INCOSE BEST PAPER AWARD

27th Annual INCOSE International Symposium (IS 2017)
Adelaide, Australia, July 15-20, 2017

Case Study: Agile SE Process for Centralized SoS Sustainment at Northrop Grumman

Rick Dove
Paradigm Shift International
rick.dove@parshift.com

Bill Schindel
ICTT System Sciences
schindel@ictt.com

Abstract. In 2015 Northrop Grumman's GCSS-J Systems Engineering group in Mclean, Virginia, supporting the DISA GCSS-J PMO, had been sustaining and evolving a critical information service portal for 12 independent user groups accessing 22 independent systems. This web-based portal is a centralized Systems-of-Systems (SoS) hub, dealing with unpredictable independent-system changes, mitigating immediate-priority security needs, and replacing uncontrollable obsolescence of COTS software elements – all the while deploying new capability in six month increments requested by expectant users. The systems engineering process combines elements of Scrum, and contract waterfall requirements simultaneously on three release instances, one in development, one in accreditation test, and one in deployed use – with a wave-like transition among the three instances every six months. The process had six years of effective employment and evolution, winning praise from GAO and users alike. Most notable is the real-time control model for re-prioritizing work-in-process, the intimate involvement of customer and users in the agile systems engineering process, and the never-ending evolution with all life-cycle stages in simultaneous activity.

Introduction

An INCOSE project-in-process is seeking a generic Agile Systems Engineering Life Cycle Model (ASELCM). The project reviews effective agile systems engineering (SE) processes in three-day, on-site, structured-analysis workshops; and develops case studies to support the eventual life cycle model. This article is one of those case studies. Companion case studies range across software, hardware, and firmware agile SE processes, and include (Dove, Schindel, Scrapper 2016, Dove, Schindel, Hartney 2017, Dove, Schindel 2017).

This case study is based on the three-day workshop held August 26-28, 2015, that analyzed a six-year-mature agile systems engineering (SE) process employed by Northrop Grumman's Global Combat Support System – Joint (GCSS-J) group in Herndon, Virginia. The GCSS-J process, supporting the DISA GCSS-J PMO, sustained and evolved functional capability for a military-critical centralized systems-of-systems web-hub that enables 12 independent military user communities access to 22 independent data sources, depicted in Figure 1.

Notable process concepts that will be discussed include:

- Intimate stakeholder involvement in the SE process.
- Asynchronous and simultaneous life cycle stage activity, in never-ending system growth and evolution.
- Hybrid Scrum/Waterfall/Wave process model integration, in contract conformance.
- CMMI level 5 procedure discipline, providing seamless new-release operational stability.
- Awareness and mitigation of external environment evolution.
- Real-time optimal process-control model, for re-prioritizing development-increment activity and acting on feedback.

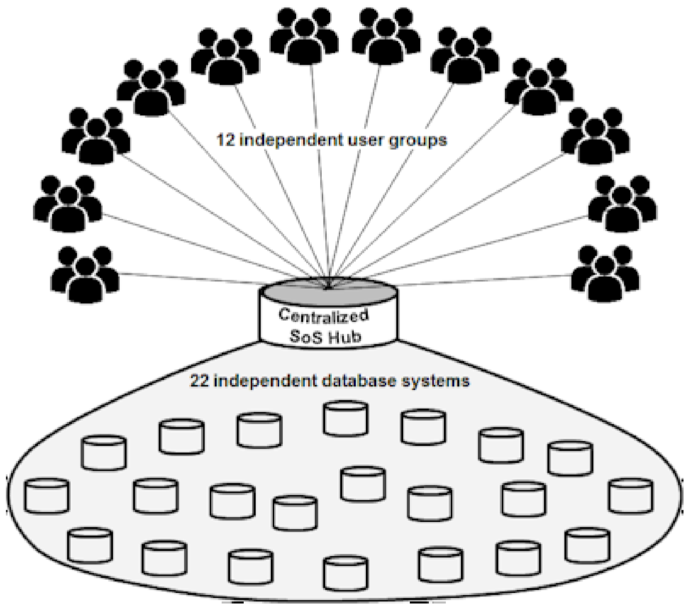


Figure 1. Centralized System-of-System hub as web-portal for accessing 22 independent databases.

The process to be described went live in 2009 to replace a waterfall process initiated in 1996 that took too long to evolve new capabilities. The process, depicted in Figure 2, decouples back-to-back Scrum-based six-month development increments from subsequent six-month phases of accreditation (government security test) and deployed operation. Development starts with a five-day planning session, and utilizes one or two 10-day Z sprints following the four 20-day development sprints. Z sprints are dedicated to defect correction and baseline stabilization. Four development sprints is typical, but three- and five-sprint releases occur on occasion. GCSS-J employs about 60 people on the program.

Process effectiveness had been demonstrated consistently over six years in seamless cutover of new capability every six months. The effectiveness of this process is marked by a US Government Accountability Office (GAO) congressional report, citing it as one of "seven investments [that] were successfully acquired in that they best achieved their respective cost, schedule, scope, and performance goals (GAO 2011)."

Overview

Systems engineering life cycle stages and processes formed the process analysis framework. It is neither expected nor necessary that workshop-analyzed systems engineering processes utilize the 15288 (ISO/IEC/IEEE 2015) process framework, but the 15288 standard provides an analysis framework that encompasses generic systems engineering activities, regardless of what they may be called.

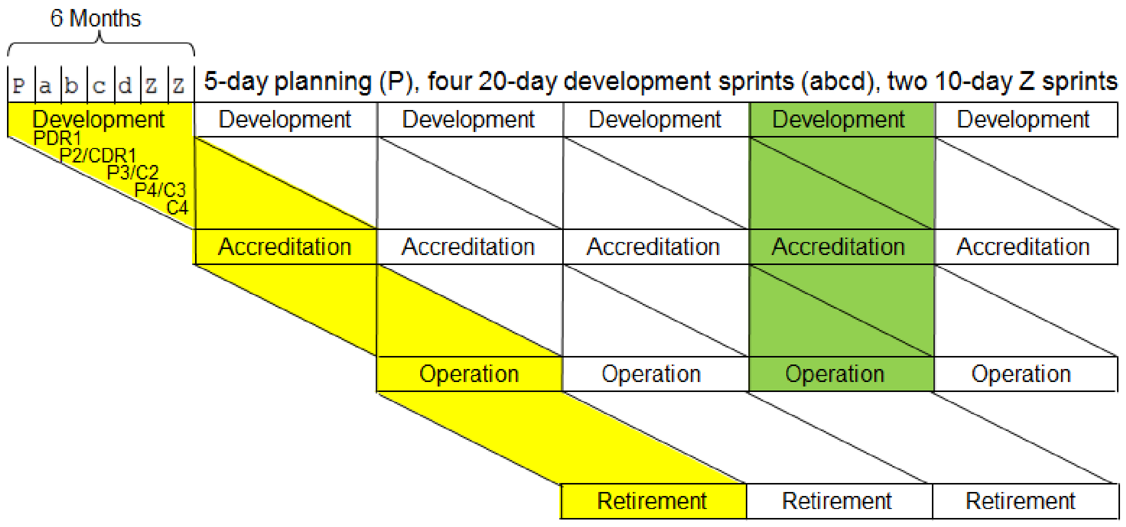


Figure 2. Yellow path depicts development-decoupled single-release waterfall progression. Green path depicts three active releases using common automated daily build and test for new development, operational patching, and security updates. Development sprints start with a ½-day meeting that combines a Preliminary Design Review (PDR) for the new sprint and a Critical Design Review (CDR) for the prior sprint.

