Agile Software Development for an Agile Force

John S. Willison
U.S. Army CECOM Software Engineering Center

I remember in a meeting I attended, an Army general arguing the need for the software community to fall in-line, saying that "Software itself has never killed anyone." Maybe, maybe not, but I have seen software kill many Army programs and careers. As the Army transforms itself into a more agile force, the Army software community continues to struggle with the challenge of effectively providing software to support that force. This article identifies some components of an effective approach to software development and provides an example that is leading the way.

It is necessary to provide a characterization of the current U.S. Army business environment to set the context for the recommended components for good business. Historically, the Army acquisition and development processes have been driven by the attempt to institutionalize success and avoid failure. The Army management and acquisition processes are based primarily on hardware models that, in turn, are based on the value-added discipline of risk management.

With hardware, it is critical to mitigate risk and get it right the first time, particularly prior to entering any stage that involves significant expense such as production. The Army has evolved into using a rigid approach where requirements are defined and then used as the basis for development, testing, and determining success. Further, as development and hardware sustainment are different activities, the Army has defined different processes and funding strategies for these distinct activities.

With software, the processes and investment strategies are different than hardware; the risks are also different, and yet we attempt to manage them the same way. Software sustainment to a large degree is simply doing more development; however, development and sustainment are often managed by different organizations and funded differently. The risks associated with software are different as well; and yet, we attempt to manage them the same as we work through a sequential series of milestones. The real risks with software are in taking too long before giving the user something that knowingly will evolve over time, and in measuring success as meeting predefined requirements as opposed to getting the user something he or she wants and likes.

There are other factors that influence the way the Army acquires and develops software. For the increasing percentage of Army capabilities that are hosted on commercial off-the-shelf (COTS) hardware platforms, competition to provide software is expanding and the barrier to enter the competition is low. Users have access to a wide range of sources, and more importantly, a wide range of sources have access to users. Increasingly, initial and incremental capabilities can be provided to users as software-only releases, and in some cases simply can be downloaded over the network.

Finally, there is this question: how much alike or different should the Army software community be from the commercial software industry? Clearly there are some differences. Most notably, the Army software community's priority is capability and readiness whereas the commercial software industry's priority is profit. Those different priorities have historically been used to rationalize the need for unique and rigid approaches to software.

Components for Good Business

The Army develops, integrates, and employs as wide a range of software-based capabilities as any other organization. While no single method for improving the Army's approach to software development would suffice, there are some common components for improvement.

Balance Between Plan-Driven and Agile Development

The October 2002 edition of CROSSTALK [1] did an excellent job of contrasting the *plan-driven* [2, 3] and agile development approaches to software, and the spectrum between these two perceived extremes. There is much to be gained from both approaches.

The Software Engineering Institute's Capability Maturity Model®, for example, has done much to address software as an engineering discipline and the need for a plan-driven approach. Agile development, as characterized by the Agile Alliance [4], finds:

> … more value in individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

Historically, the Department of Defense and the Army have emphasized process. To produce a more agile force, the Army needs a software community that has process discipline, but is more agile as well.

Get as Close to the User as Possible

The only one who truly knows what the user wants or needs is the user himself. The closer you get to the user, the closer you will get to developing software that he or she will accept and adopt; it is never too early to do this.

Show Them, Ask Them, and Repeat Often

The Army is very good at generating requirements, and generating endless cycles of life-cycle events aimed at meeting those requirements. Instead of asking users what they need and then getting back to them only after developing the solution, the Army should be prepared to show users what they could get up-front. If nothing else, this builds the users' confidence that the Army is able to deliver something. The focus should be on early and continuous software delivery.

Architecture, Architecture, Architecture

Architect Frank Lloyd Wright is believed to have said that no matter what you are building, always remember:

> It will take longer than you plan, it will cost more than you figured, and it will be messier than you could have ever have anticipated. But remember the most important thing is not what is visible. What's

