we will see, the conclusions of this investigation are not sensitive to the exact parameter values chosen here.

Figure 1 shows the projected growth in annual demand for defense software under these assumptions, separated into new code and maintenance of existing code. Bear in mind that this is a projection of *unconstrained* demand—how much the DoD is expected to want to buy, if it is available at prices comparable to historical prices.

[1] Jones (2013) estimates the maintenance costs of a nominal 1000-function point application at closer to 40% per year over the first five years. Using that estimate would result in a smaller 2006 deployed code base estimate, but much faster growth in that base in subsequent years.

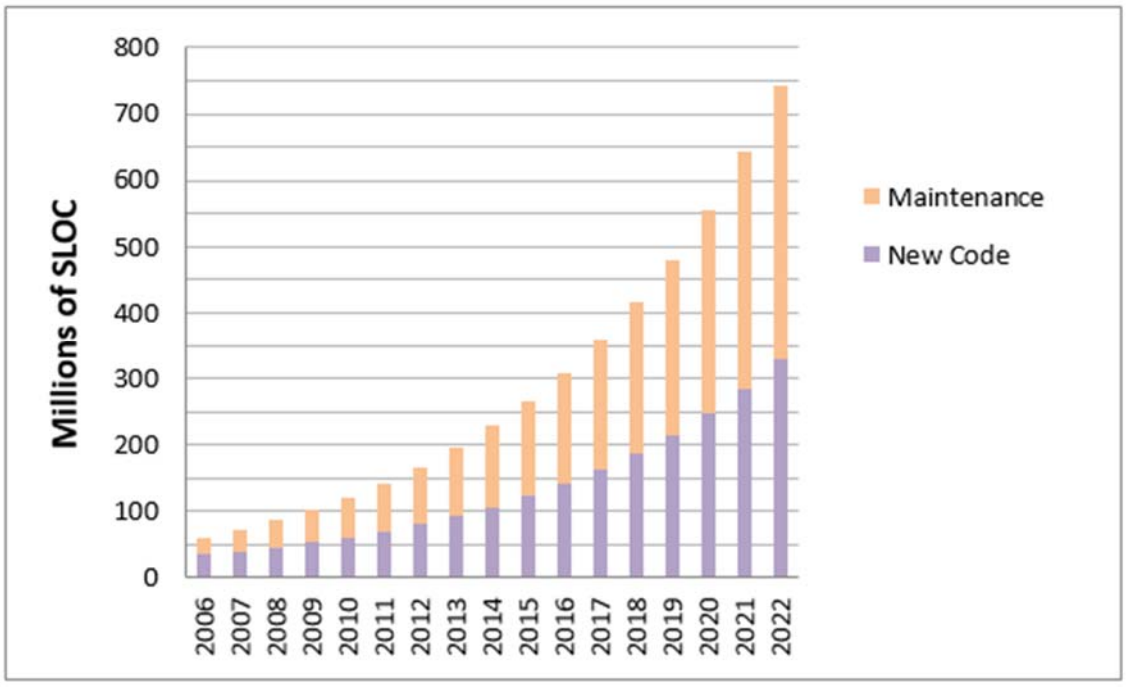

Figure 1. Forecast of DoD Software Demand

It is worth noting that, under these assumptions, the total size of the deployed code base under maintenance is projected to be more than 1 billion SLOC by 2018, and more than 3 billion SLOC by 2025. Figure 1 shows only the new effort each year, not the deployed base.

The Supply of Defense Software

Chao (2006) estimated both the size of the defense software workforce and the productivity of that workforce. The productive capacity of the industrial base is the product of those two factors. We will attempt to estimate each in turn using available data. For purposes of this analysis, we will accept Chao's estimates that there were 68,000 cleared software developers in 2006, capable of producing 75 MSLOC per year. That implies a productivity at that time of roughly 1100 SLOC per developer in 2006, or (equivalently) 900 developers required per MSLOC, as our baseline.

The Size of the Workforce

How quickly might the defense software workforce be growing? The Bureau of Labor Statistics estimates that from 2010 to 2015, total employment of software developers[2] grew almost 30%, or about 5.3% annually. However, they forecast that rate to decline sharply going forward, averaging only about 1.6% per year over the decade of 2014–2024 (BLS, 2017). The defense software industrial base will need to grow more quickly than that to keep pace with established demand growth.

[2] BLS occupation codes 15-1132 (software developers, applications) and 15-1133 (software developers, system software), total employment as of May 2010 and May 2015 (BLS, 2017).

Any scarcity of cleared software talent should translate into rising salaries and benefits for workers with those skills, providing incentive for more and more workers to enter the industry. In a free and liquid market, we would expect this to happen fairly quickly. Unfortunately, some aspects of this particular market might be problematic. The first is the requirement that workers be U.S. citizens with security clearances. This not only dramatically restricts the pool of potential entrants, but it also creates a licensure bottleneck for individuals seeking to join the labor force. There is currently a backlog of half a million unfinished security background checks, and the time required to get through the process is increasing steadily. In addition, over the past few years the total size of the cleared workforce has contracted by ~25% in reaction to high-profile spills of classified information. Defense software employers are also facing tough competition from the private sector, which is experiencing an explosion of demand for software to power the expanding role of the Internet in daily life. Of course, other industries can supplement U.S. graduates with offshore or immigrant labor—a solution unavailable to the defense sector under current regulations.

