Did We Lose Our Religion?

Lloyd K. Mosemann II
Science Applications International Corporation

This article was presented as a keynote address at the Software Technology Conference 2002 in Salt Lake City in May and has been edited for length and clarity. The author takes the government to task for using religious-like belief systems rather than objective appraisals to build and buy software. The author says that best commercial practices do not exist in government contractors as they do in the real commercial world: in-house software expertise, a robust software development environment, and sound software architecture. While he is not suggesting that the government develops software in-house, the author does suggest that it needs enough in-house software expertise to know what it is buying.

In 1990, I declared that the 1980s were a lost decade from the perspective of software development progress. The question I posed was, "Will there be a breakthrough in the 1990's?" I went on to say, "It won't happen automatically; people are too satisfied with unsatisfactory ways. We dare not make the mistake of complacency a la the automobile industry; we must push awareness and resource commitment to get ahead of the power curve of demand."

In 1994, I closed the annual Software Technology Conference (STC) with the observation that the underlying need within the defense community is for *predictability*: "From a Pentagon perspective, it is not the fact that software costs are growing annually and consuming more and more of our defense dollars that worries us. Nor is it the fact that our weapons systems and commanders are becoming more and more reliant on software to perform their mission. Our inability to predict how much a software system will cost, when it will be operational, and whether or not it will satisfy user requirements is the major concern. What our senior managers and DoD (Department of Defense) leaders want most from us is to deliver on our promises. They want systems that are on time, within budget, that satisfy user requirements, and are reliable."

The question I pose now is: "Where are we today, and where will we be tomorrow?" Did we lose our religion?

Why did I use the metaphor of *religion*? Because religion is the traditional example of faith-based behavior – that is, behavior that is based on a belief system rather than on externally imposed rules such as the law of gravity or "she that has the gold, rules." Emotional discussions regarding whether Ada or C++ should be preferred are frequently described as *religious* arguments based on beliefs rather than facts.

Sadly, I still see the world of software being governed by religious-like belief systems rather than objective appraisals. When I left the Pentagon six years ago, I described some of what was happening as *bumper sticker* management, and the situation has not changed for the better. I sometimes have the feeling that the blind are leading the blind – the leadership is blissfully ignorant of the direction in which they are headed.

The only meaningful direction from either the Office of the Secretary of Defense (OSD) or the military services in the last few years was the Gansler memo that directed the use of the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) Level 3 contractor organizations for Acquisition Category (ACAT) 1 systems. Do you know how many large-dollar (by that I mean $50 million or more) software intensive acquisitions are not ACAT 1? Virtually all Management Information System (MIS) and Command, Control, and Communications (C3) systems!

During the past two years, there has been a 5.5 percent annual growth in the cost of ACAT 1 programs due to cost and schedule estimating and engineering changes (sound like software?). Yet these programs have the most experienced DoD industry managers, and have a requirement for CMM Level 3. About two-thirds of DoD acquisition dollars are for programs below the ACAT 1 threshold for which there is currently no CMM requirement. It is my guess that these non-ACAT 1 programs are at least twice as bad as ACAT 1 programs – in other words, about $9 billion per year in cost growth associated with estimating and engineering problems, many of which are likely software related. In my opinion, they deserve more software management attention than results from the requirement to use best commercial practice.

CMM Maturity Reality

What is wrong with best commercial practice? It just does not exist among DoD contractors. It is a fantasy created by those who want to streamline acquisition, making it possible to cut the number of oversight personnel by reducing the opportunity for oversight. The best way to justify a hands-off attitude is to insist that contractors always do everything right!

There are more mature software organizations today. Virtually every large DoD contractor can boast at least one organization at CMM Level 4 or above, and several organizations at CMM Level 3. On the other hand, most DoD software is still being developed in less mature organizations – mainly because the program executive officer or program manager (PM) doesn't demand that the part of the company that will actually build the software be Level 3!

