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 as demand. This article uses the limited data that exist regarding defense software supply, demand, and productivity trends to estimate the severity of the capacity bottleneck, then briefly discusses the potential actions available to the Department to mitigate that bottleneck in the long run.

DOI: https://doi.org/10.22594/dau.19-838.27.02

Keywords: Defense Industrial Base, Capacity, Demand Growth, Supply Shortfall, Cleared Workforce
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 in his time, but could he be right about defense software? Any exponential growth in demand without a commensurate exponential growth in supply will soon be frustrated. Rapidly growing demand for new software, combined with the need to sustain the new 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 determine whether DoD should worry, we looked at each of those factors in turn.

Unfortunately, data on defense software demand and the defense software industrial base are extremely sparse. The last comprehensive attempt to estimate the total demand for national security software and the capacity of the industrial base to meet that demand is more than a decade out of date (Chao, 2006). Data collection for the DoD software effort consists primarily of Software Resources Data Reports (SRDR) (Office of the Secretary of Defense, 2019), which are generally limited to major programs, not currently curated or normalized to a degree that can support DoD-level analysis, and subject to severe data quality issues (Arnold, Braxton, & Wingrove, 2015; Morin, 2017).

At the same time, measuring software productivity is notoriously difficult, and such standards as exist are neither widely used nor consistently applied (Card, 2006; Krishnan, Kriebel, Kekre, & Mukhopadhyay, 2000).
In this article, we extrapolate from that last comprehensive baseline using a variety of more recent data sources to make plausible assumptions regarding changes in demand, workforce, and productivity. We find that even conservative estimates of these trends lead to a conclusion that, absent significant changes, future defense capabilities will be severely limited by the available productive capacity of the cleared software workforce.

**How Fast Is Defense Demand for Software Growing?**

Among the data available to support analysis of demand growth in defense software, some strong indicators are available:

- The National Research Council (2010) wrote “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 (n.d.) states that source lines of code (SLOC) in aircraft (both military and commercial) have been doubling approximately every 4 years. This corresponds to an annual growth rate of about 18%.

- The Department of 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 about 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 to 25% means doubling every 3 to 5 years, on top of which is the added workload of maintaining the growing base of deployed code.
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. Surprisingly, this information is apparently not being tracked. The most recent nationwide 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. 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 that follow are not sensitive to the exact parameter values chosen here, or the estimate of the current installed code base.

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 DoD is expected to want to buy, if it is available at prices comparable to historical prices.
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 update these estimates using such data as are available. For purposes of this analysis, we will accept Chao’s estimates that 68,000 software developers possessed security clearances in 2006, with a productive capacity of 75 MSLOC per year. That implies a productivity at that time of roughly 1,100 SLOC per developer in 2006, or (equivalently) 900 developers required per MSLOC, as our baseline.

The Size of the Workforce

The Bureau of Labor Statistics (BLS, 2017) estimates that from 2010 to 2015, total employment of software developers grew almost 30%, or about 5.3% annually. However, it forecasts that rate to decline sharply going forward, averaging only about 1.6% per year over the decade of 2014–2024 (BLS, 2016–2017). The defense software industrial base will need to grow more quickly than that to keep pace with established demand growth.

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.

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; it also creates a licensure bottleneck for individuals seeking to join the labor force. In April 2019, the Director of the National Background Investigations Bureau reported that the backlog for security clearance investigations had been reduced by nearly one-third but was still nearly half a million cases. Processing times remained long, with initial investigations averaging 234 days for Secret clearances and 468
days for Top Secret (Kyzer, 2019). Hiccups in overall growth of the cleared workforce in the last decade have also emerged, driven by government response to high-profile leaks and worker response to the breach of Office of Personnel Management personal information files (Kyzer, 2015). Defense software employers are also facing tough competition from the commercial sector, which is experiencing an explosion of demand for software to power the expanding role of the Internet in daily life. While other industries can supplement U.S. graduates with offshore or immigrant labor, that solution is 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) no general shortage of programmers existed, but a significant shortage (with corresponding salary premium) of relatively senior software project managers, architects, and developers was already apparent. At the tip of the pyramid, they cited a cadre of 500–600 “elite” individuals who play a disproportionate role in project success.

