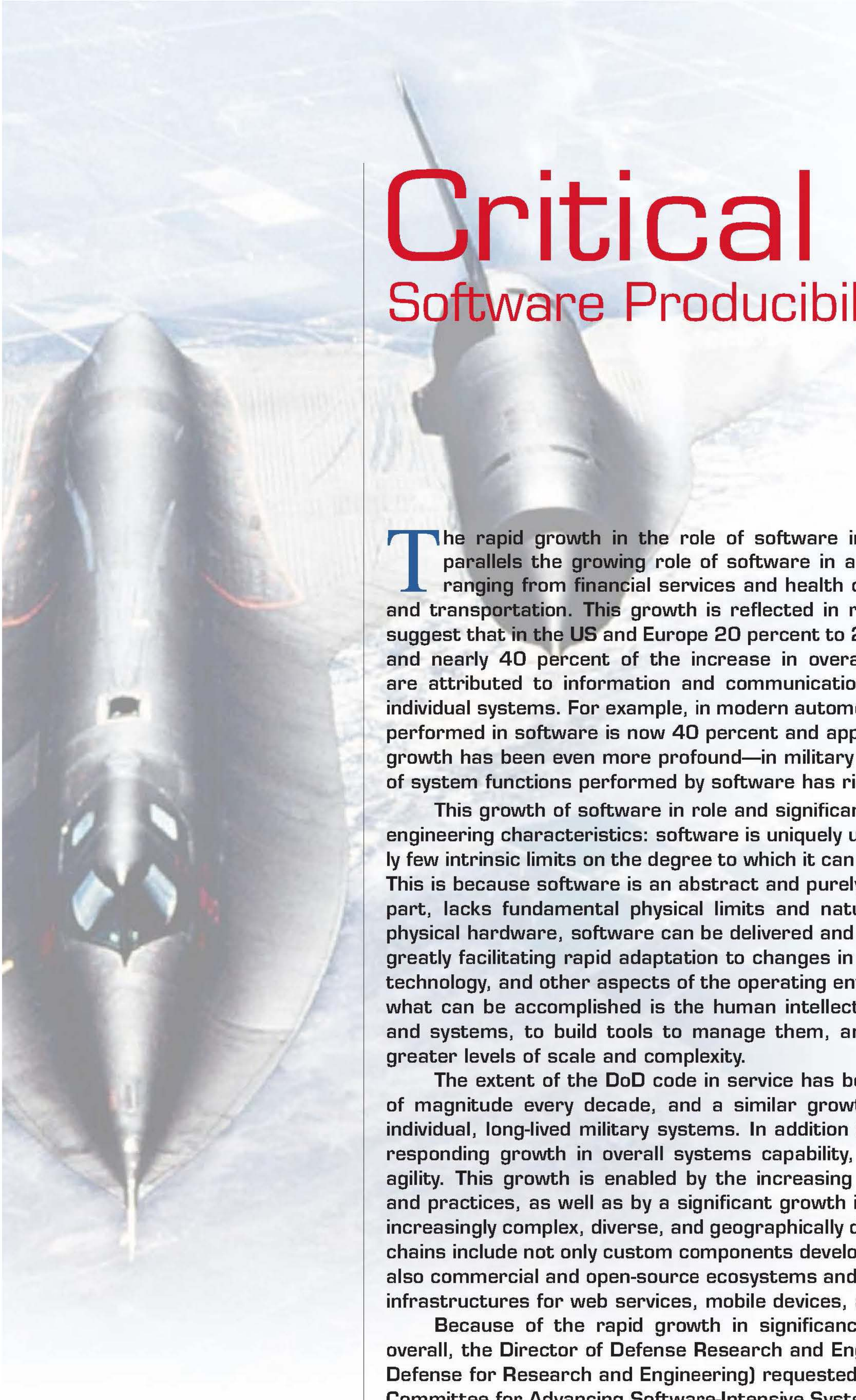


Critical Code: Software Producibility for Defense

A Short Summary

The rapid growth in the role of software in defense systems is significant and parallels the growing role of software in a broad range of application domains, ranging from financial services and health care to telecommunications, logistics, and transportation. This growth is reflected in recent macroeconomic studies, which suggest that in the US and Europe 20 percent to 25 percent of overall economic growth and nearly 40 percent of the increase in overall economic productivity since 1995 are attributed to information and communications technology. It is also reflected in individual systems. For example, in modern automobiles, the portion of system functions performed in software is now 40 percent and approaching 50 percent. In the DoD, the growth has been even more profound—in military aircraft, for example, the percentage of system functions performed by software has risen to more than 80 percent.

This growth of software in role and significance is a natural outcome of its special engineering characteristics: software is uniquely unbounded and flexible, having relatively few intrinsic limits on the degree to which it can be scaled in complexity and capability. This is because software is an abstract and purely synthetic medium that, for the most part, lacks fundamental physical limits and natural constraints. For example, unlike physical hardware, software can be delivered and upgraded electronically and remotely, greatly facilitating rapid adaptation to changes in adversary threats, mission priorities, technology, and other aspects of the operating environment. The principal constraint on what can be accomplished is the human intellectual capacity to understand problems and systems, to build tools to manage them, and to provide assurance—all at ever-greater levels of scale and complexity.

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. In addition to this growth in size, there is a corresponding growth in overall systems capability, complexity, interconnectedness, and agility. This growth is enabled by the increasing power of software languages, tools, and practices, as well as by a significant growth in the dependence of DoD systems on increasingly complex, diverse, and geographically distributed supply chains. These supply chains include not only custom components developed for specific mission purposes, but also commercial and open-source ecosystems and components, such as the widely used infrastructures for web services, mobile devices, and graphical user interaction.