Another barrier to market corrections is that the most urgent scarcities seem to be at the high end of the experience scale. Chao (2006) found that (at least in 2006) there was no general shortage of programmers, but there was already a significant shortage (with corresponding salary premium) of relatively senior software project managers, architects, and developers. At the tip of the pyramid, they cited a cadre of 500–600 "elite" individuals who play a disproportionate role in project success. If rising compensation for senior talent begins to cause an increased growth rate in software degrees, we will not see that begin to alleviate the crunch in senior talent and elite individuals for at least another 10 years.

Finally, it is not clear that the DoD *wants* the market to correct itself through increases in compensation. Contractor labor rates are closely monitored by the DoD, and the government pushes back when they rise too quickly. Senior software talent in the general economy can be as highly compensated as senior management executives. Arrington (2010a) reported that "[a Google employee] was recently offered a counter offer he couldn't refuse (except he did). He was offered a 15% raise on his $150,000 mid level developer salary, quadruple the stock benefits and … wait for it … a $500,000 cash bonus to stay for a year. He took the Facebook offer anyway." (Note that $150,000 for a mid-level developer is already well above industry norms.) Arrington (2010b) also reported that Google had paid a top software engineer $3.5 million to turn down an offer from Facebook. Allowable defense contractor labor costs are capped; companies choosing to pay salaries over those caps must take the difference out of profit. This provides a strong disincentive to paying market rates for top talent within the defense world.

On the supply side, what does the educational pipeline for software look like? The number of bachelor's degrees conferred each year in computer and information sciences has shown a striking cyclical pattern over the past four decades (see Figure 2). The general trend has been a baseline increase of ~1000 degrees per year, with superimposed boom and bust cycles. We are currently on the upswing of a boom cycle, with more than 60,000 degrees conferred per year.

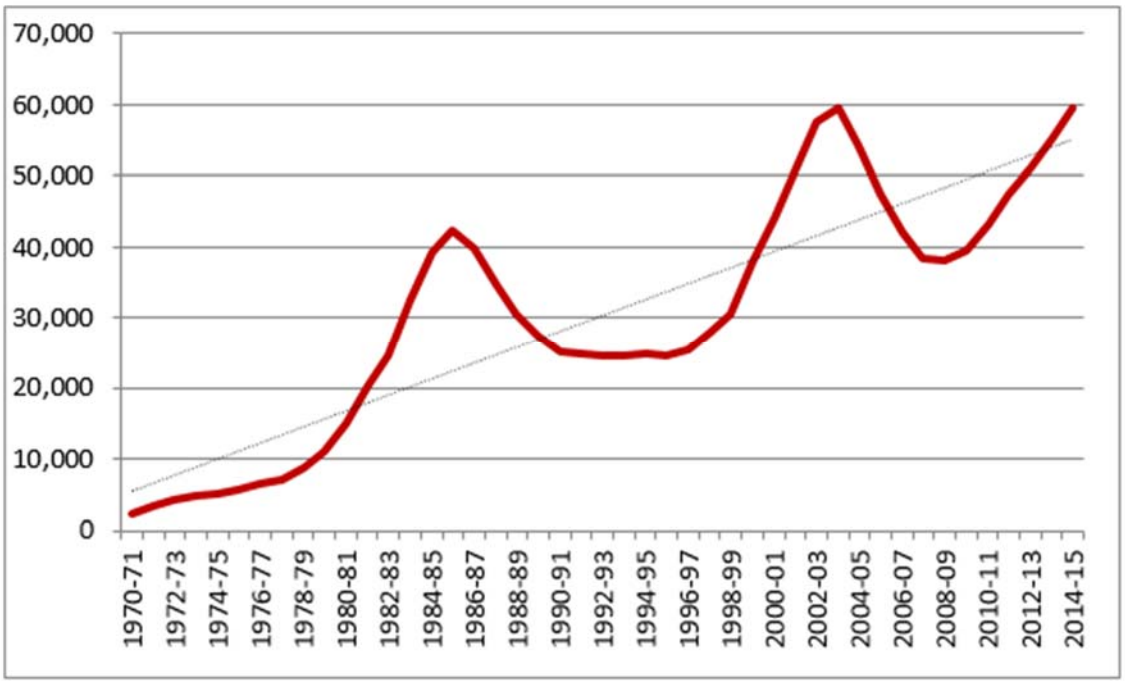

Figure 2. Annual Computer Science and Information Sciences Bachelor's Degrees Conferred

In addition to this pool of potential defense software developers, the educational pipeline for software developers also includes nontraditional educational options. More than 16,000 students graduated from “coding boot camp” programs in 2015, and that number has been growing rapidly over the few years that such programs have existed (Lauerman, 2015).

This suggests that there are as many as 80,000 potential developers graduating per year. In 2006, the cleared workforce made up 7% of the overall software workforce. Again being optimistic, if 10% of new graduates (college and boot camp combined) end up in the cleared software workforce, that would currently be about 8,000 per year, which could grow to 10,000 per year in a couple of years. This corresponds to between 5% and 10% annual growth. For purposes of our baseline analysis, we will assume annual workforce growth of 5%, comparable to recent growth in software developers and well above the forecast national average for the software industry.

