OneSAF: A Case Study in Agile Methods and Extreme Programming

Mr. Doug Parsons, LTC John Surdu
US Army, PEO STRI
Orlando, FL
doug.parsons@peostri.army.mil
john.surdu@peostri.army.mil

Mr. Derrick Franceschini
Science Application International Corporation
Orlando, FL
derrick.j.franceschini@saic.com

ABSTRACT

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 (AM) into a development environment that has resulted in several industry awards, most recently the National Training Systems Association (NTSA) Cross Function Award for the Integrated Product Team. These externally-certified CMMI level-5 processes are credited with keeping the program on-schedule and on-budget, while meeting performance requirements. This paper will discuss which XP and AM techniques were used, which were not, and why.

ABOUT THE AUTHORS

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 (PEO STRI). In this role his primary focus is toward the successful development of the One Semi-Automated Forces (OneSAF) Objective System (OOS). Mr. Parsons holds a B.S. in Mechanical Engineering from North Dakota State University, a M.S. in Systems Management from Florida Institute of Technology, and a M.S. in Industrial Engineering from the University of Central Florida.

LTC John "Buck" Surdu is the Product Manager for OneSAF, both OneSAF Testbed Baseline (OTB) and OOS. Originally commissioned as an infantry lieutenant he 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 (ITOC) within the Department of Electrical Engineering and Computer Science at West Point. He holds a Ph.D. in computer science from Texas A&M University, an M.S. in computer science from Florida State University, an MBA from Columbus State University, and a B.S. in computer science from the United States Military Academy, West Point.

Derrick Franceschini is the Product Line Architecture Development Manager for the Architecture and Integration Task Order for OOS for SAIC. He has previously served as the Common Services Team lead for OneSAF, as well as the ModSAF and OTB Project Engineer. He holds a B.S. in Computer Engineering from the University of Central Florida.

OneSAF: A Case Study in Agile Methods and Extreme Programming

Mr. Doug Parsons, LTC John Surdu
US Army, PEO STRI
Orlando, FL
doug.parsons@peostri.army.mil
john.surdu@peostri.army.mil

Mr. Derrick Franceschini
Science Application International Corporation
Orlando, FL
derrick.j.franceschini@saic.com

INTRODUCTION

Principles and practices associated with Agile Methods (AM) and Extreme Programming (XP) are not new concepts in the world of software development. Reviews in the software development community have been mixed. In his history of the Agile Manifesto Jim Highsmith uses terms like 'organizational anarchists' and 'independent thinkers' to describe the alliance embracing a unifying set of values (Highsmith, 2001). As stated in their *Manifesto for Agile Software Development* –

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.

Pure traditionalists of software development consider the techniques that many *agilists* use to follow these values as an excuse to hack undocumented, poorly designed code. The pure agilist 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 is dependent upon many factors to include (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. 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 an adaptation of a McCabe chart (McCabe, 2002) indicating the OOS relation to typical agile and conventional projects based on our project team size and notional cost of system failure.

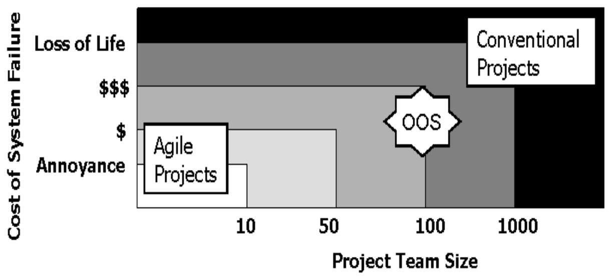


Figure 1. Spectrum for Project Type Considering Defect Cost and Team Size.

The purpose of this paper is to convey the strategy that PM OneSAF used to develop the OneSAF Objective System (OOS) baseline. In the genesis of program development we researched and considered aspects of agile methods, and extreme programming, as well as traditional strategies. None of these approaches were looked upon as cookbook methodologies, but we rather likened them to a smorgasbord. Those that appeared to be a good fit were implemented; others were left on the table. After more than three years of development and nearly 2 million lines of code, the OneSAF program remains on-budget, on-schedule, and on-track to satisfy requirements. The OneSAF program has been awarded the National Training Systems Association (NTSA) Cross Function Award for the Integrated Product Team, as well as selection by CrossTalk (Journal of Defense Software Engineering) as one of the top 5 software quality 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 "extreme programming 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 paper reviews the methods that were considered and provides rationale for the software development approaches selected for implementation.

