Army Simulation Program Balances Agile and Traditional Methods With Success

LTC John Surdu, Ph.D.
U.S. Army

Doug J. Parsons
Program Executive Office - Simulation Training and Instrumentation

The One Semi-Automated Force (OneSAF) Objective System is the next generation simulation system planned to provide the U.S. Army with an entity-level simulation to serve three modeling and simulation domains. Software development of the OneSAF application has been conducted in a highly robust systems engineering environment based on commercial and government best practices. The OneSAF program has tailored techniques of Extreme Programming (XP) and agile methods into a development environment that has resulted in several industry awards, most recently the National Training Systems Association Cross Function Award for the Integrated Product Team. These externally certified Capability Maturity Model® Integration Level 5 processes are credited with successful program execution. This article will discuss which XP and agile techniques were used, which were not, and why.

Principles and practices associated with agile methods are not new concepts in the world of software development. Extreme Programming (XP), an agile method itself, has been used successfully in a variety of software development environments. Overall reviews in the software development community have been mixed. In his history of the Agile Manifesto [1], Jim Highsmith uses terms like *organizational anarchists* and *independent thinkers* to describe the alliance embracing a unifying set of values. The following is stated in the Agile Manifesto:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
> - Individuals and interactions over processes and tools.
> - Working software over comprehensive documentation.
> - Customer collaboration over contract negotiation.
> - Responding to change over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more. [2]

Pure software development traditionalists consider the techniques that many agile software developers use to follow these values as an excuse to hack undocumented, poorly designed code. The pure agile developer would consider the plan-driven traditionalists responsible for saddling software developers with low-value processes that hinder or prevent the delivery of software. Fortunately, we, as software developers, do not need to take sides. There exist many shades of gray in the spectrum between agile and traditional methods. The shade you select that best fits your program depends upon many factors, including (but not limited to) team size, criticality of defects, ability to receive user feedback/interaction, customer expectations, budget, schedule, and stability of requirements. Since these factors are different from program to program, no two programs should have identical strategies in their software development.

"There exist many shades of gray in the spectrum between agile and traditional methods ... no two programs should have identical strategies in their software development."

Traditional methods are geared toward optimization, predictability, and control. Agile methods focus on adaptation to change, flexibility, and innovation. The new art of software development is finding the appropriate balance point among the available practices. Figure 1 is a chart developed by McCabe and Polen [3] based on a figure from Alistair Cockburn [4] indicating the relative nature between agile and conventional projects based on project team size and notional cost of system failure. Included on the chart is the placement of a U.S. Army simulation program, known as the One Semi-Automated Force (OneSAF) Objective System (OOS).

The OOS is a large software-intensive system (greater than two million lines of source code) that will be used to support research and development and train future U.S. military leaders. In the genesis of the OOS program development, we researched and considered aspects of agile methods and XP as well as traditional strategies. None of these approaches were looked on as cookbook methodologies, but rather as a smorgasbord. Those that appeared to be a good fit were implemented; others were left on the table.

After more than four years of development, the OneSAF program remains on-track to satisfy requirements and meet user needs. The OneSAF program has been awarded the National Training Systems Association Cross Function Award for the Integrated Product Team; it was also selected by CROSSTALK as one of the 2003 U.S. Government's Top 5 Quality Software Projects.

Interestingly enough, the customer base for OOS was more comfortable with *traditional* pedantic software-development methods, even though they contributed to an environment with ill-defined and often-changing requirements. The program spent a great deal of time educating the user representatives on these processes. Most have grown to embrace these methods, but the program still hears some users making statements like, "Spiral development doesn't work; it has created a configuration management nightmare," or "XP has reduced time for testing." Despite statements like these, the processes established for OOS development – a blend of agile, extreme, and traditional techniques – have been instrumental in the program reaching programmatic and technical goals.