Because of the rapid growth in significance of software capability to the DoD overall, the Director of Defense Research and Engineering (now Assistant Secretary of Defense for Research and Engineering) requested the National Research Council (NRC) Committee for Advancing Software-Intensive Systems Producibility to undertake a study to address the challenges of defense software producibility, identifying the principal challenges and developing recommendations regarding both improvements to practice and priorities for research. The NRC committee just released its final report, titled Critical Code: Software Producibility for Defense. Full copies of the report (free PDF download and book purchase), along with related prior reports, are available through the National Academy Press at http://www.nap.edu/catalog.php?record_id=12979. This article summarizes the principal findings and recommendations of that report.

The necessity of sustaining software innovation

An initial question is whether software is indeed a strategic building material, worthy of special attention. This question has been addressed periodically by the Defense Science Board (DSB) since 1985—a 2007 DSB report, for example, stated that "in the Department of Defense, the transformational effects of information technology (IT—defined here broadly to include all forms of computing and communications), joined with a culture of information sharing, called Net-Centricity, constitute a powerful force multiplier. The DoD has become increasingly dependent for mission-critical functionality upon highly interconnected, globally sourced IT of dramatically varying quality, reliability, and trustworthiness."

Despite the strength of this statement, every few years speculation surfaces that perhaps software and information technology may be approaching a plateau of capability and performance and that strategic attention to these technologies is consequently not merited. The committee emphasizes that this continues to be a false and dangerous speculation—the capability and the complexity of hardware and software systems are both rising at an accelerating rate, with no end in sight.

It is instructive, in this regard, to consider the publication in 1958—more than a half century ago—of the landmark paper by John Backus describing the first Fortran compiler. The title included the words "automatic programming." The point of this phrase, with respect to Backus's great accomplishment, is that there was a much more direct correspondence between his high-level programming notation—the earliest Fortran code—and pure mathematical thinking than had been the case with the early machine-level code. One can construe that it was imagined that Fortran enabled mathematicians to express their thoughts directly to computers, seemingly without the intervention of programmers. The early Fortran was indeed an extraordinary and historical breakthrough. But we know that, in the end, those mathematicians of 50 years ago soon evolved into programmers—as a direct consequence of their growing ambitions for computing applications.