Finally, it is not clear that DoD wants the market to correct itself through increases in compensation. Contractor labor rates are closely monitored by 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. Reimbursable federal contractor labor costs are capped by the provisions of Section 702 of the Bipartisan Budget Act of 2013 (Pub. L. 113-67); companies choosing to pay salaries higher than that cap ($525,000 in 2018) must take the difference out of profit (Office of Federal Procurement Policy, 2018). This provides a disincentive to employing top software talent on federal contracts. For federal civilian employees, the permitted salaries are even more tightly constrained.

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 4 decades (Figure 2). The general trend has been a baseline increase of about 1,000 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](image)

**Note.** Only every other academic year is labeled. Source: National Center for Education Statistics (2012, 2016).

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 as many as 80,000 potential developers are graduating per year. In 2006, the cleared software workforce made up 7% of the national software workforce and 16% of the overall cleared workforce (Chao, 2006). Again, optimistically, 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.

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).

As noted in the preceding discussion, in 2006, roughly 68,000 cleared software developers were employed 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.
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. A comparable technology revolution in software productivity could offset the growth in software demand, even if the workforce grows only slowly. Are there signs of such growth in individual productivity?

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 about 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, DoD may not yet have realized all of the productivity enhancement that can be had using current technology. The potential for leap-ahead productivity improvements—analogous to the farming breakthroughs that Malthus failed to foresee—is discussed in the Recommendations section.

Supply vs. Demand

We now have all of the pieces we need for an end-to-end estimate of future productive capacity versus projected unconstrained demand. Figure 4 shows that, even using generally optimistic assumptions and estimates, demand already exceeds the capacity of the industrial base. According to this forecast, 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.

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, given the available historical data. Varying the parameters changes the details of the forecast, but the bottom line remains the same. For example, if we assume that productivity growth post-2006 will be 8% instead of 4%, we get the results 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 results in Figure 6.

Necessarily, the reverse is also true—if annual demand growth is closer to 20%, or the 2006 installed base was significantly larger than 125 MSLOC, or cleared workforce growth stagnates, then all of these pictures
would look much worse. Assuming less optimistic values for the annual maintenance fraction (40%), or the proportion of maintenance that generates new code (>50%) (Galorath, n.d.), would also lower the estimated future capacity significantly.
If This Were Correct, Wouldn’t Someone Have Noticed?

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

Barnow, Trutko, and Piatak (2013) list 16 separate employer actions that might indicate a labor shortage. These include increased recruiting expenditures, increased use of overtime, new on-the-job training programs, relaxing minimum qualifications, etc. 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.

Evidence of these indicators is already present in the defense sector.

- Chao (2006) found that senior software architects and project managers in the cleared software sector earned at least 50% more than their counterparts in the general economy. They took this to indicate that those particular skills were already in short supply throughout 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).


- As of late 2019, ClearanceJobs.com had nearly 30,000 job postings for software developers, software engineers, or software managers. This was nearly half of all listings at that site.

- 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 (ClearanceJobs.com, 2014, Salary Rise).
BLS estimates the national unemployment rate for technology professionals at only 2.9% (ClearanceJobs.com, 2014, Vacancies).

Nearly half of recent ClearanceJobs.com survey respondents have been in their current job less than 3 years (Kyzer, 2017, Churn).

Barnow et al. (2013) also note that measuring occupational shortages is difficult, in part because occupational vacancy data are not generally available in the United States. Also, available reporting uses job classification systems that are based on outdated industrial models and are too broad to be useful for many purposes. It would be very interesting to look at (for instance) how the aggregate cost per staff month of defense software development has changed over the past decade, as reflected in SRDR reporting of major programs.

**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. In the long run, success may depend on breakthroughs in the last and most speculative of the options—investment in the transition from a craft labor model to an industrial automation model of software development.