This article is not intended to discuss the technical capabilities of the OOS;

® Capability Maturity Model is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

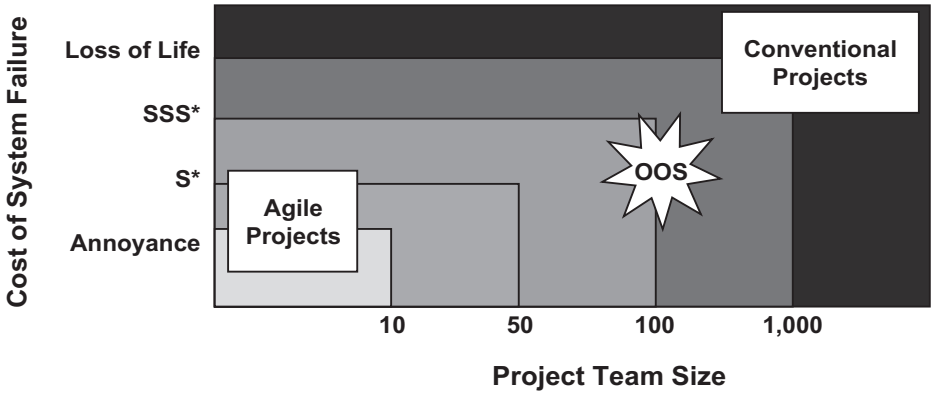
however, it is important that the reader understands some of the program's history to note similarities and distinctions with their own programs, present or past.

Mission Need and the Users

The OOS is the Army's next generation simulation system that can represent a full range of military operations, systems, and control processes. It is an entity-level simulation, meaning that it can simulate the activities of individual combatants or vehicles (as opposed to aggregate-level simulations, which represent combatants and vehicles as groupings). It will also provide the appropriate representations of the physical environment (e.g., terrain features, weather, and illumination) and its effect on simulated activities and behaviors.

The OOS is unique among Army simulations in that it is designed for use by three distinct Army Modeling and Simulation (M&S) domains. Specifically, the Advanced Concepts and Requirements domain uses M&S for experimentation and analysis on Army doctrine and force-related concepts. The Research, Development, and Acquisition domain uses M&S for acquisition analyses focused on equipping and supporting currently fielded and future forces. Finally, the Training, Exercises, and Military Operations domain employs M&S to train forces. It does so using live simulation (actual equipment on training ranges), virtual simulation (immersing the trainee into a synthetic environment), and constructive simulation (war games using computer-generated forces).

There exist two factors relative to our users that inherently pull the OOS development toward the center of the agile-conventional project spectrum. First, OOS will be used extensively for analysis and experimentation. Analysts and research scientists rely on a robust set of documentation to support verification and validation (V&V). For these users, it is critical to understand how OOS models work. Secondly, the OneSAF business model includes the distribution of source code with release of the software baseline [5]. The intention is to create an environment that will optimize the ability for extended capabilities created by the M&S community to be reintegrated into the baseline. Agile projects tend to be light on documentation and process. OOS will include a robust set of documentation and tools to support V&V. In addition, process description and documentation to aid and guide external developers of the baseline will also be provided.



* SSS stands for XXXX and S is for XXXX.

Figure 1: *Spectrum for Project Type Considering Defect Cost and Team Size*

A New Approach

While the use of commercial best practices seems intuitive, five years ago this was quite unique in Department of Defense (DoD) software development. Some of the changes were simple. The approach toward programmatic documentation was a minimalist one. Some approaches were considered heretical: utilizing the OneSAF program manager (PM) as the manager of the OOS task orders (similar to a *prime contractor*), and establishing an integrated development environment (IDE), which made the adoption of many agile methods and XP practices possible.

PM Is the Prime

A typical approach in DoD software development is for the government to select a single contractor who specifies its own set of subcontractors under a large, monolithic contract. In contrast, PM OneSAF was allowed to compete a variety of task orders under an Indefinite Delivery, Indefinite Quantity contract and manage it as the lead systems integrator. The advantage is that the government can pick a *best-of-breed* contractor rather than settle for a sub-contractor who may or may not be the best choice. During the past four years, 26 different contractor companies have been contracted to work on different parts of the OneSAF software. The contracts range in scope and include short-term studies, architecture review, knowledge acquisition, architecture and integration, model development, and integration and test.