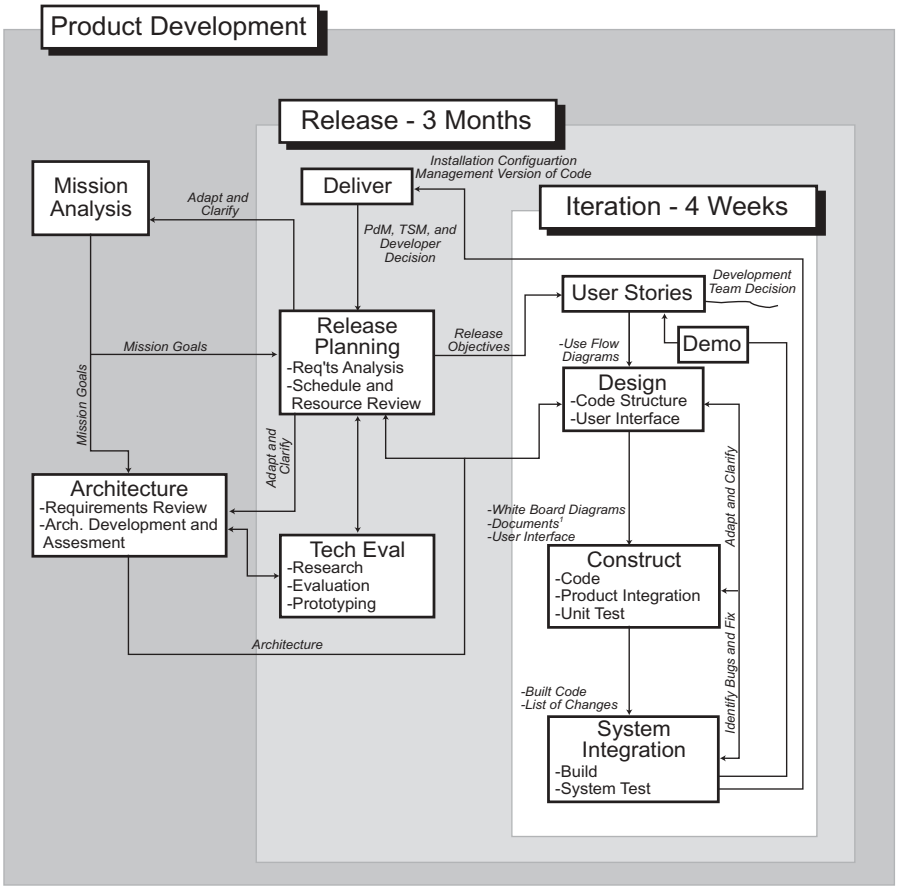
most important is the foundation.

Bad software products do not necessarily have bad software architectures. However, good software products are likely to have good software architectures. While the Army, and everyone else for that matter, has increased the amount of attention and discussion surrounding software architectures, the understanding and practices associated with software architectures are still not sufficient.

Senior Army leaders echoed the need for the Army's development approach to be more agile and responsive at a recent Association of the United States Army Symposium [6]. Lt. Gen. John Caldwell, military deputy to the assistant secretary of the Army (ASA) for Acquisition, Logistics, and Technology (ALT) said, "If you wait to put a perfect capability in the field, you will never put anything in the field," and, "People are the key to our business."

Col. Bruce Jette, director of the recently established Rapid Equipping Force, said that the goal is to go from "idea to equipping" in two to three months and that, "field commanders are tremendously accepting" of this approach.

Col. Nick Justice, director Future Force ASA ALT said, "The best way to find out how to engineer solutions is to get out with the guy who uses them." These principles are captured in the components for good business.



Note: [1]Documents are created for critical, high-risk, unclear, complicated tasks, and external interfaces.

Figure 1: *The Defined Process*

Maneuver Control System Light Program

The feasibility and benefits of applying the components for good business can best be illustrated by way of example. The Army's Maneuver Control System (MCS) program is part of the Army Battle Command System and provides the commander with the capability to plan and monitor the battle. The MCS program is managed by the product manager (PdM) MCS, under the program manager Ground Combat Command and Control and Program Executive Officer Command, Control, Communications – Tactical. Development is led by the Air Mobility Command Communications-Electronics Command (CECOM) Software Engineering Center and supported by Shonborn-Becker Systems Inc., L3 Ilex, Lockheed Martin, CECOM Research Development and Engineering Center, and others.