As noted above, in 2006, there were roughly 68,000 cleared software developers in the defense industrial base. If we assume 5% annual growth in the national security software developer workforce starting in 2006, that would translate to about 120,000 people today, reaching 150,000 by 2023. Figure 3 shows this projected growth over time.

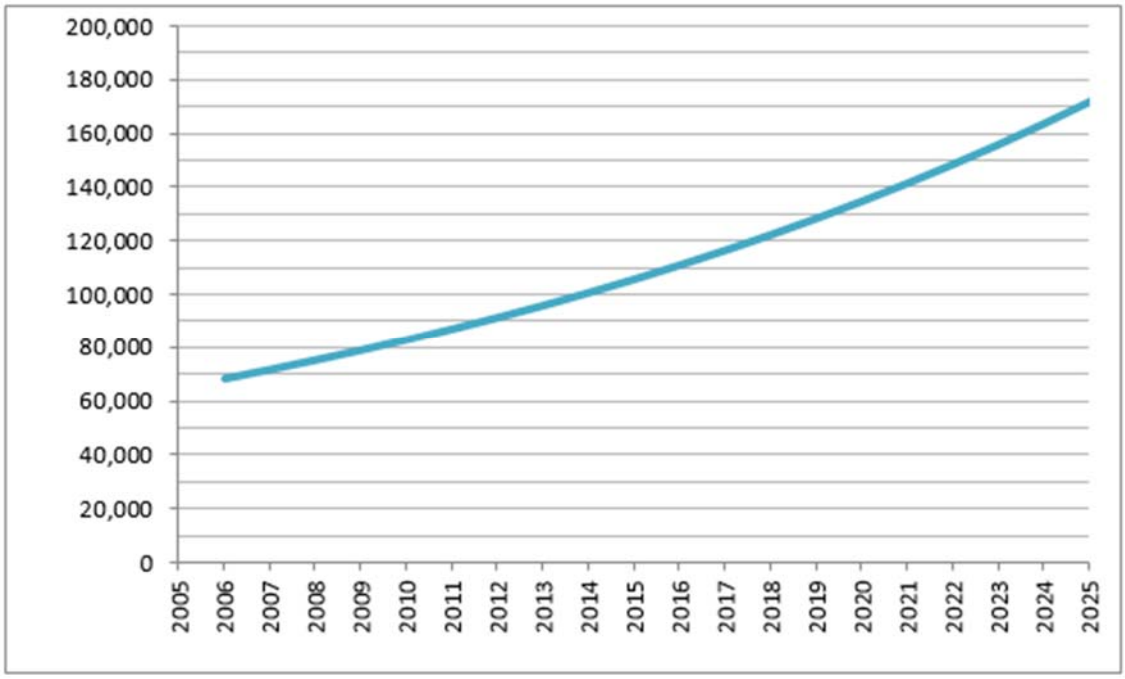

Figure 3. Forecast Cleared Software Workforce Size

The Productivity of Defense Software Developers

Malthus was wrong about hunger in England in large part because the technology for food production improved enormously over the next few centuries, making individual farmers much more productive and bringing marginal land into productive use. Could defense software development see (or already be seeing) a similar explosion in individual productivity that would be enough to make up for the slower growth of the labor force?

In 2000, Jones estimated defense software productivity at 4.2 function points (FP) per staff month (SM); in 2013, his estimate was 6.75 FP/SM. That corresponds to just under 4% annual productivity improvement. This is in line with other historical estimates of software productivity growth. For example, Longstreet (2001) estimated ~4% annual productivity growth (FP per hour) from 1970 to 2000 industry-wide. These estimates are based on FP, rather than on MSLOC. Since the number of FP per line of code has been growing historically (Jones, 2013), productivity growth in terms of MSLOC would be somewhat lower, but we will optimistically estimate MSLOC productivity growth at 4% as well.

Of course, the DoD may not yet have realized all of the productivity enhancement that can be had using current technology. We discuss these at greater length in the Recommendations section.

Supply vs. Demand

We now have all of the pieces we need for a back-of-the-envelope comparison of forecast productive capacity versus unconstrained demand. Figure 4 shows that, even given the generally optimistic assumptions we have made, we have already passed the point of being able to produce and maintain all of the software that the DoD would like. According to this forecast, the DoD will soon also reach the point of neither being able to produce all of the new code desired (without maintenance), nor to maintain all existing code (with no new development). The projected 2020 workforce of 135,000 developers would be less than half

of the 290,000 developers required to write and maintain all of the code desired up to that point.