Just a few years after the Backus paper, Fortran was used to support list-processing applications, typesetting applications, compilers for other languages, and other applications whose abstractions required some considerable programming sophistication (and representational gerrymandering) to be represented effectively as early Fortran data structures—arrays and numeric values. Any program that manipulated textual data, for example, needed to encode the text characters, textual strings, and any overarching paragraph and document structure very explicitly into numbers and arrays. A person reading program text would see only numerical and array operations because that was the limit of what could be explicitly expressed in the notation. This meant that programmers needed to keep track, in their heads or in documentation, of the nature of this representational encoding. It also meant that testers and evaluators needed to assess programs through this (hopefully) same layer of interpretation.

As languages have evolved (including more modern Fortran versions), these additional structures can be much more directly expressed—characters and strings, most obviously, are intrinsic in nearly all modern languages. It is interesting, however, that the claim of "automatic programming" continues to reappear from time to time as major steps are made in improved abstractions, for example related to data manipulation (the so-called 4GLs). These developments move us forward, but ironically they do not actually get us closer to "eliminating programmers" or otherwise emerging at some plateau of capability and near-commodity status. Instead, new software-manifest capabilities are constantly emerging—for example, techniques for machine-learning algorithms and highly parallel data-intensive analytics—that continue to demand considerable intellectual effort on the part of programmers.

The profound fact is that software capability is bounded primarily by our intellectual abilities—our human capability both to create new abstractions appropriate for application domains and to manifest those abstractions in languages, models, tools, and practices. As our understanding advances, so can our software capability advance with us.

As a consequence of this seeming unboundedness, the committee finds that *technological leadership in software is a key driver of overall capability leadership in systems*—and that at the core of the ability to achieve integration and maintain mission agility is the ability of the DoD to produce and evolve software. The committee recommends that, to avoid loss of leadership, the DoD take active steps to become more fully engaged in the innovative processes related to software producibility. In particular, the committee finds that industry, despite the extraordinary pace of innovation we are now witnessing, will not produce software innovations in areas of defense significance at a rate fast enough to allow the DoD to fully meet its software-related requirements and remain ahead of potential adversaries.

"...to avoid loss of leadership, the DoD [should] take active steps to become more fully engaged in the innovative processes related to software producibility."

A loss of leadership could threaten the ability of the DoD to manifest world-leading capability, and also to achieve adequate levels of assurance for the diversely sourced software it intends to deploy. This is an important part of the rationale for the committee recommendation that the DoD reengage directly in the innovation processes.

The committee also finds that although the DoD relies fundamentally on mainstream commercial and open source components, supply chains, and software ecosystems, it nonetheless has special needs in its mission systems that are driven by the growing role of software in systems overall. The committee recommends that the DoD regularly **undertake an identification of areas** of technological need where the DoD has "leading demand" and where accelerated progress is needed.

Three goals for software-intensive development

The committee identified three areas where improvements in practice would materially benefit the ability of the DoD to develop, sustain, and assure software-intensive systems of all kinds. Each of these areas is the subject of a chapter in the Critical Code report. (These three areas of practice correspond to Chapters 2, 3, and 4. Chapter 1 of the report focuses on the necessary role of DoD in software innovation. Chapter 5 summarizes the research agenda related to software producibility.) The three areas of practice are summarized below:

Practice improvement 1: Process and measurement

Advances related to process and measurement would facilitate broader and more effective use of incremental iterative development, particularly in the arms-length contracting situations common in DoD.

Incremental development practices enable continuous identification and mitigation of *engineering risks* during a systems development process. Engineering risks pertain to the consequences of particular choices to be made within an engineering process—the risks are high when the outcomes of immediate project commitments are both consequential and difficult to predict. Engineering risks can relate to many different kinds of engineering decisions—most significantly architecture, quality attributes, functional characteristics, and infrastructure choices.