MCS Light was born out of opportunity and necessity. The MCS Light product implements command-and-control functionality on a PC/Notebook/Windows platform. The MCS Light product has gained widespread acceptance within the Army command-and-control user community. Representative of the success of the product are comments made by Lt. Gen. John Vines, previously the commander of the 82nd Coalition Task Force (CTF82) in Afghanistan and currently the commander of the 18th Airborne Corps, who wrote:

> MCS Light is the best tool available today … recommend the Army adopt CTF82's employment of MCS-Light as its strategy to rapidly deploy a standard, interoperable, digital command-and-control system Army-wide. [7]

MCS Light has become the planning tool of choice for nine out of 10 active Army divisions. Much can be learned from examining this success.

The MCS Light development process is the result of several years of direct experience developing software in a very dynamic environment. The process is well defined and has been, in fact, in use for several years. And the process has resulted in a high-quality product that has been widely accepted by the user community. Figure 1 is a graphical representation of the defined process.

One of several key aspects of the process is the notion of iterations and releases. The MCS Light project has adopted a four-week iteration and a three-month release cycle. Within an iteration, a team may cycle through the design, construct, integrate, and demo steps several times. This can be done within a team and also across teams within the project. It is also important to note the level at which the definition of what is in a release and what is in an iteration is managed. At the release level, agreement is reached with the PdM. Definition and modification at the iteration level is managed at the project-leader level. Again, this provides for the flexibility needed to effectively manage in a very dynamic environment

Balance Between Plan-Driven and Agile Development

The MCS Light software process has struck a balance between agile development and plan-driven development, or planned agility in the following ways.

Individuals and Interactions Over Processes and Tools

The MCS Light project and its broader organization have consistently placed sig-

nificant emphasis on individuals and have backed up this emphasis with investment. Roughly half of the development team are government civilian employees, the other half are contractors working on-site as an integral part of the team. Software developers represent more than 85 percent of the project staff, and all civilian engineers have either completed or are pursuing advanced degrees in software engineering.

Consistent with agile development approaches, the overall development team is comprised of smaller teams. These teams typically consist of three to 10 individuals who are co-located within the same office. Interaction is informal, constant, and essential to the approach.

Working Software Over Comprehensive Documentation

The MCS Light project has placed considerable emphasis on the software product and has considered extensive documentation as a significant distraction from developing the end product (more than 800,000 source lines of code). Therefore, it concludes that developing such documentation represents an even greater risk than it is intended to avert.

The architecture is extensively documented and that documentation is maintained. In addition, an "MCS Light For Dummies Guide" has been developed as a training guide for users. Additional training documentation has been and will be developed to an even-greater degree as fielding of the MCS Light product progresses. There is also documentation that traces planned and delivered functionality back to the system Operational Requirements Document.

The product itself, as opposed to extensive documentation, has served as the basis for interactions between the user and the development team. Relatively speaking, little documentation has been developed on the MCS Light project, and no one has missed what has not been developed, including those paying the bills.

Customer Collaboration Over Contract Negotiation

A heavy emphasis has always been placed on collaborating with the user. For the MCS Light development team, there is also another customer: the PdM MCS. Interactions with the PdM are frequent and less formal than the requirements-based contracting approach so often implemented within the Army. The overhead associated with detailed contract negotiation – and renegotiation every time a change is necessary – is overly burdensome to any development effort looking to rapidly respond to a customer's needs. The project has adopted the equivalent of a level-of-effort agreement with the PdM. Within this approach, it can measure progress at the standard milestones and measure earned value.

Responding to Change Over Following a Plan

The Army as an institution is well versed in the development of plans. Fortunately, the Army also recognizes that no plan, even the best plan, survives long in a dynamic environment before needing to be revised. Planning for software development is not significantly different than planning for a battle. The MCS Light effort has consistently placed an emphasis on responding to change. This emphasis gives the team the flexibility to respond effectively to the constant evolving and changing user needs.

> *"The MCS Light effort has consistently placed an emphasis on responding to change."*

Get as Close to the User as Possible

On the MCS Light project, the team has been accused in the past of listening too much to the user and the surrogate user, Training and Doctrine Command System Manager (TSM), as opposed to strictly adhering to requirements definitions and programmatic structures. Doing so has served the project well. As stated earlier, as a developer the closer you are to the user, the more likely you will develop something useful. Simply put, that means having software developers and end users working side by side.