BACKGROUND ON THE ONESAF PROGRAM

This paper is not intended to discuss the technical capabilities of the OneSAF Objective System (OOS); however, it is important that the reader understands some of the program's history in order to note similarities and distinctions with their own programs, present or past.

Mission Need and the Users

The OneSAF concept originated in January 1996 following an extensive study that came to the conclusion that the Army was caught in a wasteful spending cycle, making identical or similar enhancements to legacy simulations across three different user domains – and often more than once within the *same* user domain. Furthermore, it was determined that none of the existing legacy simulations were capable of being extended to provide comprehensive support of emerging Army functional requirements and technical standards.

Realizing this, the Army decided the best approach for overcoming the problems associated with the multitude of aging simulations was to create a single general-purpose entity level simulation. In May of 1997 the Deputy Commanding General (DCG), Training and Doctrine Command (TRADOC), approved the Mission Needs Statement (MNS) for OneSAF which stated:

> *"The need for OneSAF capabilities is not a response to a specific warfighting threat against the force; the need is driven by the guidance to reduce duplication of M&S investments, foster interoperability and reuse across M&S domains, and meet the M&S requirements of the future force."*

To satisfy this need OOS will become the US Army's next generation, composable, entity-based simulation system. The fact that the OOS is being developed from day-one to serve all three of the Army's modeling and simulation domains makes it quite unique. It will provide an integral simulation service to the Advanced Concepts and Requirements (ACR); Training, Exercises, and Military Operations (TEMO); and Research, Development, and Acquisition (RDA) domains. Once available for fielding, the OOS may become the most widely distributed Army software simulation application to date.

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 (Parsons and Wittman, 2005). The intention is to create an environment which will optimize the ability for extended capabilities created by the modeling and simulation community to be reintegrated into the baseline. Agile projects tend to be documentation and process light. OOS will include a robust set of documentation and tools to support V&V. In addition, the OneSAF Electronic Process Guide provides links to process description and documentation to aid and guide external developers of the baseline.

A New Approach

Shortly after the Mission Need Statement was approved, a Cross-Domain Integrated Concept Team was established to design a simulation acquisition strategy, begin harmonizing user requirements, and perform early architecture analysis. The OneSAF Program Management Office (PMO) sought the council of the senior leadership at PEO STRI. A Red Team was formed, which worked with PM OneSAF to identify opportunities to "tailor out" heavy-weight processes that add limited value to the overall program. In addition, the Red Team empowered PM OneSAF to work within acquisition reform guidelines to adopt commercial best practices.

While the use of commercial best practices seems intuitive, five years ago this was quite unique in defense software development. Some of the changes were simple. The approach toward programmatic documentation was minimalistic. Even the most complicated task orders were executed from Statements of Objectives (SOOs), rather than Statements of Work (SOWs). In the DoD acquisition process Milestone B designates a decision to allow a program to enter the system development and demonstration phase. At Milestone C a decision is made regarding production and deployment. The OneSAF PMO recognized that combining these milestones would not create significant risk, yet result in considerable programmatic benefits. The Milestone Decision Authority agreed and Milestones B and C were combined. There were some approaches that were considered to be heretical: PM OneSAF as the manager of the OOS task orders similar to a "prime contractor" and the establishment of an Integrated Development Environment (IDE), which made the adoption of many agile methods and extreme programming practices possible.

PM is the Prime – The typical approach at PEO STRI to software development is for the government to select a single contractor, who specifies their 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 (IDIQ) contract called the STRICOM Omnibus Contract (STOC) and manage them as a 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. The 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 (ACA) were signed by each task order organization; however, an ACA is only a piece of paper that asks a contractor to play nice. Two 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. Secondly, PM OneSAF empowered the Architecture & Integration (A&I) contractor with a great deal of flexibility to establish development, integration, and test processes from industry's best practices. There were no government mandated processes, heavy with valueless documentation.