When well managed, incremental practices can enable innovative engineering to be accomplished without a necessarily consequent increase of overall programmatic risk. (*Programmatic risk* relates to the successful completion of engineering projects with respect to expectations and priorities for cost, schedule, capability, quality, and other attributes.) This is because incremental practices enable engineering risks to be identified early and mitigated promptly. Incremental practices are enabled through the use of diverse techniques such as modeling, simulation, prototyping, and other means for early validation—coupled with extensions to earned-value models that measure and give credit for the accumulating body of evidence in support of feasibility. Incremental approaches include iterative approaches, staged acquisition, evidence-based systems engineering, and other methods that explicitly acknowledge engineering risk and its mitigation.

The committee finds that incremental and iterative methods are of fundamental significance to DoD for innovative, software-intensive engineering in the DoD, and they can be managed more effectively through improvements in practices and supporting tools. The committee recommends a diverse set of improvements related to advanced incremental development practice, supporting tools, and earned-value models.

Practice improvement 2: Architecture

Advances related to architecture practice would facilitate the early focus on systems architecture that is essential particularly for systems with demanding requirements related to quality attributes, interlinking, and planned flexibility.

Software architecture models the structures of a system that comprises software components, the externally visible properties of those components, and the relationships among the components. Good architecture entails a minimum of engineering commitment that yields a maximum value. In particular, architecture design is an engineering activity that is separate, for example, from ecosystems certification and other standards-related policy setting.

For complex innovative systems, architecture definition embodies planning for flexibility—defining and encapsulating areas where innovation and change are anticipated. Architecture definition also most strongly influences diverse quality attributes, ranging from availability and performance to security and isolation. Additionally, architecture embodies planning for the interlinking of systems and for product line development, enabling encapsulation of individual innovative elements of a system.

For many innovative systems, therefore, it may be more effective to consider architecture and quality attributes before making specific commitments to functionality. Because architecture includes the earliest and, often, the most important design decisions—those engineering costs that are most difficult to change later—early architectural commitment (and validation) can yield better project outcomes with less programmatic risk.

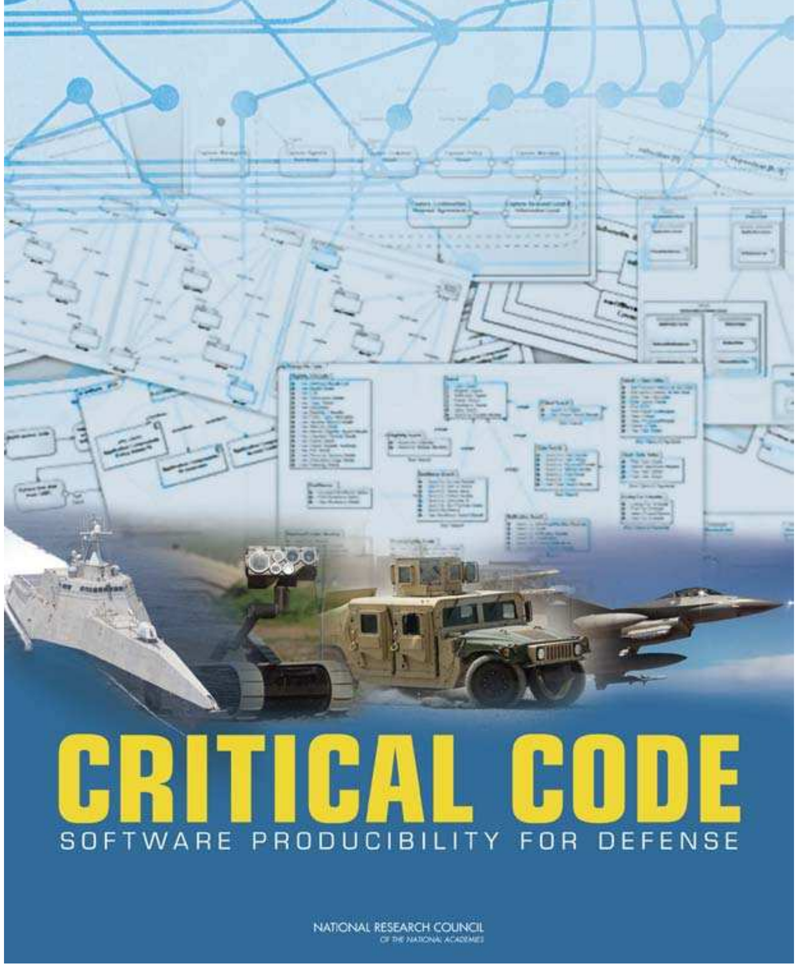
The committee finds that in highly complex systems with emphasis on quality attributes, architecture decisions may dominate functional capability choices in overall significance. The committee also notes that architecture practice in many areas of industry is sufficiently mature for DoD to adopt. The committee recommends that DoD more aggressively assert architectural leadership, with an early focus on architecture being essential for systems with innovative functionality or demanding quality requirements.