Many people used to tell me that the DoD needed to get on the dot-com bandwagon – those folks develop and deliver software fast. Yes, the Internet and the availability of Web browsers have fundamentally changed the environment within which even mission critical software is being developed. But instead of adapting proven software methods, the software research community has all but dropped its concerns with formal methods, peer reviews, clean-room processes, and other reliability techniques, including Ada, which was designed to promote reliability. Except for Barry Boehm at the University of Southern California, much of the academic community has more or less stopped investigating better ways of estimating system complexity and measuring software growth. Instead, invention of new user interfaces, new distributed computing architectures, and new (more flexible and less reliable) programming languages have been given top priority. The goals of reliable performance and pre-

dictable development costs have been largely ignored.

Wayt Gibbs, author of the 1994 *Scientific American* article, "Software's Chronic Crisis" told me: "It is tempting to argue that the lack of a disciplined approach to software development is the principal reason that so many dot-com ventures failed – and consumed such flabbergasting amounts of money as they failed."

That may be stretching it. Addle-brained business models clearly played a starring role as well. But it is interesting that, with very few exceptions, the dot-com startups embraced the programmer-as-master-craftsman model of development. That very old-fashioned model is still considered avant-garde in the open source movement. How many software startups in the past eight years have submitted to SEI evaluation or tried to achieve CMM Level 3? You won't need many fingers to count them all.

In addition to abandonment of formal methods, the Internet has also made obsolete the fortress model of security and the notion that an astute administrator can enforce central control of all components of a system. It is no longer realistic to dream of a mathematical certainty that each software component in a system is correct. In a world where network communication and user interface standards depend on change every 12-18 months, it does not make sense to lock down detailed requirements, spend a year proving them consistent, then spend two years building to spec.

A Move to Self-Sustaining Systems

There is a new move toward something called *autonomic computing*. This is a vague term that encompasses several lines of research aimed at taming complexity to achieve reliability. The approach is not *formal* but *organic*, i.e., find ways to build systems that can heal, that can achieve their mission despite bugs, equipment failures, and even sabotage. In other words, design systems that constantly monitor their own performance, that notice failures, can perform self-maintenance, and do not *crash* but degrade gradually. Some folks say these goals can be achieved in 10 to 20 years, but do not bet your paycheck on it.

Ironically, the problem facing much of industry is exactly the opposite. Wayt Gibbs told me the following: "Executives who woke up one day in 1997 to discover a time bomb in the form of Y2K bugs ticking inside their systems were forced to take a substantial hit to their bottom line. But an ironic consequence of Y2K is that many companies upgraded their affected systems in 1999 to work with or through the Internet and various fly-by-night standards and startup-built *solutions* that are now as obsolete as Algol. As a result, their systems are again ticking time bombs. Though they will not all fail at the same time, their failure will be just as unpredictable and much harder to fix."