Integrated Development Environment (IDE) - In order to support an effective and efficient software development process, PM OneSAF established an environment which brought together domain/user representatives, government engineers, program managers, 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 in order to send the message that the IDE was "neutral turf." Over the past three 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 a hundred software engineers working in concert to develop a baseline.

HOW FACTORS OF EXTREME PROGRAMMING AND AGILE METHODS ARE APPLIED TO ONESAF

This section of the paper will describe many of the characteristics of agile methods and extreme programming. Where they are not self-explanatory, those factors will be described in some detail. Then how these factors have been applied or adopted for OOS development will be described.

Communication and Collocation

The Agile Manifesto states that in agile software development (http://agilemanifesto.org/principles.html):

- 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 (see Figure 1) working in the face of vague and/or dynamic requirements (Paulk, 2002). He also states that agile teams are expected to be collocated, with typically less than 10 members. Extreme programming 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 all ten to twelve companies working on OOS are collocated in the Integrated Development

Environment facility in Orlando. In addition, the combat developer (requirements steward and customer), customer representatives (from the three Army modeling and simulation 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 Associate Contractor Agreements, 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." Quick standup 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.

The OOS team is not small or medium sized. With over 150 developers at any given time, some of the agile methods have had to be adapted. The overall team, however, is broken into many smaller teams that are small to medium sized. A practice of extreme programming is to start each day with a standing meeting. While we do not employ this practice at the beginning of the day for the overall program, many of the smaller teams do.

OOS is developed in a spiral development methodology broken into (approximately) annual Blocks containing eight- to ten-week Builds. We conduct Block Planning about half-way through the current Block for the next Block. Block planning establishes the overall goals for development over the coming year, based on ensuring requirements satisfaction at Full Operational Capability (FOC), staffing levels, and customer expectations. This high-level plan is decomposed into detailed Build Plans. In planning for a Build, we hold a dependency meeting in which the team leads hash out their dependencies on other teams during the upcoming build. In addition, before each build we hold a Build Kickoff Meeting in which each team lead briefs what their team will be developing for the next eight to ten weeks. The purpose of this meeting is to double check that dependencies have not been missed, and generally one or two issues are caught at this typically one-hour meeting.

Spiral Development: Builds, Blocks, and CMMI Level 5

In discussing requirements, McCabe states that "the customer may hardly grasp the problem, much less the best system to address it. Therefore, requirements are likely to be vague or speculative where they should be specific." (McCabe, 2002) 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 that "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." (Paulk, 2002)

The Agile Manifesto supports the notion of spiral development with the following tenets (http://agilemanifesto.org/principles.html):

- *Our 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*

Extreme programming rules and practices concerning the software development process include (http://www.extremeprogramming.org/rules.html):

- *User stories are written in place of a large requirements document*
- *Make frequent small releases*
- *The project is divided into iterations*
- *Integrate often*
- *Iteration planning starts each iteration.*
- *Always do the simplest thing that could possibly work*
- *Never add functionality early*
- *Refactor whenever and wherever possible*
- *Code must be written to agreed standards*

- *Leave optimization until last*

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 ten-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 states, "until a usable system is delivered, the customer has nothing to show for its investment." (McCabe, 2002)

Paulk asserted that "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." (Paulk, 2002) 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.

Build X-1	Build X	Build X+1
Integration & Test	Development	Knowledge Acquisition/ Knowledge Engineering

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

A major advantage of a spiral methodology was the ability of the program to adopt to requirements changes. McCabe asserted that "when needs are changing, the value of the original system as specified, however optimum it was at the time of its conception, depreciates daily." During program development, the U.S. Army made a radical change in organization from brigades and divisions to units of action, or brigade combat teams. The OOS program was able to rapidly shift focus between spirals so that it will be delivered with the new force structure, rather than the old. In addition, the enemy faced by U.S. forces changes. OOS was able to curtail representation of older, Soviet-style opposing forces and implement a more contemporary, unconventional enemy.