An initial concern was that the contractors would disagree on the means to resolve issues and the process would grind to a halt. Associate Contractor Agreements (ACAs) were signed by each task order organization; however, an ACA is only a piece of paper that asks a contractor to play nice. To help *socialize* the ACAs, the contracts were awarded over the space of 18 months. The Architecture and Integration (A&I) contract was awarded first, and the processes, tools, and procedures were established. When new contractors came on board, they were integrated into existing processes so that we avoided *food fights* about whose processes were better than others. This does not imply that existing processes were never modified; we remained committed throughout execution to continuous process improvement and aggressively sought new ideas.

Three factors contributed to the ability for PM OneSAF to successfully act as the prime. First, the OneSAF government team is involved in the day-to-day development process. This allows informed and timely decisions to be made on behalf of the PM. Second, PM OneSAF empowered the A&I contractor with a great deal of flexibility to establish development, integration, and test processes from industry best practices: There were no government-mandated processes heavy with valueless documentation. Third, the PM sought technically qualified folks across the breadth of the program. Not only are OneSAF engineers truly technical with advanced degrees in software-related disciplines and years of real engineering experience, but OneSAF managers and project directors were recruited from engineering. As there are numerous disincentives for technical people within the Army, it was quite challenging to find even the small number of qualified engineers we needed.

It is unclear whether these unique business processes alone were the major contributor to successful execution or whether the employment of agile methods was the key. It is clear, however, that without these new ways of doing business, it is unlikely that agile methods would have been employed or embraced.

IDE

To support an effective and efficient software development process, PM OneSAF established an environment that brought together domain/user representatives, government engineers, PMs, and contract or software developers. All of these teams are collocated in a single facility. A task order for facility operation and sustainment was intentionally awarded to a contractor outside any of those developing software to send the message that the IDE was *neutral turf*. Over the past four years, there have been as many as a dozen individual task orders under execution at any one time. From these task orders, there have been more than 100 software engineers working in concert to develop a baseline.

Applying Agile Methods and XP to OneSAF

Communication and Collocation

The Agile Manifesto states that in agile software development, the following should occur [6]:

- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Paulk states that agile methods generally apply to smaller teams working in the face of vague and/or dynamic requirements [7]. He also states that agile teams are expected to be collocated with typically fewer than 10 members. XP rules also indicate that the customer should always be available and that a stand-up meeting starts each day to communicate problems, solutions, and promote team focus.

OOS development has practiced collocation from its inception. As mentioned earlier, the government program office is the *prime* for OOS, and there are numerous contractors working the program. All contractors from the 10 to 12 companies working on OOS are collocated in the IDE facility. In addition, the combat developer (requirements steward and customer), customer representatives (from the three Army M&S domains), and government engineers and managers are collocated in the same facility. It often takes a newcomer to the program months to find out what company everyone works for because on Team OneSAF we concentrate on functional decomposition of the problem more than on which company is working a portion of the problem.

Through the use of ACAs, communication between the various organizations is smooth and seamless. The face-to-face coordination referred to in the Agile Manifesto occurs habitually in the IDE. A member of the combat development team said recently, "Much of the work on OOS occurs in the hallways." Informal meetings with the right two or three people in the hallway often work through some technical or inter-team coordination issue in a few minutes rather than scheduling a meeting with dozens of folks. OOS developers are encouraged to get up and walk down the hallway if they have an issue. From the earliest days of OOS development, the OneSAF PM Office prohibited weekly scheduled meetings on Wednesday through Friday. A lesson learned from other program development efforts was that engineers tended to save communication with other people on the team until the scheduled meeting. Intentionally, this forced the engineers to meet more frequently in desk-side or hallway meetings.