Practice improvement 3: Assurance and security

Advances related to assurance and security would facilitate achievement of mission assurance for systems at greater degrees of scale and complexity, and in the presence of rich supply chains and architectural ecosystems that are increasingly commonplace in modern software engineering.

Assurance is a human judgment regarding not just functionality, but also diverse quality attributes related to reliability, security, safety, and other system characteristics. The weights given the various attributes are typically determined on the basis of models of hazards associated with the operational context, including potential threats. The process of achieving software assurance, regardless of sector, is generally recognized to account for approximately half the total development cost for major projects.

In addition to overall cost, DoD faces several particular challenges for assurance. First, there is often an arms-length relationship between a contractor development team and government stakeholders, making it difficult to develop and share the information necessary to making assurance judgments. This can lead to approaches that overly focus on *post hoc* acceptance evaluation, rather than on the emerging practice of "building in" evidence in support of an overall assurance case. Second, modern systems draw on components from diverse sources. This implies that supply-chain and configuration-related attacks must be contemplated, with "attack surfaces" existing within an overall application,


CRITICAL CODE
SOFTWARE PRODUCIBILITY FOR DEFENSE
NATIONAL RESEARCH COUNCIL
OF THE NATIONAL ACADEMIES

and not just at its perimeter. This has the consequence that evaluative and preventive approaches ideally must be integrated throughout a complex supply chain. A particular challenge is managing opaque, or "black box," components in a system—this issue is addressed in the full report. Third, the growing role of DoD software in warfighting, in protection of national assets, and in the safeguarding of human lives creates a diminishing tolerance for faulty assurance judgments. Indeed, the Defense Science Board notes that there are profound risks associated with the increasing reliance on modern software-intensive systems: "this growing dependency is a source of weakness exacerbated by the mounting size, complexity, and interconnectedness of its software programs." Fourth, losing the lead in the ability to evaluate software and to prevent attacks can confer advantage to adversaries with respect to both offense and defense. It can also force us to overly "dumb down" systems, restricting functionality or performance to a level such that assurance judgments can be more readily achieved.

The Defense Science Board found in 2007 that "it is an essential requirement that the United States maintain advanced capability for 'test and evaluation' of IT products. Reputation-based or trust-based credentialing of software ('provenance') needs to be augmented by direct, artifact-focused means to support acceptance evaluation." This is a significant challenge, due to the rapid advance of software technology generally and also the increasing pace by which potential adversaries are advancing their capability. This, coupled with the observations above regarding software innovation, is an important part of the rationale for the committee recommendation that the DoD actively and directly address its software producibility needs.

In the full report, the committee addressed a broad range of issues related to software assurance, including evidence-based approaches, evaluation practices, and security-motivated challenges related to configuration integrity (particularly in the presence of dynamism) and separation (including isolation and sandboxing). The committee notes that traditional approaches based purely on testing and inspection, no matter how extensive, are often insufficiently effective for modern software systems. It emphasizes that evaluation practices that focus primarily on *post hoc* acceptance evaluation are not only very costly but are often insufficient to justify useful assurance judgments. That is, quality and security must be built in, and not "tested in"—with the consequence that evidence production in support of assurance must be integrated into software development.

