

# SOFTWARE: THE ENDLESS VALUE SPIRAL<sup>1</sup>

William L Scherlis

Carnegie Mellon University School of Computer Science

Software plays a critical role in modern global enterprises, both in manifesting new kinds of value and in increasing productivity. This is true not just in the so-called software sector—the familiar software-intensive firms such as Oracle, IBM, Microsoft, Google, and SAP, as well as Amazon, Yahoo!, and eBay—but also in many other economic sectors including national security, health care, utilities, financial services, communication, and transportation. It is fair to say that software has become the building material of choice for nearly all kinds of complex systems.

Yet despite this pervasiveness of adoption, the engineering of software has not matured into a rigorous discipline. The risks and difficulties of software are growing in severity and diversity, and we continue to experience failures of all kinds—related to reliability, security, flexibility, and other attributes.<sup>2</sup> Software-related problems are responsible for life-threatening failures in health devices, failures of space missions, failures in military systems, cascading failures in infrastructure for telecommunications and power utilities, and so on. Additionally, there are numerous failures in the process of developing and operating innovative software.

The pace of innovation shows no signs of slowing, and the extraordinary economic contribution of software continues to increase. An analysis of European economic data, for example, suggests that while the information and communications technology (ICT) sector represents just over 5 percent of the European GDP, ICT drives 25 percent of overall growth and about 40 percent of the increase in productivity.<sup>3</sup> Software is also fundamental to national security; for example, an analysis by the U.S. Defense Science Board states that in modern fighter aircraft, the percentage of total systems functions that is enabled by software has risen to 80 percent.<sup>4</sup>

---

<sup>1</sup> This is a revised and augmented version of a paper in *Renewing Globalization and Economic Growth in a Post-Crisis World The Future of the G-20 Agenda*, online at

[http://www.cmu.edu/universitypress/data/renewing\\_globalization\\_growth\\_g20.pdf](http://www.cmu.edu/universitypress/data/renewing_globalization_growth_g20.pdf)

<sup>2</sup> The Risks Digest has provided an account of such failures for more than 20 years. Current and past issues of the Digest are at <http://catless.ncl.ac.uk/Risks/>.

<sup>3</sup> S. McGibbon, *Growth and Jobs from the European Software Industry*, European Review of Political Technologies, December 2005.

<sup>4</sup> *Report of the Defense Science Board Task Force on Defense Software*, Office of the Under Secretary of Defense for Acquisition and Technology, Washington DC, November 2000.

## *Immature Discipline*

It may be perplexing that software-related risks seem to be increasing dramatically just when software technologies and practices are undergoing rapid improvement. This is unlike many traditional and established engineering disciplines. We suggest, in fact, that we are unlikely any time soon to arrive at what is generally accepted as repeatable engineering practice for innovative projects. This is a consequence of the boundlessness of software—software is a uniquely abstract and purely synthetic medium that, for the most part, lacks fundamental physical limits and the natural constraints that provide favorable conditions for establishment of the conventions and patterns that characterize repeatable practice.

This does not mean we should somehow avoid or limit the use of innovative software—indeed, it is foolish not to continue advancing our use of software and improving our practices. But we must learn to engage with a more strategic attitude, focused both on the microcosm of how we manage software projects and also on the macrocosm of why and how we undertake software innovation. In the microcosm of software projects, we see that each new technical advance both creates opportunities but also presents new difficulties of measurement and risk assessment. With respect to software innovation, we see that leadership in software capability and innovation has a leveraged impact, conferring advantage across multiple sectors. This is why software innovation must receive constant priority in industry and government.

## *Software in Organizations*

Let us consider the changing role of software in organizations in the past decade or so. There are three principal elements of this shift. First, most organizations have moved quite aggressively from managing portfolios of functionally focused systems to interconnecting systems within and across enterprises. This enables more agile and informed business models, including well established capabilities in ERP, CRM, SCM (Enterprise Resource Planning, Customer Relationship Management, and Supply Chain Management), and the like. In military systems, we see an analogous set of ideas in net-centric warfare and other modern systems concepts. This interconnection has associated risks, primarily related to the magnitude of failures, most vividly evident in the cascading failures of interconnected systems as recently experienced, for example, in telecommunications, utilities, and in supply chain systems.

Second, largely as a consequence, IT staffs are generally less involved in mediating between a system and its users—both those within an organization and those outside, including individuals. This is more efficient because it better couples decisions with actions. But it also means that many more individuals—usually inadvertently, but not always—can take actions with wide-reaching consequences, both positive and negative. Third, the systems support immediate electronic enactment of decisions. This enables agility and fast response in decisions and actions (consider the current discussion over the duration in milliseconds of the stock trading look-ahead window), but it also means that failures and compromises can happen very quickly, inside a human decision loop.

## *Software Supply Chains*

These are profound shifts, and they are enabled in part by a surprisingly recent phenomenon in IT, which is the enrichment of the component structure and, consequently, the supply-chain structure for IT systems. This enrichment is more than just outsourcing as experienced in the past half century. Supply chains now include many more players, and involve

technically rich and complex architectures, with frameworks, libraries, services, and other roles. The value of outsourcing, which was initially primarily cost reduction and access to expertise, now includes greater agility and ability to respond to changes in the operating environment. This richness in the supply chain is enabled by technical advances in areas ranging from the design of software architectures and components to the technical properties of modern programming languages and the design of modern processor architectures.