Asynchronous and simultaneous life-cycle stage and process activity is a hallmark of effective agile systems engineering processes, as modeled in Figure 3. In Figure 2 the Development cycle functions implicitly as depicted in Figure 3, while the four simultaneous release-status levels function explicitly as simultaneous stages of Development, Production, Utilization, and Retirement.

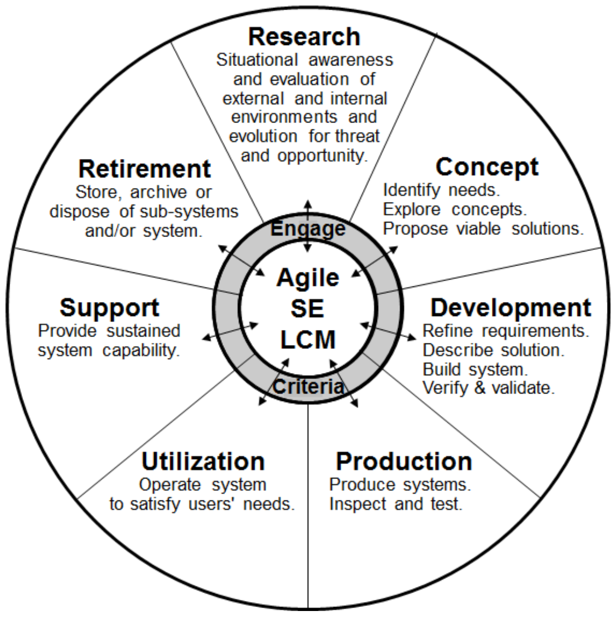

Research
Situational awareness and evaluation of external and internal environments and evolution for threat and opportunity.
Concept
Identify needs. Explore concepts. Propose viable solutions.
Development
Refine requirements. Describe solution. Build system. Verify & validate.
Production
Produce systems. Inspect and test.
Utilization
Operate system to satisfy users' needs.
Support
Provide sustained system capability.
Retirement
Store, archive or dispose of sub-systems and/or system.
Engage
Criteria
Agile SE LCM

Figure 3. Purposes for each Life Cycle Model stage, adapted from (ISO/IEC 2010, p 14), with added Research stage.

Systems and Software Engineering — Life Cycle Management — Part 1: Guide for Life Cycle Management (ISO/IEC 2010) recognizes six "commonly encountered" system life cycle stages. Figure 3 adds a seventh life-cycle stage, *Research*, as a critically necessary element of effective agile systems engineering life cycle models, as will be seen later.

Counter to the implication that a progression through stages is sequentially expected, (ISO/IEC 2010, p 32) clearly accommodates asynchronous and simultaneous activity in any and all stages with this clarification: "…one can jump from a stage to one that does not immediately follow it, or revert to a prior stage or stages that do not immediately precede it. … one applies, at any stage, the appropriate life cycle processes, in whatever sequence is appropriate to the project, and repeatedly or recursively if appropriate."

Agile systems engineering processes are justified and effective when it is expected that the engineering environment and activities will be subject to changes that affect the ongoing development effort throughout the project.

A framework for characterizing the general/high-level dynamic nature of the engineering environment provides guidance for what must be accommodated in five categories: capriciousness, uncertainty, risk, variation, and evolution (CURVE). This framework was introduced in (Dove and LaBarge 2014) as UURVE, changed here to CURVE by replacing Unpredictability with Capriciousness as it provides a more descriptive acronym.

GCSS-J characterized their CURVE environment as follows:

Capriciousness/Unpredictability (unknowable situations):

- External data sources change their services at will.
- COTS (Common Off The Shelf) software upgrades deprecate existing interfaces.

Uncertainty (randomness with unknowable probabilities):

- Software and/or hardware may go end-of-life at any point.

Risk (randomness with knowable probabilities):

- May not be able to meet 15-day schedule for delivery of security fixes.

Variation (knowable variables and variance ranges):

- Number of security vulnerabilities to address varies greatly week-to-week.
- Development man-hours available for capability evolution in competition with higher priority patches and security updates.

Evolution (gradual successive developments):

- As technology changes, the program must port existing capability to new technology.

Enabling, Facilitating, and Sustaining Agility

The discovery and description of a common Agile Architecture Pattern (AAP) for systems and processes that successfully deal with CURVE operational environments is detailed in (Dove and LaBarge 2014). In process-transformational activity, the AAP graphic is useful as a design tool for the agility-enabling Concept of Operations (ConOps), depicting what is intended to enable and facilitate agility in the systems engineering process. Thereafter it is useful as an active reflection of, and memory for, ConOps evolution. The AAP is a framework for customer and management communication, for training new team members, for capturing lessons learned, and for maintaining a current central understanding of the process' key operational concepts as they evolve. In presentations, it serves well as a single-slide road map for the systems-engineering ConOps.

For purposes of describing the relevant systems engineering process issues unambiguously, we recognize three distinct systems of interest, distinguished as Systems 1, 2, and 3. System 1 is the target system under development, a six-month-cycle evolution of a web portal hub connecting 22 independent data bases. System 2 includes the basic systems engineering development and maintenance processes, along with their operational domain that produces System 1. System 3 is the process improvement system, called the system of innovation that learns, configures, and matures System 2.

Figure 4 depicts the GCSS-J systems engineering process (System 2) as an instance of the AAP. Briefly, the architecture contains three principle elements: a pool of resources that can be configured to address the process-activity of the moment, a passive enabling infrastructure that establishes common rules for readily interconnecting these resources, and an active facilitating infrastructure with responsibilities for sustaining the agility of the SE process by maintaining and evolving the resources, the internal and external environmental awareness, the interconnection standards, and the agility-enabling ConOps.