Refactoring is a set of disciplined techniques for safe design simplification and improvement. There can be a fine line between spiral development and not doing it right the first time. Too much refactoring can indicate bad design decisions up front. What is the balance? That is a subjective assessment. Our lead architect and product line manager have to manage this balance on a daily basis.

Customer Involvement in Development

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

In addition, we have almost 50 "beta sites," where early drops of OOS have been delivered. The purpose of these beta sites is to provide feedback to the developers on the software early in the process when making corrections is easier and less expensive. We have provide all beta sites access to a Web-based user feedback tool that allows the users to submit bug reports, usability issues, etc. directly to the developers. The Problem Trouble Report (PTR) Review Board (PRB) periodically reviews the user feedback comments and assigns them to various teams for corrective action.

In addition to software distribution to Beta sites, 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.

Paired Programming

Extreme programming rules promote the use of paired programming, in which two programmers work at the same workstation with one writing code and the second performing immediate code audits. This has been shown to be effective and efficient, if counterintuitive (http://www.extremeprogramming.org/rules/pair.html). One of the authors personally worked in a paired programming environment for several months and can attest to its effectiveness. It helps less-experienced programmers gain experience quickly, provides immediate code audits, and leverages "two heads" to think through problems.

While we do not employ paired programming on OOS as a general rule, we often break out small "tiger teams" to tackle short duration, time sensitive, important tasks. We often see these tiger teams employing techniques that look strikingly like paired programming.

Documentation vs. Working Code

OOS version 1.0 will represent Build 28 of the software. Builds can vary between eight and ten weeks in duration, depending on how many holidays there are within the build, the difficulty of the tasks in that build, and how they fit in the timeframe of the Block. At the time this article is being written, OOS is testing Build 25, executing Build 26, and planning Build 27. Since Build 4, OOS has had working software that could be demonstrated and made available for user feedback. As McCabe states, "your only real knowledge comes from a working system." Through the use of CVS, the OOS team can regenerate any build of the software from Build 4 to Build 25.

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

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." (Paulk, 2002) This is not to imply that the OOS software is not adequately documented. The architecture and integration 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 also the results of the processes (the development products/artifacts) are captured. Metrics are captured on the execution of processes as well, and the Architecture and Integration 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 on-line 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.

In addition, as mentioned earlier, the intent for the analysis community to use OOS drove a requirement for voluminous documentation of models, algorithms, behaviors, architecture, and design. Volume 3 of the Product Line Architecture Specification of OOS is specifically designed to facilitate development of OOS modules and enhancements by developers external to the base program team.

Occasionally the users want to know what exactly is going to be in Build *X*. Since the exact functionality being built in an upcoming Build is planned one build prior, it is often hard to answer that question. In addition, the focus on developing new software can mean that exact documentation of what functionality is present in Build *X* is often not available in a form the user would like. The user might have to look in multiple sources to get a comprehensive view of the entities, units, behaviors, physical models, environment (i.e., terrain) capabilities, usability features, etc. that should work in a given Build.