Spiral Development: Builds, Blocks, and Early Delivery

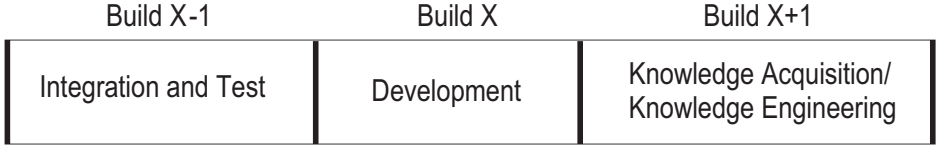
In discussing requirements, McCabe states:

> … the customer may hardly grasp the problem, much less the best system to address it. Therefore, requirements are likely to be vague or speculative when they should be specific. [2]

Recognizing that the customers' requirements and desires would evolve as they saw working prototypes (or alpha versions of the software), OOS adapted a spiral development methodology. Paulk noted:

> … with their emphasis on addressing requirements volatility, agile methodologies could be a powerful synthesis of practices that DoD contractors could leverage to make planning more responsive to change. [7]

Figure 2: *Three-Build Process for Creating a Functional Capability in OOS*



The Agile Manifesto supports the notion of spiral development with the following tenets [6]:

- The highest priority is to satisfy the customer though early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Working software is the primary measure of progress.

XP rules and practices concerning the software development process include the following [8]:

- Make frequent small releases.
- The project is divided into iterations.
- Integrate often.
- Iteration planning starts each iteration.
- Never add functionality early.
- Code must be written to agreed-upon standards.

The development methodology adopted for OOS was one of frequent iterations, or spirals. It involved breaking the overall program into a series of eight- to 10-week builds. Several of these builds were then designated as user assessment baselines that were made available to users for assessment and azimuth correction. User involvement is discussed below.

The early and frequent delivery of builds was the OOS' way of implementing these tenets of the manifesto. Being able to see working code – even if that code initially was focused on architecture and tools rather than interesting military capability – gave the users confidence in the process and the development team. As McCabe and Polen state, "Until a usable system is delivered, the customer has nothing to show for its investment" [3].

Paulk asserted the following:

> Agile methodologies, with their rapid iterations, require continual planning. Customer collaboration and responsiveness to change are tightly linked, if perhaps inconsistent with typical government-contractor relationships. [7]

The overall, strategic program requirements and overall architecture changed little over the four years of software development; however, the specific, tactical requirements and many of the smaller design decisions were made as late as possible.

The overall goals of a particular block were locked down during the block planning a few weeks before the beginning of

the block, not at the beginning of the program in a *big bang*. Additionally, the fine-grained requirements and design of a build were finalized only one build before execution. While executing build *X*, the program is planning build *X*+1 and testing Build *X*-1. A change in requirements, therefore, could be reacted to within 16 weeks, not a year. Figure 2 illustrates this process.

A major advantage of a spiral methodology was the ability of the program to adopt to requirements changes. McCabe and Polen asserted, "When needs are changing, the value of the original system as specified, however optimum it was at the time of its conception, depreciates daily" [3]. During program development, the Army made a radical change in organization to a focus on Brigade Combat Teams. The OOS program was able to rapidly shift focus between spirals so that it would be delivered with the new force structure rather than the old. Additionally, the threat faced by U.S. forces changed. OOS was able to curtail representation of older, Soviet-style opposing forces and implement a more contemporary, unconventional enemy.

Customer Involvement in Development

Turner and Boehm state:

> One of the major differences between agile and plan-driven methods is that agile methods strongly emphasize having dedicated, collocated customer representatives, while plan-driven methods count on a good deal of up-front, customer-developer work on contractual plans and specifications. [9]

Because the customer representatives are collocated in the IDE, they participate in all our meetings, and are available to answer questions or reach back into their customer base for feedback and input.

Several of the OOS builds were designated User Assessment Baselines (UAB). These UABs were available in the IDE for users to evaluate. In addition, a number of assessment events were held in the IDE during which users from around the Army were invited to participate. Finally, the PM office took the software to user sites for more formal assessments. It may seem obvious to some, but users use the software differently than engineers and developers. While the users often tried to treat these developmental assessments as operational tests, resulting in often vitriolic feedback, the events were excellent opportunities to allow users to identify bugs in the software in an operational-like environment.