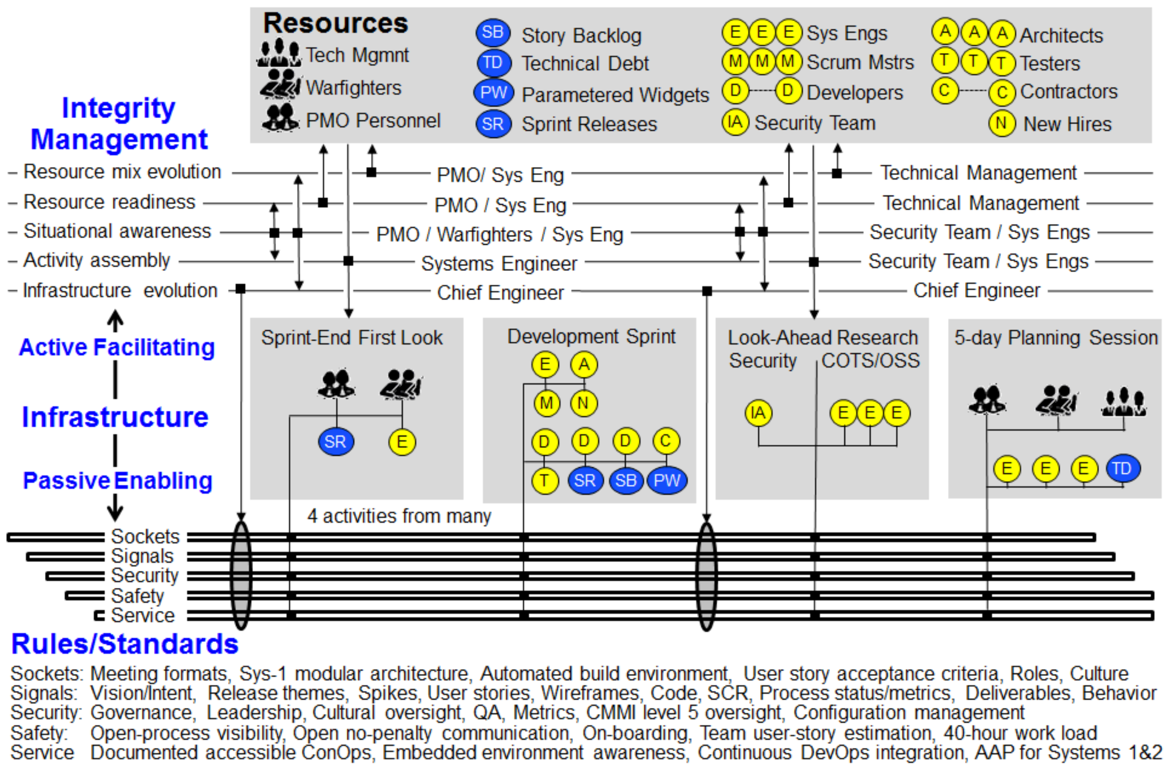


Figure 4. Agile-process architecture depicting four process-activity configurations assembled from available resources in conformance with the interconnection rules and standards of the passive enabling infrastructure.

The AAP instance of the GCSS-J process in Figure 4 serves to frame the discussion of the *key* process elements and their relationships. The process architecture is structured to configure a variety of process activities with personnel and other resources as and when needs arise. Activities draw upon pools of available resources, assembled into a variety of configurations upon need, interfacing with each other according to resource-interconnect rules and standards. The four activities depicted in Figure 4 relate to Figure 3 as instances of Production, Development, Research, and Concept stages.

The principle intent of this section is to discuss the passive enabling infrastructure and the active facilitating infrastructure, but necessarily starts with a description of the resources.

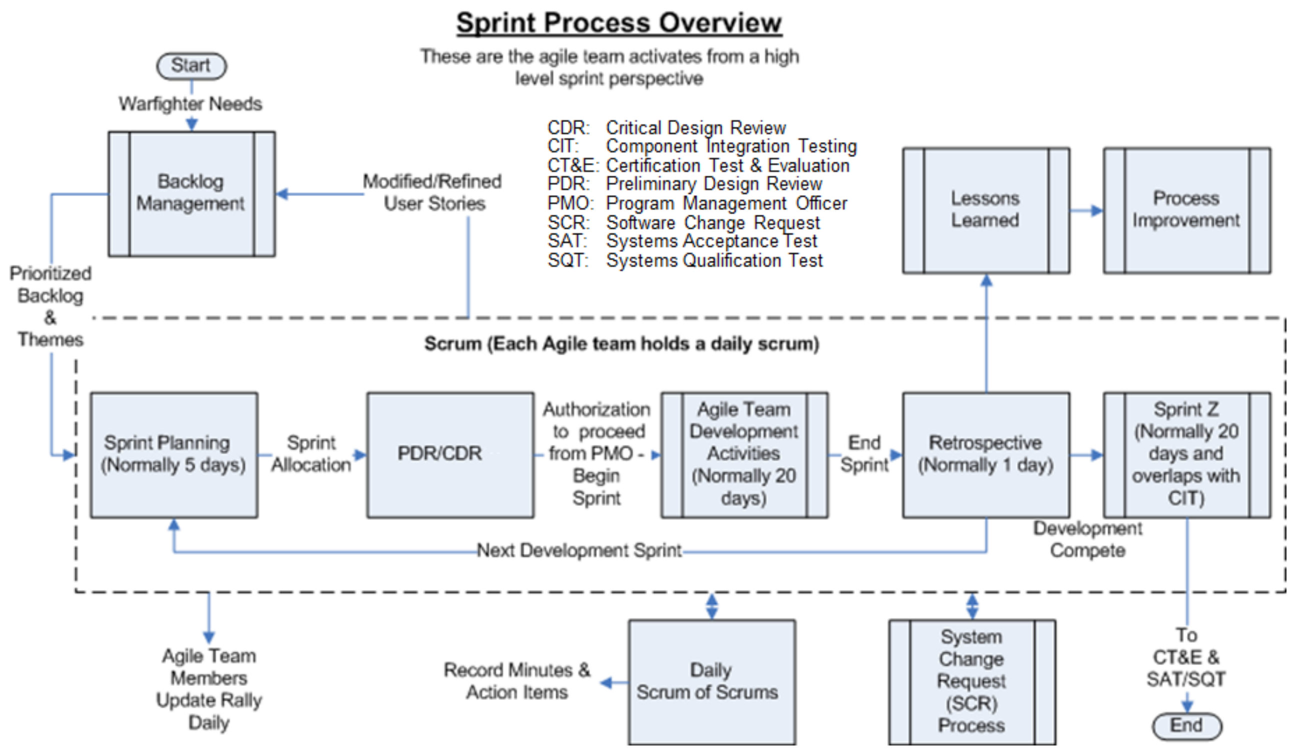
The AAP calls out the resources that are employed in assembling process-activity configurations:

- Scrum Team Resources – Typically three Scrum teams were active on three simultaneous sprints within the same development release, with teams configured and reconfigured from appropriate resources as needed.
    - Systems Engineers – one per Scrum team, served as the liaison to the PMO personnel for requirements, worked directly with the Scrum team to oversee the implementation of the requirements, and supported the test team to ensure the requirements are tested accurately and completely.
    - Architects – one per Scrum team, defined the software architecture for their portion of the system, met weekly with the other architects and the Chief Engineer to enhance the overall system architecture, peer reviewed the code developed by the team, and supported the scrum team developers as a mentor.
    - Scrum Masters – one per Scrum team, lead the scrum team on a daily basis, managed the personnel on the scrum team, worked with the other scrum masters to coordinate and share resources, and represented the Scrum team at the daily Scrum-of-Scrum meetings.
    - Developers – five code developers per Scrum team.
    - Testers – one per Scrum team that did daily testing of the prior night's code build.
    - Contractors – The GCSS-J program utilized personnel from both the prime contractor and five sub-contractors. GCSS-J operated in a badgeless environment with no separation of duties based on prime and sub status. This approach to providing Scrum team resources improved the program's ability to attract and keep top talent no matter what the badge said.
    - New Hires – Regardless of the program, some Scrum team resource attrition will occur. An on-boarding process was in-place to provide guidance on the GCSS-J processes, with automation to expedite the construction of a new employee's development environment, and mentoring to quickly bring a new employee up to speed.
    - Security Team – Information assurance specialists.
- PMO Personnel (Program Management Office product owners) – Established and re-established release priorities.
- Technical Engineering Management – Architectural oversight and program technical direction.
- War Fighters (Users) – Did first-look sprint testing, contributed to release-requirements development, and participated in next-release theme development.
- Story Backlog – user stories developed for each release and parceled out to development sprints according to evolving priorities.
- Technical Debt – User stories not completed for a release could be carried over to subsequent releases as technical debt. Stories were built for all tasks of any kind.
- Parameterized Widgets [1] – Parameterized GUI capability code enables reusable employment with similar but varying user-story needs.