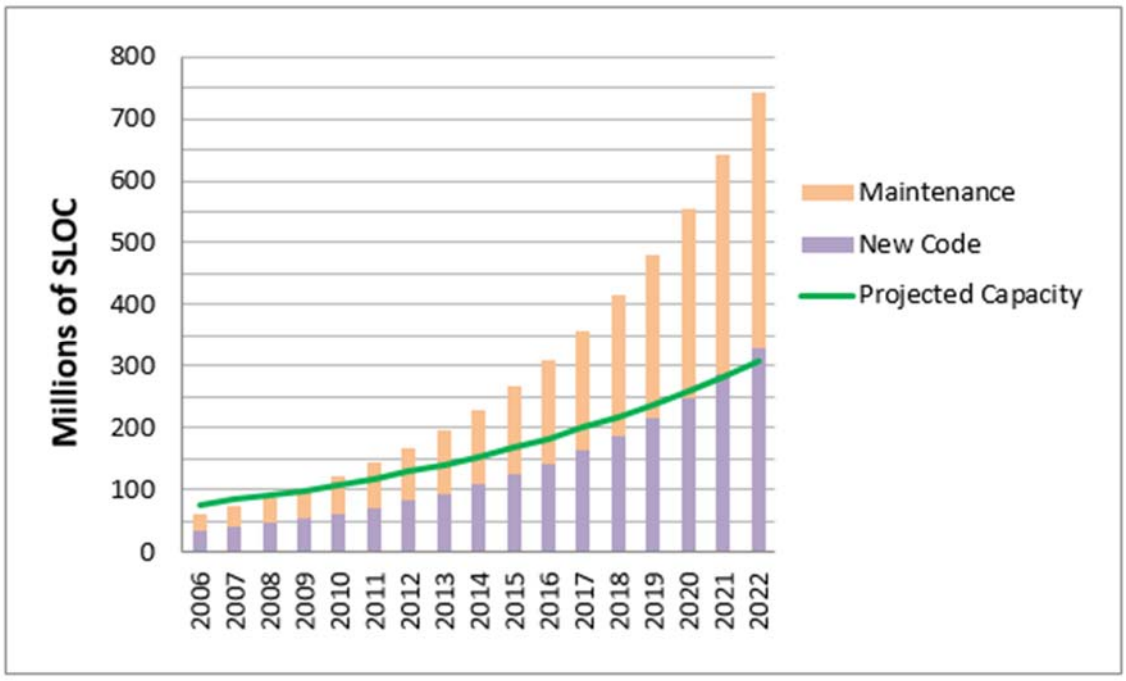

Figure 4. Forecast Supply vs. Unconstrained Demand

Revisiting the assumptions behind this forecast, we have assumed:

- 15% annual growth in demand for new code
- 5% annual defense software workforce expansion
- 4% annual productivity growth
- A workforce of 68,000 in 2006
- Demand for 35 MSLOC in 2006
- An installed base of 125 MSLOC in 2006
- Productive capacity of 75 MSLOC in 2006
- 20% annual maintenance effort
- 50% of maintenance resulting in new code
- 10% annual retirement of software in the base

Most of these assumptions could be fairly described as optimistic, based on historical data. Varying the parameters changes the details, but the shape of the situation remains the same. For example, if we assume that productivity growth post-2006 will be 8% instead of 4%, we get the picture in Figure 5. Software development is still capacity-constrained in this case, but not as severely. Conversely, if we keep productivity growth at 4% but allow the workforce to grow by 10% per year, we get the picture in Figure 6.