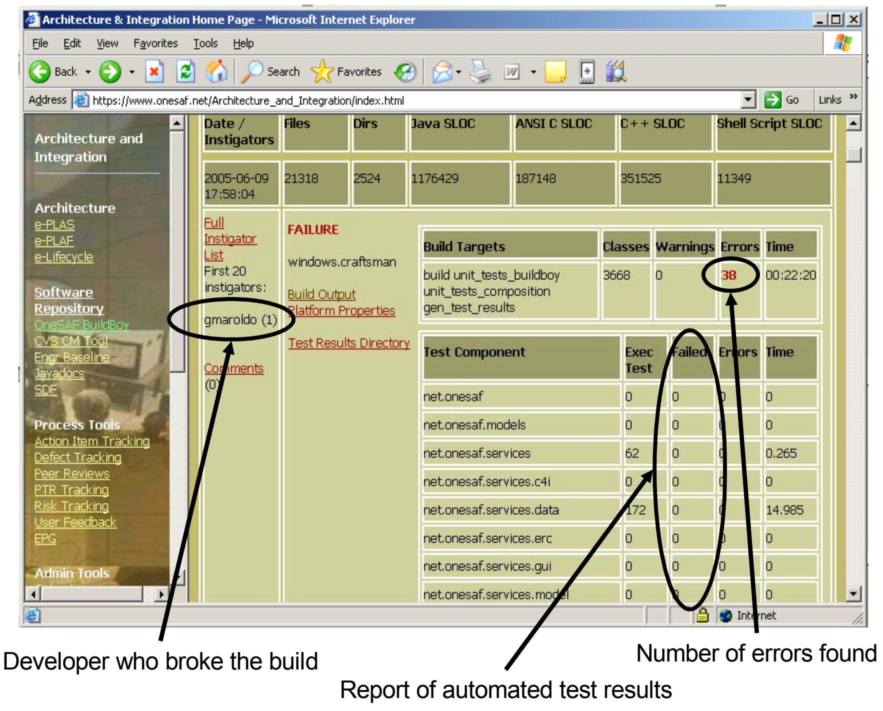
Extreme Testing

With respect to testing, agile methods indicate the projects should:

(http://www.extremeprogramming.org/rules.html)

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

This paper has already described the build and block process of OOS. In addition, whenever a developer commits their changes to CVS (http://www.gnu.org/software/cvs/) a tool, called BuildBoy, builds the software for Linux, Windows, and Solaris, and runs nearly 3000 automated tests on each operating system. If the build fails, the developer and his or her supervisor are sent an Email. 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 eight times a day.



Date / Instigators	Files	Dirs	Java SLOC	ANSI C SLOC	C++ SLOC	Shell Script SLOC
2005-06-09 17:58:04	21318	2524	1176429	187148	351525	11349

Full Instigator List
First 20 instigators:
gmaroldo (1)
Comments (0)

FAILURE
windows.craftsman
Build Output
Platform Properties
Test Results Directory

Build Targets	Classes	Warnings	Errors	Time
build unit_tests_buildboy unit_tests_composition gen_test_results	3668	0	38	00:22:20

Test Component	Exec Test	Failed	Errors	Time
net.onesaf	0	0	0	0
net.onesaf.models	0	0	0	0
net.onesaf.services	62	0	0	0.265
net.onesaf.services.c4i	0	0	0	0
net.onesaf.services.data	172	0	0	14.985
net.onesaf.services.erc	0	0	0	0
net.onesaf.services.gui	0	0	0	0
net.onesaf.services.model	0	0	0	0

Figure 3: A view of BuildBoy results

McCabe states that "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." (McCabe, 2002) 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 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 this bug.

Agile Tools

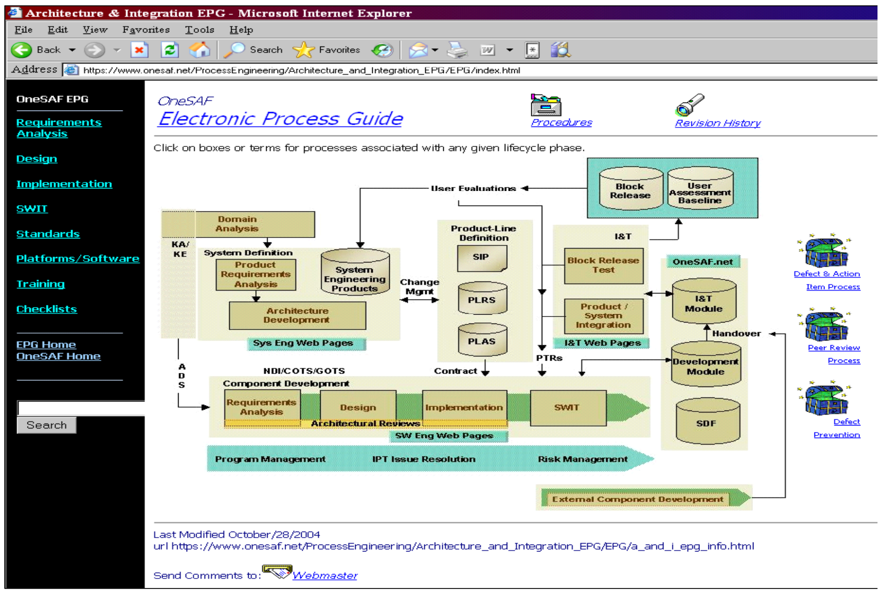
In order to support agile software development, we have applied "agile tools." Many of these tools, such as the Electronic Process Guide and BuildBoy, have been discussed previously in the paper. Additional tools used include OneSAF.net, compDocs, and the EPLAF.