[1] Widget: "a generic term for the part of a GUI [graphical user interface] that allows the user to interface with the application and operating system. Widgets display information and invite the user to act in a number of ways." Vangie Beal, www.webopedia.com/TERM/W/widget.html

- Sprint Releases – Typically four sprint releases occur sequentially in a six-month development increment, each augmenting and improving the prior release.

Figure 5 shows an overview of the team activities with the development Sprint at its center. Notably, PDR and CDR were embedded within the Sprint, where PDR reviewed the upcoming Sprint and CDR reviewed the prior Sprint, both in a single meeting day.



Sprint Process Overview
These are the agile team activates from a high level sprint perspective
Start
Warfighter Needs
Backlog Management
Modified/Refined User Stories
CDR: Critical Design Review
CIT: Component Integration Testing
CT&E: Certification Test & Evaluation
PDR: Preliminary Design Review
PMO: Program Management Officer
SCR: Software Change Request
SAT: Systems Acceptance Test
SQT: Systems Qualification Test
Lessons Learned
Process Improvement
Prioritized Backlog & Themes
Scrum (Each Agile team holds a daily scrum)
Sprint Planning (Normally 5 days)
Sprint Allocation
PDR/CDR
Authorization to proceed from PMO - Begin Sprint
Agile Team Development Activities (Normally 20 days)
End Sprint
Retrospective (Normally 1 day)
Sprint Z (Normally 20 days and overlaps with CIT)
Next Development Sprint
Development Compete
Agile Team Members Update Rally Daily
Record Minutes & Action Items
Daily Scrum of Scrums
System Change Request (SCR) Process
To CT&E & SAT/SQT
End

Figure 5. Sprint process overview with supporting activities

Process Enabling and Facilitating Infrastructure

Infrastructure consists of passive and active sections. The passive section includes the resource interconnection standards that enable effective process-activity assembly. The active section designates responsibilities for maintaining and evolving agile process capability that facilitate sustainable process agility.

Passive Enabling Infrastructure

Figure 4 at the top shows the principle SE-process resources that can be assembled into process-activity configurations for specific situations. The ability to drag-and-drop these resources into plug-and-play configurations is enabled by the passive infrastructure, so called because it encompasses the fairly stable rules that enable effective resource interconnection.

In the text below System 1 refers to the product being developed (SoS web portal), and System 2 refers to the process doing the development (GCSS-J SE process).

Sockets – process physical interconnects:

- Meeting formats were documented as part of the process Concept of Operations (ConOps).

- System 1 modular architecture – web-page based System 1 provided encapsulated page-modularity with URL links providing the passive infrastructure interconnect standard. Enabled reconfiguration/augmentation agility.
- Automated build environment for nightly development additions and operational patches to all three release instances: development, accreditation, operation.
- User story acceptance criteria provided a testable list of criteria that defined what a user story must implement.
- Roles describing how people fill positions documented in the process ConOps.
- Culture of full-team mission-focus engagement.

Signals – process data interconnects:

- Vision was established by a mission set of requirements shaped and enhanced by the customer and customer's governing body. Intent was defined in the set of Epic user stories – high level user stories that describe a significant capability.
- Spike user stories are new ideas or changes to the overall design identified and validated through R&D. These were executed separately from the development code baseline.
- Release themes developed in the release planning sessions.
- User stories produced for all development tasks.
- Wireframes (GUI display architecture) produced by designated developers.
- Code produced daily.
- Software Change Request (SCR) produced by Scrum teams and by security team, managed through the approval process by a systems engineer.
- Process status was tracked daily at the Scrum-of-Scrums and provided to the customer at the weekly customer meetings.
- Deliverables: PDR/CDR, and end-of-sprint developed code and documentation.
- Team behavior was exhibited by, and evident to, all team members.

Security – process trust interconnects:

- Governance was managed jointly by the customer Program Management Office and the contractor Program Management Team. The contract statement of work identified contract scope, required deliverables, etc.
- Leadership was provided by the program manager and the deputy program manager.
- Cultural oversight was organically supported by the team in place, and stability was maintained by a low turn-over rate and a cadre of long-term team members.
- QA tracked software quality, document quality, and performs independent peer review of documents for delivery.
- Metrics were collected on SLOC (Source Lines of Code), peer reviews, software defects, document defects, etc., and trends over time are plotted for process improvement.
- CMMI level 5 oversight was performed by a trained practitioner on the program, with periodic reviews by both the contractor and the certifying agency.
- The GCSS-J program had a dedicated Configuration Management resource. Responsibilities include validating any changes to the COTS/OSS (Open Source Software) products used within the program, tracking licensing, and validating and delivering required contract deliverables.

Safety – of SE-process users, process, process environment:

- Open process visibility was documented in a shared wiki and also included in the documentation delivered to the customer. QA checked that processes were being observed, and audits were conducted by the corporation to validate program adherence to the processes.
- Open no-penalty communications were facilitated by daily team scrums, daily management scrums, weekly meetings with the customer, and ad-hoc (occurring almost daily) phone calls and email exchanges at all levels of the customer-contractor engagement. Communication at all levels was strongly encouraged with policies in place to facilitate open communication that do not penalize individuals for bringing up concerns.
- New hire on-boarding mentoring by architects, with reduced new-hire productivity expectation for their first sprint.
- User-story estimation was a team task, eliminating individual risk.
- 40-hour/week was scheduled for development and testing personnel to support a sustainable work load. Developer time for sprint work was allocated at less than 8 hours/day to allow time for meetings and other required activities.

Service – the SE-process ConOps:

- The Concept of Operations for the hybrid scrum/waterfall/wave process executed in overlapping decoupled six-month development, accreditation, and operational cycles was documented in the program wiki, with diagrams showing the flow of information through the process, the responsible parties for each step in the process, and all process operational aspects and interactions. CMMI Level-5 certified conformance was enforced.
- Embedded awareness of potential external disruptions from likely security and COTS-obsolescence events by assigned responsibilities; and embedded awareness of internal operational issues and opportunities was openly discussed in full-day sprint retrospectives.
- Continuous DevOps[2] integration method with automated build and test tools.
- A collective culture of engagement was embraced by the full team, monitored and enforced by management oversight.
- Agile Architecture Pattern (AAP) employed for both System 1 and Systems 2. System 1 is the systems engineering process, with resources reconfigurable to accommodate situational activity needs. System 2 is the continuously developed/evolved SoS hub portal, with loosely-coupled web pages and reusable parameterized widgets.