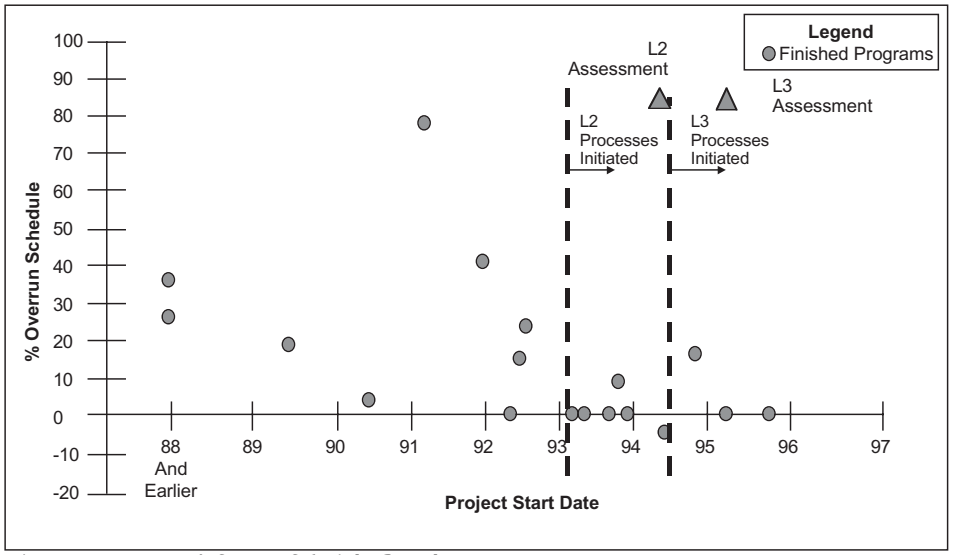


Figure 1: *Impact of SPI on Schedule Compliance*

The Missing Agenda

Some subjects have been notably missing from the plenary sessions of conferences during the past few years: software engineering, product line development, formal methods programming, and predictability.

In 1991, Paul Strassmann, then-director of Defense Information, said: "The No. 1 priority of the DoD, as I see it, is to convert its software technology capability from a cottage industry into a modern industrial method of production."

Guess what? That has not happened. Why not? Because this requires software engineering, which encompasses a set of three key elements – methods, tools, and procedures – that enable the manager to control the process of software development and provide the practitioner with a foundation for building high quality software in a productive manner.

The fundamental ingredient in a software engineering approach is the design of robust software architecture. Architecture does not refer to grouping and linkage of servers, routers, and PCs, but rather to the inherent design of the software itself – the identity of modules and their relationships, including the infrastructure, control, and data interfaces that permit software components to operate as a system.

I was told by an attendee at a DoD Systems Design Review several months ago that a contractor had described his proposed system as modular. That is a good architectural term, and it was accepted at face value. In fact the system, when looked at by the independent attendee, only had two modules. When this was brought to the government program manager's attention he said, "The contractor says it is modular. He's smarter than we are." This little incident underscores two facts: architecture was understood by neither the contractor nor by the government PM, and the probability of a successful software delivery is low.

All too often the DoD excuse for not requiring an architectural evaluation is that "requirements change rapidly – we can't afford to be locked into a specific architecture." Wrong. That is the very reason that attention should be given to architecture, so that changes to requirements can be accommodated easily.

I am told that SEI is still being asked to do Independent Technical Assessments of why a software acquisition has not produced the desired working software system. Why wait to ask SEI to come in to do a post-mortem and tell you how you screwed up? Why not ask them to come in first and review the Request for Proposal (RFP) and Statement of Work and, second, assist in evaluating the software engineering practices, software development environment, and architecture proposed in response to the RFP, and then afterwards assess the quality of the software engineering and development process? SEI is not cheap, but terminating a $100-million project for lack of satisfactory performance is not cheap either.

Interestingly, when one thinks about best commercial practice, there are two very different worlds out there. There is the government contractor world and there is the real commercial world – banks,

insurance companies, UPS, FedEx, Eckerd Drug, and Disney World. These companies developed their own software using the best available tools like Rational's software development environment. They did not pick the cheapest tools. They did not rely on commercial-off-the-shelf (COTS) or outside software developers – their software is their business. They consider that it provides them a competitive advantage. They want to control it, and they use the best tools available regardless of cost.

Architecture-Centered Approaches

Product line developments are also becoming increasingly commonplace in the true commercial world. The Swedish firm CelsiusTech was the first to exploit the benefits of a product line architecture approach to software application development back in the late 1980s. There are now a number of well-known firms who are using an architecture-centered approach: Nokia, Motorola, Hewlett-Packard, Philips, Cummins Engine, and (believe it or not) one government application at the National Reconnaissance Office (NRO).

The NRO is enjoying a 50 percent reduction in overall cost and schedule, and nearly tenfold reductions in defects and development personnel. Let me list the most common and relevant-to-DoD reasons that these companies give for adopting the architecture-centered product line approach to software development:

- Large-scale productivity gains.
- Improved time-to-market = field weapons faster.
- Improved product quality.
- Increased customer satisfaction = warfighter satisfaction.
- Enabled mass customization.
- Compensated for inability to hire software engineers.