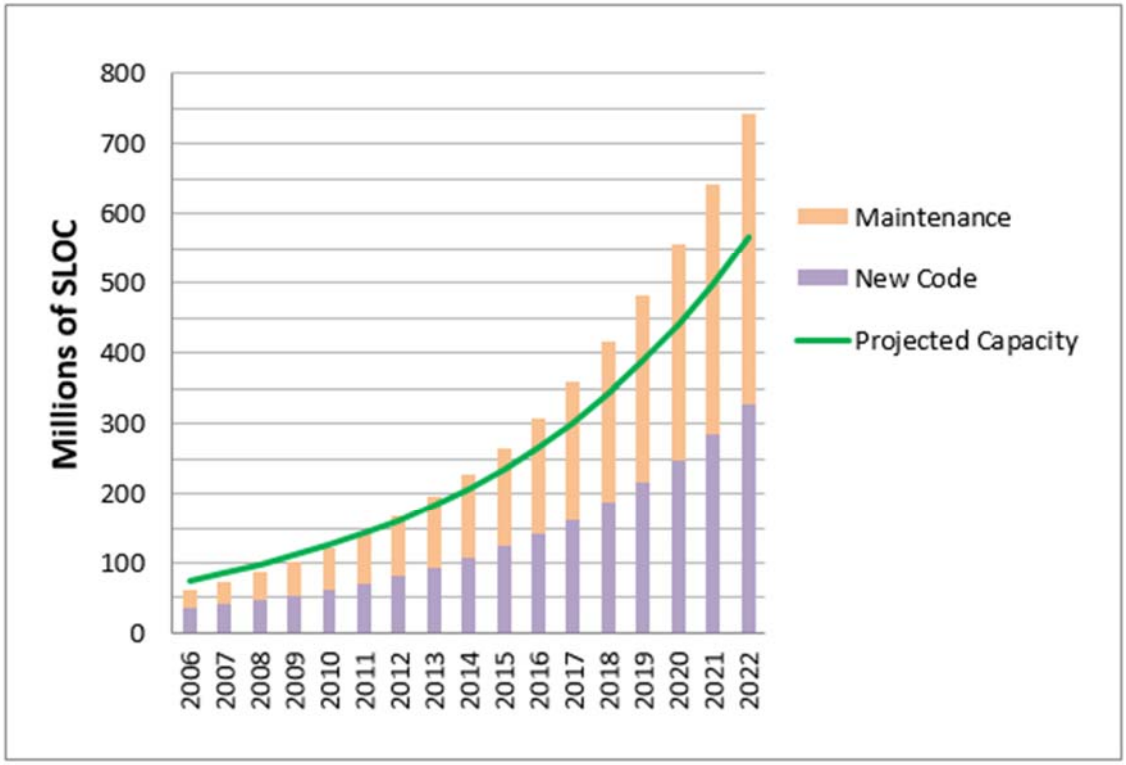

Figure 5. Supply vs. Demand at 8% Productivity Growth

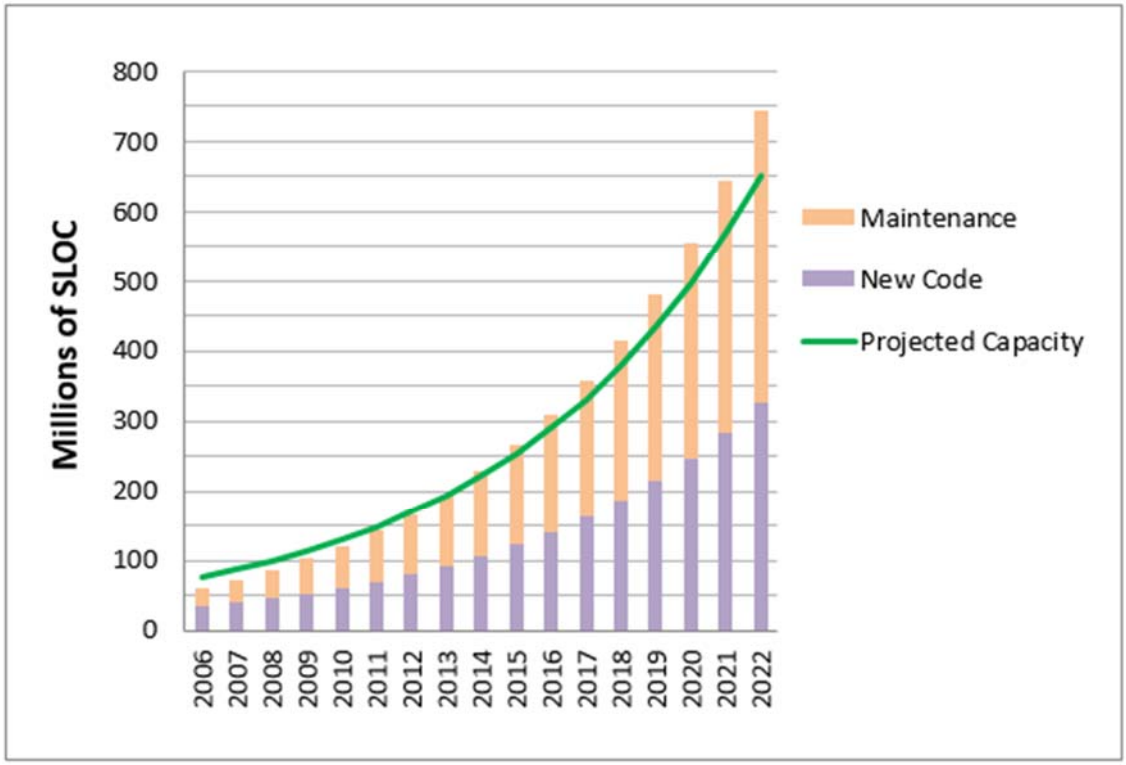

Figure 6. Supply vs. Demand Assuming 10% Workforce Growth

It goes without saying that the reverse is also true—if we assume 20% annual demand growth, or a 2006 installed base significantly larger than 125 MSLOC,[3] all of these pictures look much worse. Similarly assuming less optimistic values for the annual maintenance fraction (40%), or the proportion of maintenance that generates new code (>50%) (Galorath, n.d.), would lower the forecast capacity significantly.

If This Were Correct, Wouldn't We Have Noticed?

Is it really possible that we could be suffering a (possibly severe) shortage of software developers in the defense sector without anyone noticing? What symptoms should we look for?

Barnow, Trutko, and Piatak (2013) list 16 separate actions that employers might take that are indicative of a labor shortage. These include increased recruiting expenditures, increased use of overtime, new on-the-job training programs, relaxing minimum qualifications, and so forth. These are in addition to the operational symptoms of resource shortage, such as increased development times, lower-than-predicted staffing levels, and higher ratios of systems engineering/program management costs to touch labor costs.

Are these things happening in the defense sector? There is some evidence that they are.

- Chao (2006) found that senior software architects and project managers in the cleared software sector earned 50+% more than their counterparts in the general economy. They took this to indicate that there was already a shortage of those particular skills in the defense industrial base.
- Lucero (2009) found that many defense software positions were being filled by personnel with no formal software engineering training (on-the-job training).
- There are currently more than 10,000 job postings for software developers and software engineers at ClearanceJobs.com, which is more than half of all listings at that site (vacancies).
- Salaries for cleared information technology program/project managers rose 10% in one year between 2013 and 2014, faster than any other category and passing engineers as the highest-compensated cleared occupation group (salary rise) (ClearanceJobs.com, 2014).
- BLS estimates the national unemployment rate for technology professionals at only 2.9% (vacancies) (ClearanceJobs.com, 2014).
- Nearly half of recent ClearanceJobs.com survey respondents have been in their current job less than three years (churn) (Kyzer, 2017).