Active Facilitating Infrastructure

The active infrastructure is what sustains the agility of the SE process. In order for new activity-configurations to be facilitated when needed, five responsibilities are required: the roster of available resources must evolve to be always what is needed, the resources that are available must always be in deployable condition, the assembly of new activity configurations must be effectively accomplished, and both the passive and active infrastructures must evolve in anticipation and/or satisfaction of new needs. These five responsibilities are outlined in standard

[2] DevOps: A general concept that integrates certain development and operations activities rather then have them occur by independent teams with independent methods. In this article, DevOps refers to GCSS-J use of a single process employing automated tools for creating and testing developer code and patches made by operations.

role descriptions, assigned to appropriate personnel, and embedded within the process to ensure that effective process-activity is possible at unpredictable times.

- **Resource mix evolution –** ensures that existing resources are upgraded, new resources are added, and inadequate resources are removed, in time to satisfy needs. This responsibility was triggered by situational awareness, and dispatched as shown in Figure 3 activities.
- **Resource readiness –** ensures that sufficient resources are ready for deployment at unpredictable times. This responsibility was ongoing, and dispatched as shown in Figure 3 activities.
- **Situational awareness –** monitors, evaluates, and anticipates the operational environment in relationship to situational response capability. This responsibility was ongoing, and dispatched according to the situation of interest, typically by Architects, Systems Engineering Team, Information Security Team, Technical Management.
- **Activity assembly –** assembles process-activity configurations. This responsibility was triggered by situational awareness, as and when needed, and dispatched according to the activity of interest, typically by Architects, Systems Engineering Team, or Information Security Team.
- **Infrastructure evolution –** evolves the passive and active infrastructures as new rules and roles become appropriate to enable response to evolving needs. This responsibility was triggered by situational awareness, and dispatched by the Chief Engineer.

Key Operational Process Aspects

Six elements of the process enabling (passive) and facilitating (active) infrastructures are central to the effectiveness of the GCSS-J SE process Concept of Operations, which warrant further discussion.

Passive Infrastructure: Intimate stakeholder involvement in the SE process.

- The customer was intimately involved with the process operation, establishing the intended release features, allocating time for COTS obsolescence mitigation, allocating effort to backlog tasks, attending daily scrum meetings, and directing sprint backlog re-prioritization as high-priority security bulletins arise. This amounts to explicit customer risk allocation acceptance and task allocation prioritization.
- Both the customer and actual users (warfighters) provided the principal end-of-sprint first-look testing feedback.

Passive Infrastructure: Asynchronous and simultaneous life cycle stage activity.

- The SoS hub was in perpetual feature evolution, and existed in three stages simultaneously (development, accreditation, operation) that all received security updates immediately when necessary. Research on external awareness for look-ahead on pending COTS and OSS (Open Source Software) obsolescence and likely security bulletins happened continuously in parallel with all other life cycle stages. Internal awareness was integrated with development sprints in both daily scrum meetings and sprint retrospectives. DevOps integrated operational utilization and support stage activities with development and production stage activities, sharing the same automated build and test tools and processes. Retirement occured simultaneously with development as COTS software and OSS were replaced. The concept stage was frequently invoked as spikes (infrastructure-migration sprints) and were explored in parallel with development sprints for integration at the start of a future development sprint.

Passive Infrastructure: Hybrid Scrum/Waterfall/Wave process integration.

- The choice of a hybrid Scrum/Waterfall/Wave model is discussed later. Basically, this hybrid model used Scrum for development, Waterfall for accreditation testing at completion of development, and Wave to decouple simultaneous development, accreditation testing, and operational releases. This hybrid process was accommodated by contract provisions that recognized the advantage obtained with on-time operational releases every six months, and integrated sprint-staggered PDR and CDR at the start of development sprints.
- Each development sprint provided a first-look (off-line) release for testing by the customer and users, somewhat equivalent to Scrum's desired deliverable at end of every sprint.

Passive Infrastructure: CMMI Level 5 procedure discipline.

- The process had been a CMMI Level 5 certified program since 2009, which likely accounted for its demonstrated seamless operational-release stability. Operational releases had historically exhibited virtually no critical bugs. First-look feedback and accreditation testing contributed to this as well.

Active Infrastructure: Awareness and mitigation of external and internal environment evolution.

- Information security team did ongoing look ahead for pending security issues in advance of immediate-attention bulletins
- Systems engineering team did ongoing look ahead for pending COTS and OSS obsolescence.
- Chief Engineer and Architects did look ahead for early warning on changes in SoS data bases (often there is no warning). Look ahead involved communication with DoD partner programs, monitoring of DoD and DISA standards efforts, and observation of industry best-practices.
- Systems engineering team did look ahead on infrastructure evolution needs with spike-sprint development and seamless start-of-sprint cut over.
- Everyone became aware of process issues discussed openly in a full day reflection/retrospective after every sprint. These provided frequent checks of where things were and if they were on course, a key principle for the program gleaned from both the Scrum principles and CMMI Level-5. Issues were raised earlier and problems were identified earlier, allowing course corrections before the change created significant impact to program progress.
- The customer convened warfighter workshops to obtain input and feedback directly from the users, with members of the GCSS-J systems engineering team present to obtain awareness of user's needs directly.
- Management actively monitored conformance to ingrained cultural-principles: automate whenever possible, build security in from the beginning, follow the program standards, test early and often, etc., with mitigation as appropriate.

Active Infrastructure: Real-time optimal process control model.

- Discussed at more length in a later section are patterns of real-time optimal process control for re-prioritizing development-increment activity and acting on feedback. An effective agile process is driven in real time by awareness gleaned from external and internal active research, rather than simply following a standard procedure.

On Choosing the Hybrid Scrum/Waterfall/Wave Model

As background, the program initially follow a Waterfall model, taking 1.5 years before a new capability got operationally deployed. This often resulted in program requirements changing by the time code was released for operation. Requirements had no clear prioritization as to what to work first. The requirements for each release grew beyond available development resources, due to the long release cycle, putting the schedule at risk. Applying patches and security updates to the operational system took a long time and was error prone.

The principle reasons indicating that an agile SE approach was needed were outlined in this article's CURVE elements earlier. The decision to use an agile SE approach was influenced by the need for more frequent operational deployments, a clear way to prioritize requirements, and a process-facilitating way to re-prioritize work-in-process when security imperatives, COTS end-of-life, and data-base changes required immediate attention.

The program was software-focused with releases needed in six-month increments, and the issues were centrally program and process management. Spiral[3], more suited to mixed Hardware/Software programs with longer release cycles, did not appear appropriate. SAFe[4] wasn't on the radar when the choice was made and subsequently went live in 2009, and neither SAFe's team-scalability nor program-portfolio management were issues.

Scrum[5] appeared to have what was needed, and was chosen as the core of the process model for six-month software development cycles. But the program required a partial waterfall approach, in that CDR and PDR development-in-process gates were required by contract; and accreditation required another six-month cycle following the development cycle. Incorporating CDRs and PDRs "naturally" in the agile development cycle was an evolutionary learning experience discussed later in the Lessons Learned section. The decoupling of development from accreditation permitted development cycles to follow each other immediately, with a 3-month accreditation cycle followed by a 6-month operational cycle – in a classic Wave[6] approach.

Waterfall is inherent in the sequential separation of development, accreditation testing, and operational phases; but was mitigated with a Wave model approach that decoupled these three phases to operate in parallel on staggered releases.

Lessons Learned

An agile SE process is necessarily and inherently a continuous learning-based process, and accommodates frequent adjustment to the SE process based on lessons learned.