These technical steps have permitted a better alignment of business structures and IT structures in organizations, in addition to allowing greater flexibility in buy-versus-build decisions. For example, a goal of many organizations is to outsource common infrastructure such as databases, application servers, and software frameworks, and to in-source only those critical elements that provide unique capabilities and advantage over competitors. As technologies evolve and “commoditize,” it may be perceived that the trend is to shift function from in-source to outsource. But this is not always the case, as new dimensions of capability and differentiation emerge for formerly commoditized infrastructural elements, as is happening now for data centers and their architectures.

### *Immature Discipline, Revisited*

The changes sketched above may seem to contribute to the maturing of software as an engineering discipline, and in one sense this is very much the case. But there are also key characteristics of software that cause measurable and predictable outcomes—a hallmark of maturity—to remain disappointingly elusive. The principal characteristic is that advances in the underlying technology and practice of software are continuing at a rate that has been unabated for decades, in a manner roughly analogous to Moore’s Law.

In other words, software is not at a plateau, despite apparent declarations of victory made on a regular basis. One of the first examples was the publication more than fifty years ago of the 1958 landmark paper by John Backus describing the first Fortran compiler. The title referenced “automatic programming.”<sup>5</sup> The point of this phrase is that there was a much more direct correspondence between the programming notation—the earliest Fortran code—and pure mathematical thinking, than had been the case with the early machine-level code. With Fortran, mathematicians could express their thoughts directly to computers seemingly without the intervention of programmers. This was an extraordinary and historical breakthrough. But we know that, in the end, those mathematicians soon evolved into programmers as their ambitions for computing applications increased to the next level. The same story can be told about claims sometimes associated with the so-called “Fourth Generation” database languages a few decades later and, more recently, with languages for business rules. These are all major innovations, but it is fair to say that the specialness of software is that these developments move us forward but do not actually get us closer to “being there” with a fully established mature discipline. New software-manifest capabilities are constantly emerging—for example, machine learning technology is now used in applications ranging from data mining to robot design and quality-of-life enhancement for seniors. The profound fact is that software seems to be limitless.

Consider, for example, the traditional “stacks” of infrastructure and capability on which systems are built—technical progress has enabled us to advance our infrastructure capability,

---

<sup>5</sup> J.W. Backus, *Automatic programming: properties and performance of FORTRAN systems I and II*. In Proc. Symp. on the Mechanisation of Thought Processes. Teddington, Middlesex, England: The National Physical Laboratory, 1958.

commoditizing and increasingly automating greater levels of capability in operating systems, databases, application servers, frameworks of various kinds, data centers, and so on. But, at the same time, we continue to innovate above and around the infrastructure, creating new kinds of capability and differentiation. Additionally, we continue to innovate within the infrastructure to add capability or make other enhancements. In other words, while there is a seemingly inevitable commoditization of software component capabilities, there is also a seemingly indefinite deferment of reaching the goal of safe decision making regarding innovative software-manifest capabilities in systems. This is a key point about the intrinsic lack of limits of software, and indeed this is a principal characterizing feature of software as an engineering building material.

### *Conclusions*

We draw several conclusions from these observations. First, mere *presence* in the market—just being a software user—requires keeping pace with ongoing software innovation and improvements to practices. This is true even for individual software components—software starts to die the moment it stops evolving. It is also true for practices—continuous improvement in practices and processes is essential for survival. Second, *leadership* in the market requires, additionally, an active organizational role in defining the architecture of systems, and doing so as a first mover. Software economics are focused on externalities—where the technical manifestation of the *system* structure is the software architecture and internal framework and component interfaces. This requires sustained technological leadership and clear thinking about the significance of architectural control. Third, software *technical challenges* are broadening. These include, for example, software assurance, ultra-scale architecture, concurrency (multi-core and distributed), framework design, programming language improvements for assurance and scale, concepts for “big data” systems, and so on. Fourth, *risk management* models need to be continually adjusted to accommodate the new realities of software and of IT-enabled business practices, as noted above.<sup>6</sup> Finally, the role of *software leadership* in the global economy is growing, and this is increasingly recognized, with the result that competition is becoming more intense at every level of capability. Such leadership must be maintained through constant investment in innovation and in people, both. While costly physical facilities are not needed in the software economy, education and technical currency are fundamental and ongoing challenges.

Software has diverse and critical roles, and a disproportionate influence on value creation in many economic sectors. Technical progress is rapid, and there seems no real limit in sight. The software ecosystem is constantly evolving as architectures shift and commoditization comes (and goes). But the risks are significant. Software users must keep pace with the technology, and they must constantly rethink how to balance risks and benefits from both an engineering perspective and a business or mission perspective. Software leaders must do all that the users do, and, in addition, continuously invest in sustaining software capability at a level where they can maintain technological and therefore architectural leadership.

---

<sup>6</sup> See J. Rosenoer and W. Scherlis, *Risks Gone Wild*, Harvard Business Review, May 2009