Barnow et al. (2013) also note that measuring occupational shortages is difficult, in part because occupational vacancy data are not generally available in the U.S. and available reporting uses job classification systems that are based on outdated industrial models and too coarse to be useful for many purposes. It would be very interesting to look

[3] Given that the Army alone claimed to have 240 MSLOC under sustainment in 2009, 125 MSLOC defense-wide in 2006 seems improbably low.

at (for instance) how the cost per staff month of defense software development has changed over the past decade, as reflected in the Software Requirements Data Reporting database.

What Are the Policy Options?

We identify several available short-term and long-term policy options associated with both the supply and demand for defense software.

Option 1: Moderate Demand

The obvious short-term solution to a scarcity of software productive capacity is to ask for less software. At the present time, it seems unlikely that the defense establishment would be willing or able to accomplish this. Software is viewed as vital to any hope of maintaining the United States' traditional technological advantage in military capability. A significant overall reduction in software demand would also require the several Services to cooperate effectively to optimize the allocation of software development capacity to the most important software-intensive programs. Given that the services struggle to allocate resources efficiently within and among their own acquisition portfolios, this seems like a stretch. The results, then, would be a less-efficient allocation of software resources to capabilities, an associated effective loss of software productivity, and failure to reap the potential benefits of software-mediated capabilities.

In the longer term, natural factors limit the growth in demand for software. Defense budgets do not grow without limit, so the exponential growth in software demand reflects, to some extent, substitution of software for other categories of expenditure—primarily analog hardware and human labor. There are natural limits to that process. Regardless of the underlying desire for software-mediated capabilities, the DoD cannot procure more software than the industrial base is able to provide.

Perhaps just as importantly, there is a tension between the size and complexity of the software in a system and how long it takes to develop that system. If rapid response to a rapidly changing world is one of the motivations for implementing capabilities in software, it makes no sense to pursue designs whose complex software will require 20+ years to design, build, and test. Prior analysis of the dependence of development cycle times on software content assumed development times unconstrained by industrial base issues (Tate, 2016). If Major Defense Acquisition Program/Major Automated Information System (MDAP/MAIS) software projects are now subject to chronic resource shortfalls, those past lead time estimates were optimistic. Increased demand for software-mediated functions thus has a twofold negative effect on schedules: first by adding work to the critical development path of each program, and second by starving the programs of the resources necessary to do the work on the critical path. From a policy perspective, it does not seem practical for the DoD or Congress to mandate reduced use of software overall, or limits on the amount of software in any one program. Not only would those policies be counterproductive, they would also be unenforceable and prone to wasteful gaming by the services and defense contractor base. Demand-side policy options would not seem to be helpful here.

Option 2: Grow the Workforce

From a policy perspective, there are several plausible mechanisms for increasing the growth rate of the defense software base:

- Encourage students to pursue software education, both through traditional college degrees and nontraditional (e.g., boot camp) training programs. Incentives could include low-interest loans, direct subsidies/scholarships, loan forgiveness, etc. These could be made contingent on a minimum tenure of employment in the defense sector.

- Invest in improving the throughput of the security clearance process, especially for software workers.
- Relax barriers to employing foreign nationals. The software industry has thoroughly globalized, but the defense sector is not permitted to take advantage of that at present. As we shall see below, there are ways of doing this implicitly that do not involved relaxing security standards.
- Allow contractors to pay true market salaries for software talent.

The first three of these options would tend to reduce the price of defense software by increasing supply, thus somewhat offsetting the investment required. Allowing higher salaries for key software professionals looks like it would tend to increase the cost of any given system—but it might not. It might improve efficiency and increase supply by enough to offset the higher cost per hour of that labor. It might also make it possible to have that system at all, or improve its quality, or permit the DoD to acquire it in time for it to be useful.

Option 3: Improve Productivity Dramatically

There have been multiple drivers of significant productivity improvement in the commercial software world over the past few decades. These include computer-aided software engineering (CASE) tools, automated test environments, improved programming languages,[4] agile (and similar) development processes, and modular open system architectures. The defense software base has participated in the first three (though the use of improved programming languages was long delayed by the mandate to write in Ada), but it has not leveraged the last two nearly as much.

Definitions of "agile development" invariably lead to arguments among both advocates and skeptics, but in general the phrase refers to a strategy of rapid, small-scale, incremental development and release of software functionality, driven not by prespecified requirements or specifications but rather by close, iterative interaction with future users of the software being developed. The key features here are as follows:

- **Small**—Features are added in many small increments, rather than a few large blocks/versions/updates.
- **Rapid**—New releases happen on a scale of weeks, not months or years.
- **No fixed requirements**—Users and developers together explore the space of potential features and discover which are the most useful.
- **Interactive**—Users and developers participate as a collaborative partnership, rather than as customer and vendor, with developers in self-organizing teams.