OneSAF.net is a Web-based repository of all information regarding the OOS development effort. One can trace requirements from the requirements document down to the JavaDocs for objects and methods that help satisfy a given requirement. Records of all past meetings, system technical notes describing implementation decisions, WebRT (http://packages.debian.org/stable/virtual/webrt) tracking of bugs and their resolution, knowledge acquisition documents, etc. are reposed and accessible through the OneSAF.net interface.

One of the documents that is available through OneSAF.net is the Electronic Process Guide (EPG), as shown in Figure 4. The EPG describes all the processes used by the developers on a daily basis, as well as the mechanism for modifying (improving) those processes. Our processes are thereby made available to prospective co-developers for tailoring. It also explains to those co-developers what is expected of them at software handover, for instance.

Another tool available on OneSAF.net is something we refer to as compDocs, or composition documentation. This documentation, built by "scraping the code," is a comprehensive, hyperlinked description of the various unit, entity, and behavior compositions that are part of the baseline. This tool lowers the bar for developers to comply with the documented processes. Instead of documenting a product in two disparate tools, OOS generates the documentation of a product from the product itself. This has the side benefit guarantying the documentation is up-to-date and reduces staffing needed to produce the documentation.

The reader may notice that many of these tools are open source tools. OOS will be delivered as an open-source product under a government open source "license agreement." To facilitate development, many of the tools we use are open source, and we have leveraged existing government and open source (e.g., Open Map) products.



Click on boxes or terms for processes associated with any given lifecycle phase.

Last Modified October/28/2004
url https://www.onesaf.net/ProcessEngineering/Architecture_and_Integration_EPG/EPG/a_and_i_epg_info.html

Figure 4: Sample, hyperlinked view of the Electronic Process Guide

SUMMARY

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. After four years of coding, the program continues to meet cost, schedule, and performance goals and is on track to deliver a product in March 2006 – the date set four years ago.

REFERENCES

Agile Alliance, Principles Behind the Agile Manifesto, Retrieved on 30 May 2005 from http://agilemanifesto.org/principles.html.

DoDI 5000.2, Operation of the Defense Acquisition System, 23 October 2000.

Highsmith, J. (2001), *History: The Agile Manifesto*, http://agilemanifesto.org/history.html.

Highsmith, J. (2002), *What is Agile Software Development?*, CrossTalk, Oct 2002 issue taken from http://www.stsc.hill.af.mil/crosstalk/2002/10/highsmith.html.

McCabe, R. & Polen, M. (2002), *Should You Be More Agile?*, CrossTalk, Oct 2002 issue taken from http://www.stsc.hill.af.mil/crosstalk/2002/10/mccabe.html.

McMahon, P. (2005), *Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective*, CrossTalk, Vol. 18, No. 5, pg. 16-19.

OneSAF Mission Need Statement, HQ TRADOC, DCSSA, 19 June 1997, CARDS reference number 04-97.

OneSAF Operational Requirements Document (ORD) Version 1.1, date approved August 2001, US Army Training and Doctrine Command (TRADOC).

Parsons, D. & Wittman, R., *Open Source Opens Opportunities for Army's Simulation System*, CrossTalk, Jan 2005, Vol. 18, No. 1, pg. 11-14.

Paulk, M. (2002), *Agile Methodologies and Process Discipline,* CrossTalk, Oct 2002 issues taken from http://www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html.

Turner, R. & Boehm, B., *People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods*, CrossTalk, Vol. 16, No. 12, Pg. 4-8.

Wells, D. (2004), *Rules and Practices of Extreme Programming*. Retrieved on 31 May 2005 from http://www.extremeprogramming.org.