Documentation Versus Working Code

OOS Vers. 1.0 represents Build 29 of the software. Builds can vary between eight and 10 weeks in duration, depending upon the difficulty of the tasks in that build, and how they fit in the timeframe of the block. Since Build 4, OOS has had working software that could be demonstrated and made available for user feedback. As McCabe and Polen state, "Your only real knowledge comes from a working system" [3].

Having working code since Build 4 reflects the agile method's bias toward working code rather than documentation-centric development. However, extensive design documentation, knowledge acquisition documentation, and technical notes do exist and are reposed in <www.OneSAF.net>, the program's collaborative information warehouse. In addition, user documentation is maintained in a form that is simultaneously compiled into a users' manual and online help.

"Being able to see working code – even if that code initially was focused on architecture and tools rather than interesting military capability – gave the users confidence in the process and the development team."

Paulk states that when employing agile methods, a project must:

> Decide where to place the balance point in documentation and planning to alleviate the concerns of the stakeholders (and regulatory requirements) while achieving the flexibility and benefits promised in the agile philosophy. [7]

This is not to imply that the OOS software is not adequately documented. The A&I contractor, however, has been externally certified at Capability Maturity Model Integration (CMMI®) Level 5, so the development processes are well documented. Not only are the processes documented, but the results of the processes (the development products/artifacts) are captured. Metrics are captured on the execution of processes as well, and the A&I contractor conducts periodic *defect prevention* sessions to examine and correct the root cause of common issues.

The documentation of these processes and products is captured in a Web-enabled manual known as the Electronic Process Guide. The artifacts that document developers' adherence to these processes are found in several Web-enabled repositories such as the online Software Development Folders, Web-based tracking tools for action items, trouble reports, defects, peer reviews, and risks. The very nature of having a Web-enabled tool set reduced the burden on developers for complying with these processes and enabled communication across teams.

Extreme Testing

With respect to testing, agile methods indicate the projects should include the following [8]:

- Code the unit test first.
- All code must have unit tests that must pass before being released.
- When a bug is found, tests are created.
- Acceptance tests are run often and the score is published.

Whenever an OOS developer commits his or her changes to configuration management (i.e., Concurrent Versions System), a tool – called BuildBoy – builds the software for Linux, Windows, and Solaris and runs nearly 3,000 automated tests on each operating system. If the build fails, the developer and his or her supervisor are sent an e-mail. The developer can then reference the BuildBoy Web page to determine the nature of the failure and take corrective action. In this way, the software is built and subjected to unit and integration tests on average of eight times a day.

McCabe and Polen state:

> Writing automated unit tests first is a clever way of inducing developers to not only unit test their code efficiently, but also to write their code efficiently without superfluous logic. [3]

While we agree in concept, this is an area in which we need improvement. All code is delivered at the end of a build with

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

appropriate JUnit tests, and some of the teams do build the tests first, but we are not consistently using the methodology of building the test first.

With respect to the tenet of building a test when a bug is found, we have adapted this process to large, complex software. While we do not build a test for every bug that is fixed, when we find the same pattern of bug appears many times, we build an automated test to trap it.

Conclusion

The OOS software is being developed through a combination of agile, extreme, and traditional techniques. We have not blindly adopted all of the techniques and tenets of these agile approaches; however, we have used a great many of them. The program still has room for improvement in some areas, and our CMMI Level 5 continuous process improvement is ever vigilant for opportunities to do so. In particular, we are in need of more agile testing and an increased number of automated tests.