All of these key features pose problems for traditional DoD acquisition. Having many small incremental releases of functionality breaks the logistics system whereby new software releases are coordinated and deployed to far-flung operational units. The absence of fixed formal requirements is antithetic to the Joint Requirements Oversight Council (JROC) mission of specifying formal, validated requirements with threshold levels. It may also cause legal and practical headaches for the writers of requests for proposals and the awarders of contracts, not to mention cost and schedule estimators. The interaction

[4] For our purposes, *improved* simply means more FP of product per SM of effort on average.

between developers and users requires active, ongoing participation of uniformed and civilian personnel who would traditionally never get near the system under development until (perhaps) Operational Test and Evaluation. That ongoing collaboration might last for years.

The other dominant recent development in the commercial world that has generated significant productivity gains is the use of modular open system architectures. Stephen Welby (2014), during his time as Deputy Assistant Secretary of Defense for Systems Engineering, described these as "technical architectures that leverage technical standards to support a modular, loosely coupled and highly cohesive system structure." There are actually two distinct and separately important ideas here: modularity, which is about the way the software's functions are organized into independent composable units, and openness, which is about who can see, modify, publish, or use the code. Not all modular architectures are open; not all open source software is modular. There is a synergy between the two ideas, however—modularity increases the efficiency of individual contributions to the open code base, while openness allows more individuals to contribute.

For our purposes, the key features that drive enhanced productivity are the following:

- Composable software modules that can be combined in many ways to execute more complex functions
- Well-defined, standardized, documented interfaces for these modules
- Universal transparent access to (nearly) all of the source code
- Extensive rights to modify or enhance existing source code
- A large base of independent agents actively engaged in developing/improving the set of modules

Examples of thriving modular open software ecosystems include the Linux operating system, the Apache web hosting platform, the FreeRTOS real-time operating system for embedded systems, the R and Python programming environments, the emacs document editor, and the MySQL relational database. The collaborative nature of the communities of developers working with these tools can lead to enormous total effort—the Linux Foundation estimated in 2008 that the total cost to develop the Fedora 9 distribution of Linux (including the Linux kernel itself) from scratch would have been more than $12 billion (McPherson, Proffitt, & Hale-Evans, 2008). That was nearly a decade of additional development ago.

Modular open architectures enhance productivity through three principal mechanisms: reuse, parallelism, and scrutiny. Modularity allows large parts of the code base to be reused in new applications with little or no modification, greatly reducing development times. It also makes it easier for program managers to decompose complex development projects into weakly-dependent subprojects, so that less work lies on the critical path. Openness, on the other hand, invites large numbers of developers to work on continuous improvements to the ecosystem, so that there is an ever-richer set of existing modules to re-use. This widespread active attention to improving the code in turn results in a higher level of scrutiny—and thus generally lower defect rates—for frequently-used modules in such environments (Brockmeier, 2003). Similarly, software assurance and cybersecurity can be easier for open source software than for proprietary software (Wheeler, 2010).

The openness and transparency of open source ecosystems also provides a welcome indirect mechanism for opening defense software development to the non-cleared workforce. Any defense software that is based on Linux, or written in Python, or implemented using FreeRTOS, is leveraging the efforts of thousands of developers outside the usual defense workforce. In the end, this might be the best argument in favor of modular

open source—that it promises not only significant productivity gains for individual programmers, but also the largest available expansion of the defense software workforce.

Recommendations

Thus far, we have presented some plausible guesstimates concerning actual supply and demand, some optimistic yet sobering forecasts, and an enumeration of possible policy options. Given all of that, what should be done?

First, collect data. Study the industrial base; measure the effective demand; measure the maintenance efforts. The forecasts in this paper are built on sparse data from inconsistent sources. An improved update to the Chao (2006) investigation of the state of the defense software industrial base is long overdue, and could replace those credible guesstimates with actionable facts. If we discover that supply has kept pace with demand just fine over the last decade, good. We will have learned something about how the unique defense labor market responds to internal demand surges and competition from the commercial market. If, however, we discover that significant amounts of software maintenance are being deferred, all projects are understaffed, and new programs are executing by stealing from existing programs, then we can sound the alarm.

Second, adopt commercially proven productivity-enhancing acquisition models. In recent decades, the DoD has bet that the boom in commercial software is a rising tide that would lift defense software productivity as well. This turned out not to be true; the needs of the DoD are sufficiently different from those of the commercial world that productivity advances arising in the commercial sector did not necessarily translate to the defense sector. Agile development and large-scale telework are good examples of productivity multipliers in the commercial sector that are not as useful for defense without significant adaptation.

In particular, embrace modular open source software ecosystems. Of the known productivity enhancers, this is the only one that might potentially provide both ongoing rapid productivity growth and an effective expansion of the workforce. Doing this would require substantial regulatory, cultural, organizational, and perhaps legal changes across the defense acquisition enterprise and the defense industrial base. There is also a nonzero risk that such efforts could fail to produce the critical mass of actively engaged developers necessary to realize the benefits of open source ecosystems. Evidence from the commercial world suggests that not only would it be worth the risk, it might be necessary in order to keep up with the pace of technology change and threat evolution. DoD leadership have been pushing in this direction (DoD, 2017), but there is considerable institutional inertia and active resistance to be overcome, both within government and within the industrial base. Furthermore, the early stages of developing such ecosystems might well not look much like progress.