The committee finds that assurance is facilitated by advances in diverse aspects of software engineering practice and technology, including modeling, analysis, tools and environments, traceability and configuration management, programming languages, and process support. The committee also finds that, after many years of slow progress, recent advances have enabled more rapid improvement in assurance-related techniques and tools. This is already evident in the most advanced commercial development practice. The committee also finds that simultaneous creation of assurance-related evidence with ongoing development has high potential to improve the overall assurance of systems. The committee recommends enhancing incentives for preventive software assurance practices and production of assurance-related evidence throughout the software lifecycle and through the software supply chain. This includes both contractor and in-house development efforts.

The challenge of DoD software expertise

The committee also took up the issue of software expertise that is specifically aligned with DoD interests. The committee found that DoD has a growing need for software expertise, but that it is not able to meet this need through intrinsic resources. This need is essential for the DoD to be a smart software customer and program manager, particularly for larger-scale innovative software-intensive projects. In particular, access to DoD-aligned expertise is important for the DoD to be able to take effective action in the three areas of practice that are identified above. Access to DoD-aligned expertise has been an area of ongoing challenge to the DoD, with recommendations made by various panels and committees since the 1980s.

The need to reinvigorate DoD software engineering research

In addition to recommending improvements to the three areas of practice, as outlined above, the committee identified seven areas of supporting research for consideration by science and technology program managers (managing 6.1, 6.2, and 6.3a funds and equivalent). These areas are identified on the basis of four criteria: (1) Advances would yield significant potential value for DoD software producibility. (2) A well-managed research program would result in feasible progress. (3) The goals are not addressed sufficiently by other federal agencies. (4) The pace of development in industry or research labs would be otherwise insufficient.

In each of the seven areas, the committee identified specific goals for research and technology development that, in its judgment, could feasibly meet the four criteria. The areas and, for each, the identified goals are summarized below. (Details are in the full report.)

1. Architecture modeling and architectural analysis. Goals include: (1) Early validation for architecture decisions; (2) Architecture-aware systems management, including: Rich supply chains, ecosystems, and infrastructure; (3) Component-based development, including architectural designs for particular domains

2. **Validation, verification, and analysis** of design and code. Goals include: (1) Effective evaluation for critical quality attributes; (2) Components in large heterogeneous systems; (3) Preventive methods to achieve assurance, including process improvement, architectural building blocks, programming languages, coding practice, etc.

3. Process support and economic models for assurance. Goals include: (1) Enhanced process support for assured software development, (2) Models for evidence production in software supply chains, (3) Application of economic principles to process decision-making

4. Requirements. Goals include: (1) Expressive models, supporting tools for functional and quality attributes; (2) Improved support for traceability and early validation

5. Language, modeling, coding, and tools. Goals include: (1) Expressive programming languages for emerging challenges, (2) Exploit modern concurrency: shared-memory and scalable distributed, (3) Developer productivity for new development and evolution

6. Cyber-physical systems. Goals include: (1) New conventional architectures for control systems, (2) Improved architectures for embedded applications

7. Human-system interaction. Goals include: (1) Engineering practices for systems in which humans play critical roles. (*This area is elaborated in a separate NRC report.*)

Under the auspices of the Office of Science and Technology Policy (OSTP) and the National Science and Technology Council (NSTC), there is a National Coordination Office for the Networking and Information Technology Research and Development (NITRD) program. The NITRD program provides a framework for diverse federal agencies to coordinate R&D in areas related to networking and information technology. The framework includes two areas that primarily relate to software producibility, which are Software Design and Productivity (SDP) and High Confidence Software and Systems (HCSS). There is also a third area, Cyber Security and Information Assurance (CSIA) that encompasses some activities related to software producibility.

The committee undertook a longitudinal study of sponsored R&D budgets as identified in NITRD reports, with specific focus on SDP and HCSS. It found that while NITRD overall has grown over the past decade, there has been a significant reduction in both overall and DoD-sponsored R&D in SDP and HCSS. The committee recommends that DoD take immediate action to reinvigorate its investment in software producibility research, with focus in **the seven identified areas.**