- When first implemented the GCSS-J agile SE process started with the traditional two-reviews (PDR and CDR) at two-days each for each of four sprints. A total of eight days for a four-sprint

[3] Boehm, Barry. 1988. A Spiral Model for Software Development and Enhancement. IEEE Computer. May.
[4] SAFe refers to the Scaled Agile Framework developed by and described in Leffingwell, Dean. 2007. *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley Professional.
[5] Schwaber, Ken and Jeff Sutherland. 2013. The Scrum Guide. www.scrum.org.
[6] Dahmann, Judith, Jo Ann Lane, George Rebovich, Jr. and Kristen J. Baldwin. 2011. An Implementers View of Systems Engineering for Systems of Systems. IEEE International Systems Conference, Montreal, Canada, 4-7 April.

release. These reviews evolved into half day reviews that combined both the PDR for the current sprint with the CDR for the prior sprint in the half day combined-review session. The combined PDR/CDR reviews were initially held during the sprint planning week, but that was found to be disruptive and didn't work well, so they were moved to occur on the first day of a sprint. There was no overall PDR/CDR for an operational release, just ones for each sprint.

- The Functional Requirements Working Group (FRWG), which includes both the customer and GCSS-J personnel, was getting bogged down in design discussions rather than being focused on reviewing and approving the user stories to be worked. A White Board Session, suggested and so-named by the customer, was added where the customer and GCSS-J personnel do a lot of the pre-planning and understanding of what the customer wants at a detailed level. This allowed attendees to focus each meeting on its respective goal.
- The systems engineering team discovered that some COTS and OSS software embedded in the GCSS-J system were no longer supported, increasing security exposure to an unacceptable level. A monthly look-ahead on the versions of COTS and OSS being used was instituted, allowed GCSS-J to incorporate a plan for upgrades into the rhythm of the program.
- The information security team watched the time allowed by the customer for responding to high-priority security bulletins shrink over time, and the frequency increase. This prompted a look-ahead for newly discovered vulnerabilities in available security discussions that would likely become security-bulletin material.
- The program started with a lot of churn in requirements, with changes being made often and too late (once coding had already started). GCSS-J used the development hour metrics to show the customer the effects of that churn: less capability for the same hours. This led to an agreement that user stories would be solidified before the start of the sprint and no/minimal changes were allowed during the sprint. If requirements changed, the changes would be implemented in a later user story.
- Correcting each defect and implementing each new feature required a good deal of development time because the changes were scattered throughout the code base. GCSS-J negotiated bundled fixes instead of one-off fixes. Bundled fixes were several changes/corrections that applied to a particular widget or section of code. By making all of the changes to one area, the developer cold fix several things more efficiently, and the testing could also focus on that one area. This increased the number of changes that could be accomplished within the allotted time. However, this did mean that some changes might get delayed if they were singular to a widget or code section and not a high priority. Those changes could be delayed until additional changes to the area of the code base were required. High priority changes were still made, even if they were isolated.
- Widgets had been developed for a number of years, but initially each scrum team did it their own way, with no standards in place. Eventually the whole user interface was overhauled so everything had a consistent look and feel.
- User interface components, referred to as widgets, were constructed to accept parameterized data as input. The widgets could be reused in appropriate places simply by changing the parameters passed in. This had a positive effect by allowing code reuse, reducing debug time (fix it in one place and several bugs are removed), and reduced the overall number of failed test cases because existing mature code was used instead of creating yet another widget.

- Automation for code builds and deployment was very successful, but systems operations and security patching were still handled separately. The systems operations team agreed to merge their work into the software development automation solution. The integration was successful and the program then moved into a DevOps paradigm. All changes to the system, whether software development, operational system patches, or security fixes, flowed through the same create-build-deploy-test cycle. This streamlined the process to the point that the program was able to achieve the short release cycle required by the customer.

Metrics

- Required by CMMI Level 5 and corporate oversight, the program collected a lot of metrics. Metrics were tracked for every sprint: technical debt, level of effort actuals compared to estimates, number of defects found in testing per user story, defect change over time, how much SLOC (source lines of code) and how much it changed each sprint. Trends were watched over time. A spike or bad trend caused a root cause analysis and a task team was spun off to find out why. Defect rate was watched closely to indicate too much is being tried when that goes up. This instigated a debate: the defect rate was extraordinarily low, and considered a possible sign of too much caution that might permit pushing a little harder with results still good enough. If a spike occurred in SLOC the customer and the GCSS-J personnel wanted to know why, so analysis was done ahead of time in anticipation of questions. Velocity was tracked. An important part of the development scrum was monitoring velocity and the burn down rate. Velocity was not measured in story points but rather in actual hours, at customer insistence. This gave the customer incredible visibility into exactly what was being done, because they knew how many people and how many hours every sprint. That varied, so every bit of maintenance wanting to be done was coordinated with the customer, who saw how maintenance takes away from new functionality. This resulted in a great deal of trust, but also micro-management and overhead. Everything to be done was expressed as a user story, with the customer deciding the priorities.
- Statistics were collected on code defects, re-deliveries, document red lines, and operational up and down time. These predicted how many defects could be expected, how often, and the lines of code that might cause defects.

Pattern-Based Model View of Key Operational Aspects

The ASELCM Pattern

The ASELCM Pattern is a formal MBSE reference model describing the framework of system life cycle management from an agility perspective, providing a non-prescriptive reference emphasizing the principles of agility, for analysis purposes. It is described further in (Schindel and Dove, 2016). Figure 6 is one view of that model, summarizing three key system boundaries, configured here for the GCSS-J case study:

System 1: The Target System, subject of innovation over managed life cycles in the GCSS-J system being developed, deployed, supported.

System 2: The Target System Life Cycle Domain System, including the entire external environment of the Target System—everything with which it directly interacts, particularly its operational environment and all systems that manage the life cycle of the Target System. For

GCSS-J this includes all the external data systems integrated by the focal DoD information system, as well as all the (agile or other) development, deployment, support, security, accounting, performance, and configuration management systems that manage the (System 1) Information System.

System 3: The System of Innovation, which includes System 1 and 2 along with the systems