On MCS Light, the project leader, all team leaders, and a significant number of project engineers have spent a significant amount of time in the field with users. MCS Light software engineers have worked side by side with users in garrison, at war-fighting exercises, and have even deployed with units to Afghanistan and Iraq. Being that close is harder than not, but it is the only way to develop a useful product.

Show Them, Ask Them, and Repeat Often

Key to the MCS Light success has been establishing a Beta Site concept. Leveraging industry practices, some operational units were identified as official Beta Sites. As a Beta Site, the units were provided with developmental releases of software. The premise was simple: the team would provide incremental releases of software, the user would provide feedback, and the team would respond rapidly where possible with another incremental release.

Instead of having to wait years for a new version of software that would likely not satisfy their needs, users were rapidly and frequently given developmental releases of software that, incrementally, met more and more of their needs. Confidence and trust between the developers and the users were formed. With trust comes the need for less bureaucracy, thereby enabling the streamlining of the approach even more. Since its inception, every active Army division has come online and requested to become an MCS Light Beta Site.

The benefits of this approach cannot be overstated. Through this approach, it is worth noting that Army units have demonstrated a willingness to accept good enough software much sooner over the promise of better quality software much later. If they do not like the product delivered, or the product delivered does not work, the user has no problem saying so.

The best case is that the team rapidly responded to the user's need and got valuable feedback as to what else was needed. The worst case is that the team learned what the user did not want or need, and only lost the time invested since the previous release. In that respect, the Army is no different than commercial industry – time to market or time to field is a priority, and only an agile approach will do.

Architecture, Architecture, Architecture

It would not have been enough to simply be close to the user and provide early and frequent development releases. The product also needed to be sound and evolvable. From the onset of the project, architecture definition and evolution has been a cornerstone of the development effort. Software architecture was defined almost from day one, and a well-defined architecture has been kept up to date and have served as the basis for all development efforts.

Also key to the success of the project has been a well-structured architecture. In the case of MCS Light, a three-tier architecture was defined and adopted. This

architecture has served the project well in allowing developers to leverage COTS products and tools across the different tiers as well as in providing a powerful approach to managing data. While everyone talks about how important architectures are, MCS Light as a project has actually implemented an architecture-based approach to development, and the continued evolution of the product is the best testimony to that case.

Summary

Insanity has been defined as doing the same thing over and over again and expecting different results. If the Army software community is to truly, that is truly, achieve gains in effectiveness and efficiencies, it must be willing to abandon those practices that have not served it well. The Army must be willing to adopt practices that strike a balance between discipline and agility.◆

References

1. U.S. Air Force Software Technology Support Center. "Agile Software Development." CROSSTALK 15.10 (Oct. 2002) <www.stsc.hill.af.mil/crosstalk/2002/10/index.html>.
2. Paulk, Mark. "Agile Methodologies and Process Discipline." CROSSTALK 15.10 (Oct. 2002): 15-18 <www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html>.
3. Boehm, Barry. "Get Ready for Agile Methods, With Care." IEEE Computer Jan. 2002.
4. Agile Alliance. "Agile Software Development Manifesto." 13 Feb. 2001 <www.agilealliance.org>.
5. U.S. Army. "Transforming Current Operations." Association of the United States Army Acquisition Symposium, Falls Church, Va., 8 Sept. 2003.
6. Vines, M.G. John. "Commander CTF82, Memorandum Thru Commander CTF180 and Commander U.S. Central Command For U.S. Army Deputy Chief of Staff for Plans and Operations." 15 Jan. 2003.

About the Author

John S. Willison is director of Advanced Battlespace Solutions for the U.S. Army Communications Electronics Command (CECOM) Software Engineering Center, Fort Monmouth, N.J. CECOM is responsible for developing software architectures and products for Communications, Command, Control, Computer, Intelligence, Electronic Warfare and Sensors systems. Willison is experienced in the application of software technology, software architecture, prototyping, and management. He has received numerous awards, including the Army's Distinguished Service Award, the Secretary of the Army Award for Outstanding Achievement, the Federal Technology Leadership Award, and the Federal 100 Award. Willison has a Bachelor of Science in electrical engineering from Lafayette College and a Master of Science in software engineering from Monmouth University.

CECOM Software Engineering Center
ATTN: AMSEL-SE-AT
Fort Monmouth, NJ 07703
Phone: (732) 532-2342
E-mail: john.willison@us.army.mil