The unique blend of these techniques has been instrumental in the award-winning success of the OOS development effort. Some readers may wonder which of these techniques were most beneficial and if they could be applied to other programs with equal success. The answer to both of these questions is based upon our original premise: the best method depends upon the nature of the program itself. Program characteristics such as team size and impact of system failure are evident; however, each PM needs to consider other issues that define the technical, programmatic, and political landscape: Will upper management support less conventional techniques? Does the government team have the appropriate skill set to work closely with the software development team, not just monitor contractor activity and check end results? Are the users open to agile methods and willing to actively participate throughout the process?

The best methods are as unique as the programs themselves. Since our program initiation, we have engaged with other organizations that have tried to emulate the form – without the substance – of these agile methods and innovative business processes; their successes have been limited. The authors considered what methods and techniques worked best for the OneSAF program. After pondering this question, the answer illustrates the shades of gray between traditional and agile methods. Strictly following the CMMI Level 5 processes (traditional) and the individual interactions (agile) among users, developers, and government representatives were essential contributors. Four years ago we did not know what would work best, or what would work at all for that matter. However, we were willing to try something innovative, willing to make changes along the way, and bold enough to see it through.◆

References

1. Highsmith, J. "History: The Agile Manifesto." The Agile Alliance, 2001 <http://agilemanifesto.org/history.html>.
2. Agile Alliance. "Agile Software Development Manifesto." Agile Alliance, 13 Feb. 2001 <www.agilemanifesto.org>.
3. McCabe, R., and M. Polen. "Should You Be More Agile?" CROSSTALK Oct. 2002 <www.stsc.hill.af.mil/crosstalk/2002/10/mccabe.html>.
4. Cockburn, Alistair. "Learning From Agile Software Development - Part One." CROSSTALK Oct. 2002 <www.stsc.hill.af.mil/crosstalk/2002/10/cockburn.html>.
5. Parsons, D., and R. Wittman. "Open Source Opens Opportunities for Army's Simulation System." CROSSTALK Jan. 2005 <www.stsc.hill.af.mil/crosstalk/2005/01/0501parsons.html>.
6. Agile Alliance. "Principles Behind the Agile Manifesto." Agile Alliance, 30 May 2005 <http://agilemanifesto.org/principles.html>.
7. Paulk, M. "Agile Methodologies and Process Discipline." CROSSTALK Oct. 2002 <www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html>.
8. Wells, D. "The Rules and Practices of Extreme Programming." Extreme Programming. 28 Feb. 2004 <www.extremeprogramming.org/rules.html>.
9. Turner, R., and B. Boehm. "People Factors in Software Management, Lessons From Comparing Agile and Plan-Driven Methods." CROSSTALK Dec. 2003 <www.stsc.hill.af.mil/crosstalk/2003/12/0312turner.html>.

About the Authors

LTC John "Buck" Surdu, Ph.D., is the product manager for the One Semi-Automated Forces Objective System. Originally commissioned as an infantry lieutenant, Surdu served in operational assignments in the 82nd Airborne Division, Europe, and Korea. He worked as a research scientist at the Army Research Laboratory and a senior research scientist and assistant professor in the Information Technology and Operations Center within the Department of Electrical Engineering and Computer Science at West Point. He has a Bachelor of Science in computer science from the United States Military Academy, West Point, a Master of Science in computer science from Florida State University, a Master of Business Administration from Columbus State University, and a doctorate in computer science from Texas A&M University.

12350 Research PKWY
Orlando, FL 32826-3276
Phone: (407) 384-5103
Fax: (321) 235-1484
E-mail: john.surdu@us.army.mil

Doug Parsons is the lead engineer of the Intelligent Simulation Systems Team at the U.S. Army Program Executive Office for Simulation, Training, and Instrumentation. His primary focus is toward the successful development of the One Semi-Automated Forces Objective System. Parsons has a Bachelor of Science in mechanical engineering from North Dakota State University, a Master of Science in systems management from Florida Institute of Technology, and a Master of Science in industrial engineering from the University of Central Florida.

Program Executive Office-Simulation Training and Instrumentation (PEO-STRI)
12350 Research PKWY
Orlando, FL 32826-3276
Phone: (407) 384-3821
E-mail: doug.parsons@us.army.mil