These companies and the NRO do have *best practices* but are not yet widely recognized as such. Frankly, it will take DoD program executive officers (PEOs) (not program managers) and their overseers, the service acquisition executives, and especially the DoD comptroller and program analysis and evaluation folks, to recognize and direct the product line approach to software development and acquisition. (Unfortunately, these folks are not known for their software acumen.)

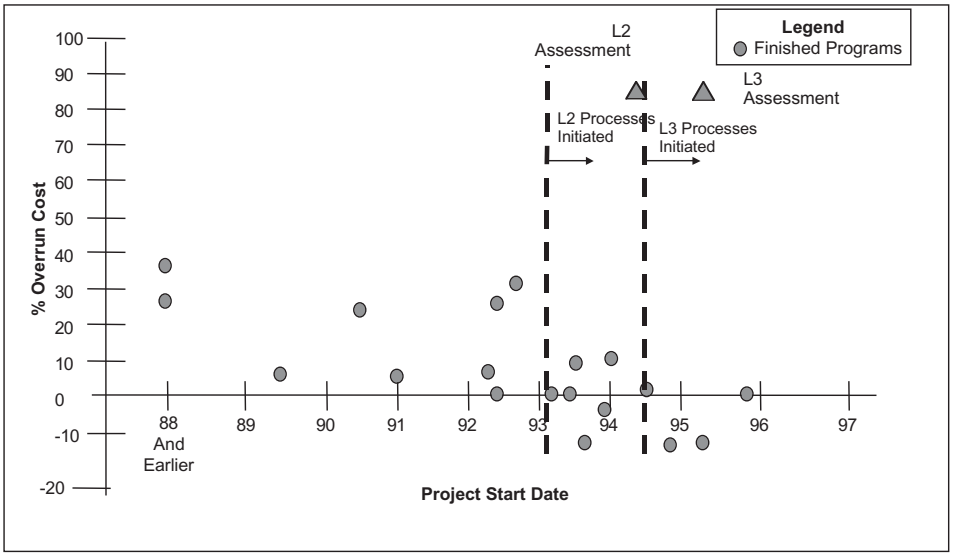
The major impediment to the product line development approach, aside from ignorance of its benefits, are cultural, organizational, and, especially, the DoD's stovepipe approach to budgeting and funding. The DoD has many potential software product lines. None of them have been built largely for *political* and stovepipe budgeting reasons. As a result, development and fielding of defense systems continues to take longer, cost more, and lack predictable quality. Product lines require strategic investment, which appears to be outside the DoD comptroller and acquisition communities' frames of reference. Yet, it is the DoD that most often uses the term *strategic*.

Cummins' Engine Company used to take a year or more to bring software for a new engine to the test lab – now they can do it in less than a week! I strongly recommend that readers obtain a copy of a new book, "Software Product Lines," just published by Addison-Wesley. The authors are from SEI. Sadly, with the exception of the NRO, it appears that the readers are mainly from commercial organizations.

I work for one, but let me tell you, although government contractors are commercial organizations, they do not have an identifiable best commercial practice. They basically provide what the government asks for. The only reason many of them can now boast of having at least a few SEI maturity level organizations is because 10 years ago, many government RFPs required a Software Capability Evaluation (SCE) as a condition of bidding. In fact, sad to say, many contractors today put satisfactory CMM credentials in their proposals but then perform the work with an organization that could not spell CMM.

Why does the government let this happen? Why aren't there SCEs anymore? Do they take too long and cost too much? Is it better to make a quick award, and then, down the line, accept an inferior product or terminate for convenience? Too often what the government has been asking for is COTS. How many failures of COTS-based acquisitions have there been over the past decade? Too many!

Figure 2: *Impact of SEI on Cost Compliance*



What is Best Commercial Practice?

Best commercial practice is not eliminating all software smarts in government and relying 100 percent on contractors to deliver good software. Best commercial practice is what real commercial companies are doing. They have in-house software expertise, they use a robust software development environment, and they base their software development on sound software architecture. It is no secret that Rational and their competitors have a growing market in the commercial world and a shrinking market in the government. I am not suggesting that the government can or should develop software in-house. I am strongly suggesting that the government needs enough in-house software expertise to know what it is buying. It is still true that you "get what you pay for."