**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 federal agencies that procure national security software to cooperate effectively to optimize the allocation of software development capacity to the most important, software-intensive programs. Given that these agencies 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, DoD cannot procure more software than it can afford, or than the industrial base is able to provide.

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 or more to design, build, and test.

Perhaps just as important, the size and complexity of the software in a system affects how long it takes to develop and field 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 or more 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 DoD or Congress to mandate reduced use of software overall, or to set 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 appear to be unhelpful here.
Option 2: Grow the Workforce

From a policy perspective, several plausible mechanisms are available to increase the effective 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. Incentives like this have been successful in increasing primary care physician recruitment in underserved areas (Verma et al., 2016).

- Continue to invest in improving the throughput of the security clearance process, especially for software workers. While this has been a priority for DoD in recent years, progress has been slow (Kyzer, 2019).

- 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 in the discussion that follows, increased use of open source software accomplishes this implicitly without relaxing security standards.

- Adjust restrictions on allowable contract costs 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 reimbursable 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 expert labor. It might also improve the availability, timeliness, and quality of delivered systems.

Option 3: Improve Productivity Dramatically

Multiple drivers of significant productivity improvement have permeated the commercial software world over the past few decades. These include computer-aided software engineering (CASE) tools (Krishnan et al., 2000), automated test environments (National Institute of Standards and Technology, 2002), improved programming languages (Jones, 2013, Table 14.2), agile (and similar) development processes, open source ecosystems (Lerner, 2010), and modular open system architectures. The
defense software base has shared in the benefits of CASE tools and automated test environments, and to some extent from open source software (Wheeler, 2010). Improved productivity through programming language modernization was temporarily delayed during the 1990s by the mandate to write defense software in Ada, and continues to be hampered by the large installed base of defense software in obsolescent languages such as COBOL, FORTRAN, and HAL/C. DoD has not yet leveraged agile development practices or modular open architectures to a significant degree (Defense Innovation Board [DIB], 2019).

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 final requirements or specifications, but rather by close, iterative interaction with future users of the software. The key features here are:

- 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, developers, and other stakeholders together explore the space of potential features and discover which are the most useful
- Interactive—stakeholders and developers work 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 (Broadus, 2013). 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 awardees of contracts, not to mention cost and schedule estimators. The interaction 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.
Open source software refers to software that is collaboratively developed and maintained by a community of volunteer contributors. Examples of thriving open source 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. For example, the Linux Foundation estimated in 2008 that the total cost to develop from scratch the Fedora 9 distribution of Linux (including the Linux kernel itself) would have been more than $12 billion (McPherson, Proffitt, & Hale-Evans, 2008). That was nearly a decade of additional development ago.

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” (p. 3). Note that modularity and openness are distinct concepts, each of which contributes separately to potential productivity improvements. Modularity is about the way the software’s functions are organized into composable units. Openness is about who can see, modify, publish, or use the code. Not all modular architectures are open; not all open systems are modular. However, a synergy exists between the two ideas—modularity increases the efficiency of individual contributions to the open code base, while openness allows more competition and participation.

For DoD, the following key features could drive enhanced productivity:

- Composable software modules that can be combined in many ways without modification 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, and maintaining the software without being directly paid by the eventual users
Modularity would allow large parts of the code base to be reused in new applications with little or no modification, greatly reducing development times. It would also make future upgrades faster and cheaper. Judicious use of open source software, on the other hand, would enable DoD to take advantage of a huge body of well-maintained and inexpensive software that already exists. Studies have shown that the community approach to development results in a higher level of scrutiny—and thus, generally lower defect rates—for frequently used modules in such ecosystems (Brockmeier, 2003). Similarly, software assurance and cybersecurity can sometimes be easier for open source software than for proprietary software (Wheeler, 2010).

Open source ecosystems also provide a potential indirect mechanism for opening defense software development to the noncleared 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 open source—that it promises not only significant productivity gains from leveraging existing commercial software, but also the largest available effective expansion of the defense software workforce.
Recommendations

Thus far, we have seen estimates of supply and demand, some optimistic yet sobering forecasts, and an enumeration of possible policy options. These lead naturally to three principal recommendations:

Recommendation 1: Collect data.