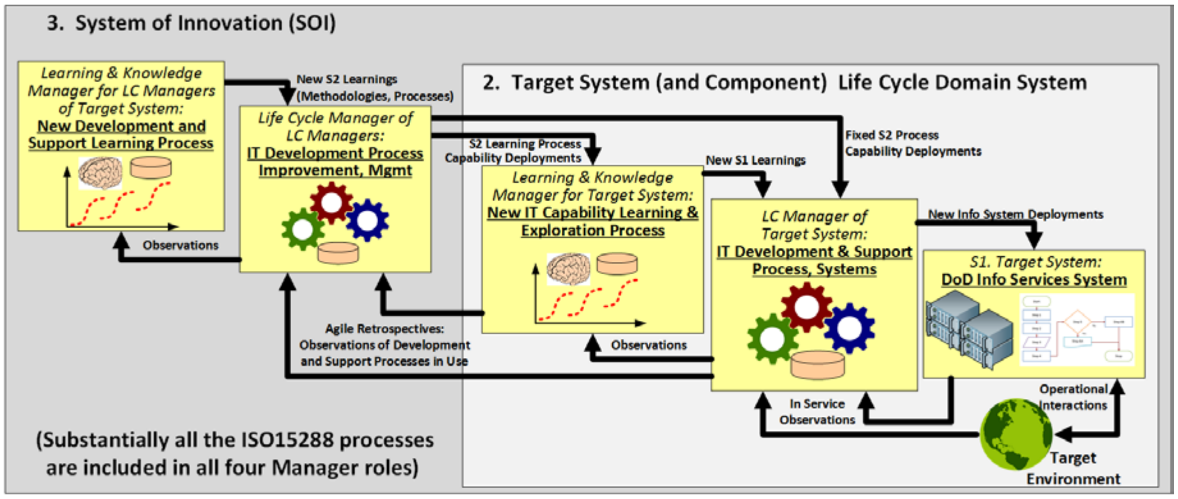

3. System of Innovation (SOI)
Learning & Knowledge Manager for LC Managers of Target System:
New Development and Support Learning Process
New S2 Learnings (Methodologies, Processes)
Life Cycle Manager of LC Managers:
IT Development Process Improvement, Mgmt
Observations
2. Target System (and Component) Life Cycle Domain System
S2 Learning Process Capability Deployments
Fixed S2 Process Capability Deployments
New S1 Learnings
Learning & Knowledge Manager for Target System:
New IT Capability Learning & Exploration Process
LC Manager of Target System:
IT Development & Support Process, Systems
New Info System Deployments
S1. Target System:
DoD Info Services System
Agile Retrospectives: Observations of Development and Support Processes in Use
Observations
Operational Interactions
In Service Observations
Target Environment
(Substantially all the ISO15288 processes are included in all four Manager roles)

Figure 6. ASELCM pattern system reference boundaries, configured for GCSS-J

managing (improving, deploying, supporting) the life cycle of System 2. In the GCSS-J this includes the systems that define, observe, analyze (as in agile Process Retrospective) improve and support the processes of development, deployment, service, or other managers of System 1.

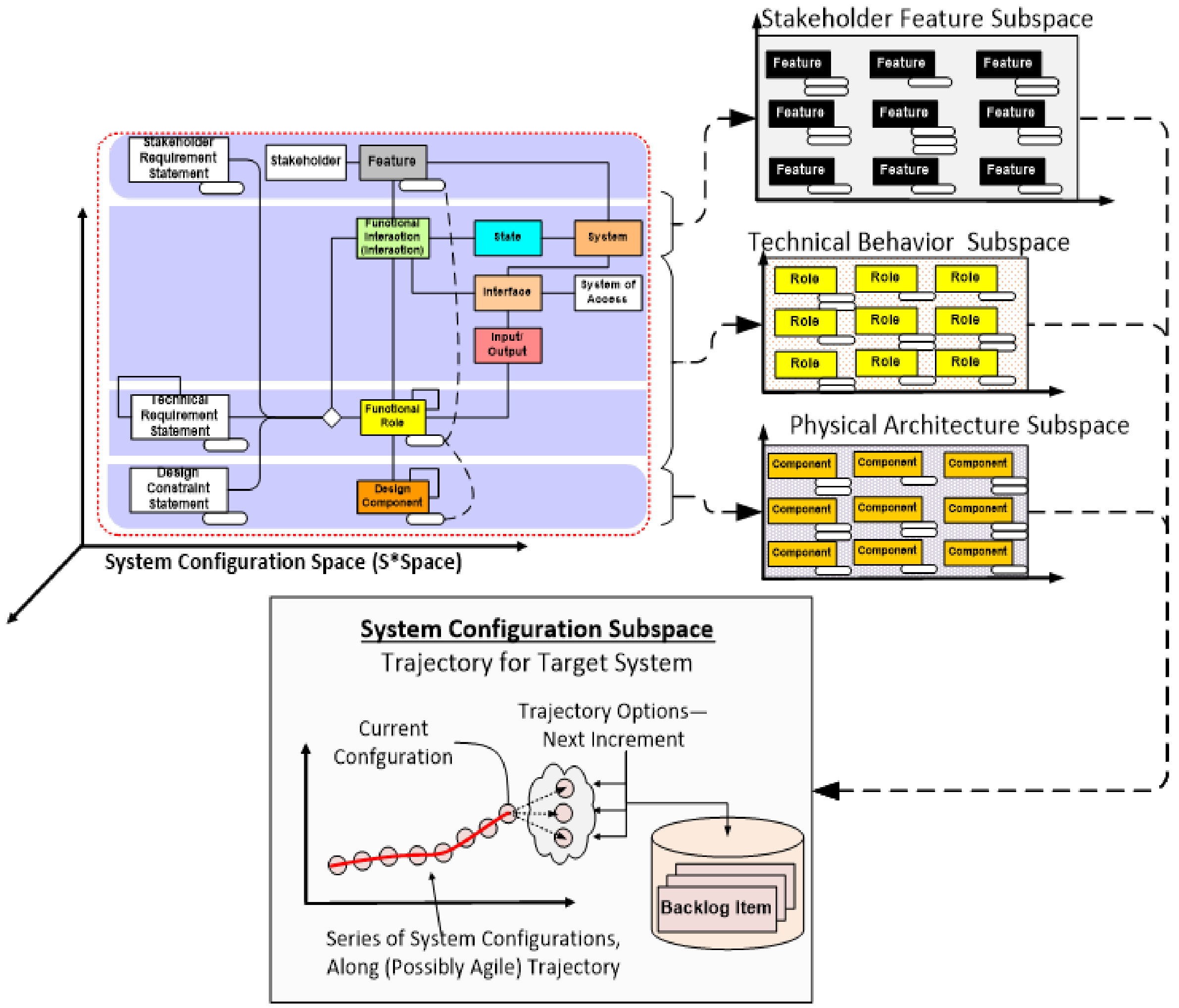
Trajectories in System 1 Configuration Space; Optimal Control Estimation

For purposes of this discussion, the "System Configuration" of the (System 1) Target Information System includes its evolving stakeholder fitness space (needs, values), details of System 1 technical behavior, physical equipment, software design, or other content, and whether in the form of written descriptions, or information in the minds of developers, or described by agile Stories, traditional Specifications, system Models, hallway conversations, or internalized expertise – all of this is considered to be the System Configuration of the Target System, whether formally tracked or not. Figure 7 summarizes this, emphasizing subspaces of Stakeholder Features, externally visible Technical Behavior, and Physical Architecture.

The ASELCM Pattern in Figure 6 is a reference model describing pursuit of agile trajectories through system configuration space, but this analysis model does not assume that a given environment (such as this case studied) uses model-based development methods. However, the analysis framework behind the ASELCM Pattern provides a model-based foundation that can be used as a basis for analysis of agile principles and their appearance in any development or life cycle management process, model-based or not.

For example, Figure 5 and Figure 7 shows that sprints involve the prioritized selection of Backlog items, resulting in incremental changes in the target information system Stakeholder Features (value, fitness), as well as Technical Behavior and Physical Architecture. The choice and sequence of these selections effectively determines a trajectory through System Configuration Space of the

Target System. The System of Innovation Pattern models this as a problem of optimal control in a noisy, uncertain environment. In the GCSS-J, as in most contemporary agile development processes, heuristics such as WSJF (weighted shortest job first) are used to optimize the rate of progress (the "job" part) through Stakeholder Fitness Space (the "weighted" part) with respect to time, effort, and risk (the "shortest" part).