Finally, fund basic productivity research the way the DoD used to do. Without fundamental improvements in software productivity, weapon system capabilities will be limited by the time it takes to develop new software-intensive systems, and that limit may not be very far beyond what is currently being produced. The DoD has little use for highly capable systems that take 25 years to field. In the long run, the key breakthrough will be the automation of software development as a process, so that it is no longer a manual craft labor activity. That vision—autonomous systems writing software from scratch with the dependability required of defense systems—is currently still in the realm of science fiction.

References

Arrington, M. (2010a). Google making extraordinary counteroffers to stop flow of employees to Facebook. Retrieved from https://techcrunch.com/2010/09/01/google-making-extraordinary-counteroffers-to-stop-flow-of-employees-to-facebook/

Arrington, M. (2010b). Google offers staff engineer $3.5 million to turn down Facebook offer. Retrieved from https://techcrunch.com/2010/11/11/google-offers-staff-engineer-3-5-million-to-turn-down-facebook-offer/

Barnow, B. S., Trutko, J., & Piatak, J. S. (2013). How do we know occupational labor shortages exist? *Employment Research, 20*(2), 4–6. doi:10.17848/1075-8445.20(2)-2

Brockmeier, J. (2003). Comparing free and proprietary software defect rates. Retrieved from https://lwn.net/Articles/22623/

Bureau of Labor Statistics (BLS). (n.d.). *Occupational outlook handbook, 2016–17 edition.* Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

Bureau of Labor Statistics (BLS). (2017). Occupational employment statistics. Retrieved from https://www.bls.gov/oes/tables.htm

Chao, P. (2006, October). An assessment of the national security software industrial base [Briefing]. Retrieved from https://www.csis.org/analysis/assessment-national-security-software-industrial-base

ClearanceJobs.com. (n.d.) Retrieved from https://www.clearancejobs.com/jobs?keywords=software

ClearanceJobs.com. (2014). In the clear: Compensation decline for security-cleared professionals levels off despite strong headwinds. Retrieved from https://about.clearancejobs.com/hubfs/pdfs/ClearanceJobs_Compensation_Survey_2014.pdf

Department of the Army. *Army Organic Industrial Base Strategic Plan: 2012–2022.* Retrieved from https://www.army.mil/e2/c/downloads/276549.pdf

DoD. (2017, February 23). DoD announces the launch of “Code.mil,” an experiment in open source (No. NR-077-17) [Press release]. Retrieved from https://www.defense.gov/News/News-Releases/News-Release-View/Article/1092364/dod-announces-the-launch-of-codemil-an-experiment-in-open-source?source=GovDelivery

Dvorak, D. L. (Ed.) (2009). *NASA study on flight software complexity*. Pasadena, CA: NASA Jet Propulsion Laboratory, California Institute of Technology.

Galorath, Inc. SEER by Galorath. Retrieved from http://galorath.com/software_maintenance_cost

Jones, C. (2000). *Software assessments, benchmarks, and best practices*. Boston, MA: Addison-Wesley.

Jones, C. (2013). Function points as a universal software metric. *ACM SIGSOFT Software Engineering Notes, 38*(4), 1–27. doi:10.1145/2492248.2492268

Kyzer, L. (2017). Clearance salary trends in 2017. Retrieved from https://news.clearancejobs.com/2017/01/19/clearance-salary-trends-2017/

Lauerman, J. (2015). Coding boot camp enrollment soars as students seek tech jobs. Retrieved from https://www.bloomberg.com/news/articles/2015-06-08/coding-boot-camp-enrollment-soars-as-students-seek-tech-jobs

Longstreet, D. (2001). Software productivity since 1970. *Function point training and analysis manual.* Retrieved from http://www.softwaremetrics.com/Articles/history.htm

Lucero, D. (2009, April). *Software sustainment challenges in defense acquisition*. Presented at the AIAA Infotech@Aerospace Conference, Seattle, WA. doi:10.2514/6.2009-1816

McPherson, A., Proffitt, B., & Hale-Evans, R. (2008). Estimating the total cost of a Linux distribution. Retrieved from https://www.linux.com

/publications/estimating-total-cost-linux-distribution

National Research Council, Committee for Advancing Software-Intensive Systems Producibility. (2010). *Critical code: Software producibility for defense*. Washington, DC: National Academies Press. doi:10.17226/12979

System Architecture Virtual Integration (SAVI). Retrieved from http://savi.avsi.aero/about-savi/savi-motivation/exponential-system-complexity/

Tate, D. M. (2016). *Software development may drive future acquisition cycle times* (IDA Research Insights NS D-8053). Retrieved from https://www.ida.org/idamedia/Corporate/Files/Publications/Insights/CARD-D-8053-Tate.ashx

Welby, S. P. (2014, October). *Modular open systems architecture in DoD acquisition*. Presented at the 17th Annual NDIA Systems Engineering Conference, Springfield, VA.

Wheeler, D. A. (2010, April). *Open source software (OSS/FLOSS) and security*. Presented at the International Workshop on Free/Open Source Software Technologies, Riyadh, Saudi Arabia. Retrieved from https://www.dwheeler.com/essays/oss_security_saudi_arabia.pdf

Acknowledgments

This work was performed under the IDA Central Research Program, supported by Department of Defense contract HQ0034-14-D-0001.

Acquisition Research Program
Graduate School of Business & Public Policy
Naval Postgraduate School
555 Dyer Road, Ingersoll Hall
Monterey, CA 93943

www.acquisitionresearch.net