Study the industrial base; measure the effective demand; measure the installed code base and ongoing maintenance efforts; measure industrial base capacity and productivity. Indeed, formulating sensible strategy without this basic information is impossible.

The forecasts in this article are built on sparse data from inconsistent sources, because those are the only data that exist. An improved update to the Chao (2006) investigation of the state of the defense software industrial base is long overdue and would enable DoD to replace the very uncertain estimates in this article with actionable facts.

Recommendation 2: Adopt commercially proven, productivity-enhancing acquisition models.

In recent decades, DoD has bet that the boom in commercial software is a rising tide that would necessarily lift defense software productivity as well. As documented throughout this article, this turned out to be only partially true; the needs and culture of DoD acquisition are sufficiently different from those of the commercial world that some productivity advances arising in the commercial sector did not automatically translate to the defense sector. For example, agile development methods, though greatly desired by both DoD and external advisors (DIB, 2019), will require radical changes in requirements management, stakeholder involvement in development, and acquisition planning and budgeting processes (Broadus, 2013). Similar institutional and cultural barriers prevail that oppose or seek to limit expanded use of open source software and modular open systems architecture. 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.

Recommendation 2A: In particular, embrace open source software ecosystems. Of the existing productivity enhancers, this is the only one that might potentially provide both immediate rapid productivity improvement and an effective expansion of the workforce. Evidence from the commercial world suggests that embracing open source software would not only
be cost-effective, it might be necessary in order to keep up with the pace of technology change and threat evolution. Furthermore, the early stages of developing such ecosystems might well not look much like progress.

**Recommendation 2B:** Enforce modular architectures and pursue data rights to enable fast and efficient insertion of future upgrades into legacy platforms and systems. One of the best ways to improve future productivity is to build systems that are easy to modify. When major platforms are in service for decades, the ability to host the essential capabilities of the future is more important than the requirement that they deliver maximum capability the day they are fielded (Patel & Fischerkeller, 2013).

**Recommendation 3: Fund basic productivity research the way DoD used to do.**

The federal government, and in particular DoD, played an enormous role in funding and guiding the development of core software technologies that enabled U.S. dominance in that industry for the first few decades of the computer age (Mowery, 1999). Without additional fundamental improvements in software productivity, DoD’s ability to field needed capabilities quickly enough to keep pace with changing threats will be limited by the time it takes to develop the software that implements those capabilities. DoD is already struggling to keep pace; major leaps forward are needed.

In particular, focus research on freeing software development from its current 19th-century industrial model. At present, each software application written is the custom hand-tooled product of skilled craftsmen, analogous to the way automobiles were made before Henry Ford revolutionized that industry. In the long run, the key productivity breakthrough must be the automation of software development as a process, enabling mass production and industrialization. Software product lines (Hinchey, 2018) are a start, but the transformational end-state, with autonomous systems with only high-level human guidance writing software from scratch with the dependability required of defense systems, is currently still in the realm of science fiction.
References


Endnotes

1 “National security software” is taken to include software used for national security applications and missions by the Department of Defense, Department of Energy, and Department of Homeland Security. At present, defense software is by far the largest portion of national security software.

2 Jones (2013) estimates the maintenance costs of a nominal 1000-function point application at closer to 40% per year over the first 5 years. Using that estimate would result in a smaller 2006 deployed code base estimate, but much faster growth in that base in subsequent years.


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

5 For our purposes, “improved” simply means more Function Points (or lines of code) of product per staff month of effort, on average.

Acknowledgments

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

Dr. David Tate

joined the research staff of Institute for Defense Analyses’ Cost Analysis and Research Division in 2000. Prior to that, he was a Senior Operations Research Analyst (Telecom) at Decision-Science Applications and an Assistant Professor of Industrial Engineering at the University of Pittsburgh. Dr. Tate holds bachelor’s degrees in Philosophy and Mathematical Sciences from the Johns Hopkins University, and MS and PhD degrees in Operations Research from Cornell University.

(E-mail address: dtate@ida.org)