Watts Humphrey recently published "Winning with Software - An Executive Strategy." This book is directed primarily at executives of commercial companies. But every DoD acquisition executive, PEO and PM needs to read and understand its simple message: Software projects rarely fail for technical reasons; invariably, the problem is poor management.

Watts poses two interesting questions buttressed by numerous examples: "Why do competent software professionals agree to dates when they have no idea how to meet them?" "Why do executives accept schedule commitments when the engineers offer no evidence that they can

meet these commitments?" He asserts that management's undisciplined approach to commitment contributes to every one of the five most common causes of project failure:

- Unrealistic schedules.
- Inappropriate staffing.
- Changing requirements.
- Poor quality work.
- Believing in magic.

What is Formal Methods Programming (FMP)? Basically, FMP is all of the above rolled together: sound management, established engineering processes, robust software development environment, model-based architecture, and a reliable programming language. Peter Amey of Praxis Critical Systems said in March 2002 CROSSTALK: "There is now compelling evidence that development methods that focus on bug prevention rather than bug detection can raise quality and save time and money." He went on to say that a key ingredient is the use of unambiguous programming languages that allow rigorous analysis early in the development process.

I was an early and vocal advocate of Ada, primarily because, unlike other languages, it enforces basic software engineering practice. Amey's article describes the use of a subset of Ada known as SPARK, which he says requires software writers to think carefully and express themselves clearly; otherwise, lack of precision is exposed by SPARK Examiner. He said there were significant savings in using SPARK in a critical avionics program, including an 80 percent reduction in formal test costs. Unfortunately, this is an isolated DoD example.

Finally, let me say a word about predictability. Predictability is the only metric that warfighters care about. The question is, how can we make the warfighters (including the PEOs and PMs charged with delivering the needed capabilities) know that you cannot just buy software as a product off the showroom floor? There must be an understanding of the software-engineering paradigm. More than this, to be assured of getting software that works on a predictable schedule and at predictable cost requires that someone in government enunciate requirements. Or else, competitors will lowball price and set an unrealistically fast schedule and be awarded the contract. To perform at low cost means no robust software development environment, no time and effort devoted to creating a valid software architecture, and probably means cheap people. Sufficient process guidance must be given to assure that contractors all bid at the same capability level. Government should be explicit about its need for architecture, a robust software development environment, and perhaps even the requirement for a language like SPARK. At a minimum, it needs to specify at least CMM Level 3. As the figures illustrate, (see pages 23, 24) at Level 1 virtually all projects have cost and schedule overruns, whereas at Level 3 virtually all projects are on target. Regarding defect rates and cost ($) per source line of code, there is substantial improvement on the order of 20 percent to 50 percent. It really is true that "you get what you pay for." If you want it cheap, you'll get it cheap – but the software may not work in the manner envisioned, if it will work at all.

As for CMMI, I believe it is very important for the embedded world… but it would be a gross mistake to discontinue support of the Software CMM.

Are there best commercial practices? You bet! The banks, insurance companies and other truly commercial enterprises have them. Not because of some automatic external happenstance, but because their senior managers have had the moxie to realize that it takes money to make money, and that it takes software expertise to develop or acquire software. The government needs to *go and do likewise*. Otherwise, the decade of 2000 will likely not show any lessening of the software crisis that has carried over from the 1990s.◆

About the Author

Lloyd K. Mosemann II, is the senior vice president of Corporate Development for Science Applications International Corporation (SAIC). Formerly, for almost 25 years, Mosemann was deputy assistant secretary of the Air Force for Communications, Computers, and Logistics. During this time, he chartered and guided the Air Force's Software Technology Support Center and sponsored its annual Software Technology Conference. Prior to that, Mosemann spent 11 years with the Navy. He has a bachelor's and master's degree from the University of Chicago, and has received two Presidential Meritorious Rank Awards, five Air Force Exceptional Service Medals, among other awards.

Science Applications International Corp.
E-mail: lloyd.k.mosemann.II@saic.com