Stakeholder Feature Subspace
Feature
Technical Behavior Subspace
Role
Physical Architecture Subspace
Component
Stakeholder Requirement Statement
Stakeholder
Feature
Functional Interaction (Interaction)
State
System
Interface
System of Access
Input/ Output
Technical Requirement Statement
Functional Role
Design Constraint Statement
Design Component
System Configuration Space (S*Space)
System Configuration Subspace
Trajectory for Target System
Current Confguration
Trajectory Options— Next Increment
Backlog Item
Series of System Configurations, Along (Possibly Agile) Trajectory

Figure 7. GCSS-J agile trajectory in system configuration space and sub-spaces

Trajectory Uncertainty & Risk; Reducing, Trading, & Sharing Risk; Decisions

Figure 8 is the formal interactions model of the agile loops summarized by Figure 5. The emergent effect of these loops is to traverse System 1 configuration space shown in Figure 7, with increment directions of this trajectory set by the following interactions of Figure 8:

- Initiate Product Backlog
- Review Priority Items & Set Sprint Thematic Goal
- Forecast Sprint Content Items
- Analyze Future Item Requirements
- Split, Merge, Rescope Future Items

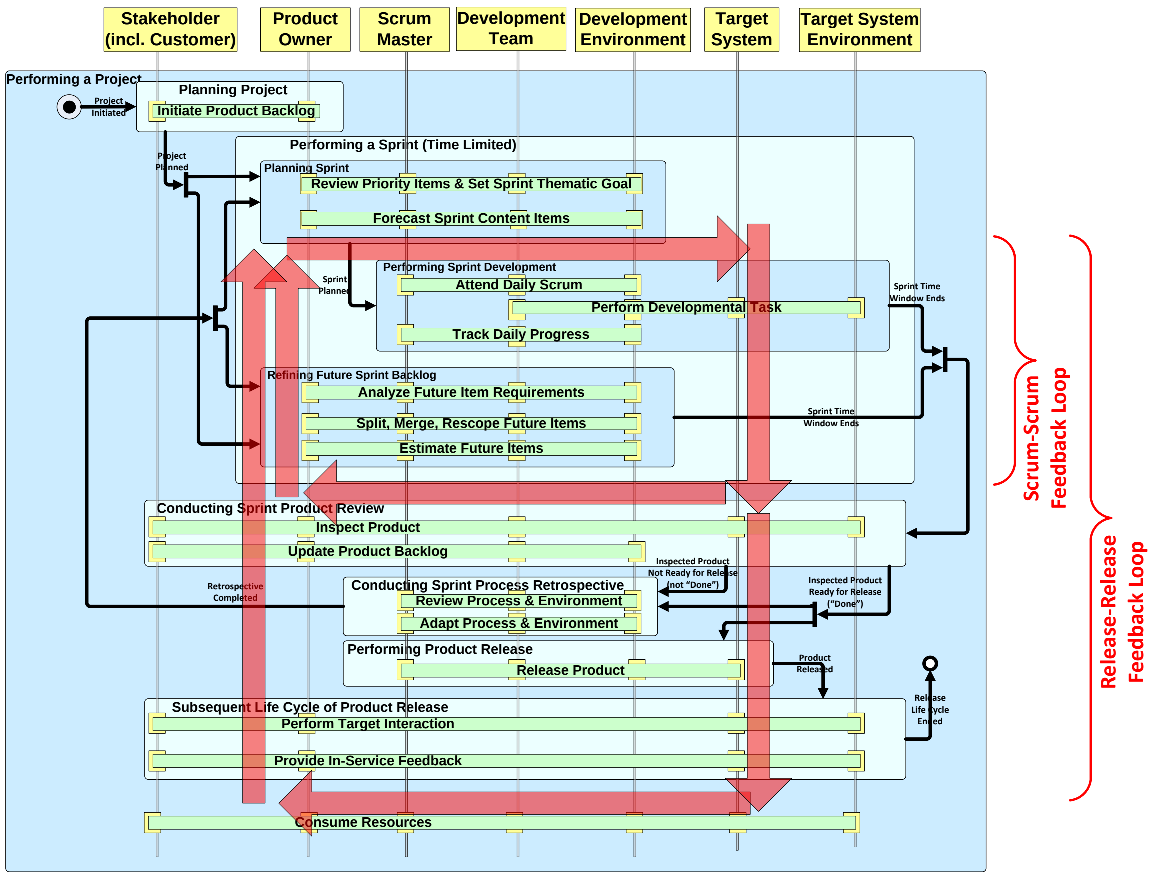

Stakeholder (incl. Customer)
Product Owner
Scrum Master
Development Team
Development Environment
Target System
Target System Environment
Performing a Project
Project Initiated
Planning Project
Initiate Product Backlog
Project Planned
Performing a Sprint (Time Limited)
Planning Sprint
Review Priority Items & Set Sprint Thematic Goal
Forecast Sprint Content Items
Sprint Planned
Performing Sprint Development
Attend Daily Scrum
Perform Developmental Task
Track Daily Progress
Sprint Time Window Ends
Refining Future Sprint Backlog
Analyze Future Item Requirements
Split, Merge, Rescope Future Items
Estimate Future Items
Sprint Time Window Ends
Conducting Sprint Product Review
Inspect Product
Update Product Backlog
Inspected Product Not Ready for Release (not "Done")
Inspected Product Ready for Release ("Done")
Retrospective Completed
Conducting Sprint Process Retrospective
Review Process & Environment
Adapt Process & Environment
Performing Product Release
Release Product
Product Released
Release Life Cycle Ended
Subsequent Life Cycle of Product Release
Perform Target Interaction
Provide In-Service Feedback
Consume Resources
Scrum-Scrum Feedback Loop
Release-Release Feedback Loop

Figure 8. Nested feedback loop processes traverse system configuration space

For GCSS-J these interactions were performed by human analysts, supported by basic information repository tools. These strategy interactions include decisions that involve stakeholder risk. For GCSS-J this risk was traded between entities by agreement to include the customer in not only the definition of Stories, but also the selection of those backlog items for inclusion in sprints. From optimal stochastic control theory, we know that effective ability to observe, estimate, and control are paramount to effective agility. The underlying sprint loop includes frequent measurement and feedback of current state, including in Stakeholder Feature Subspace, with frequent feedback control opportunities at the end of each sprint loop.

States, Modes, and Learning in System 2

The states in Figure 8 are sub-states of the Development state of Figure 3, and may occur concurrently with others, as suggested by Figure 2. During these processes, there are opportunities for learning, which is an emphasized characteristic of agile methods. Note that the system configuration trajectory progress of Figure 7 does not in itself imply any learning. Indeed, Figure 6 shows that previous learning could be the entire basis of that configuration progress, if the past learning is available for reflexive use as called upon by the situation (suggested by the "gears" portion of Figure 6). An example of this observed in the GCSS-J was the progressive accumulation of system test and DevOps methods that could be invoked as called upon by future needs.

Whether learning is accumulated informally by humans or in more explicit automated and configurable models, Figure 9 conveys that System 1 learning accumulates for use by System 2, as underlying patterns across all the same sub-spaces as those of Figure 7.

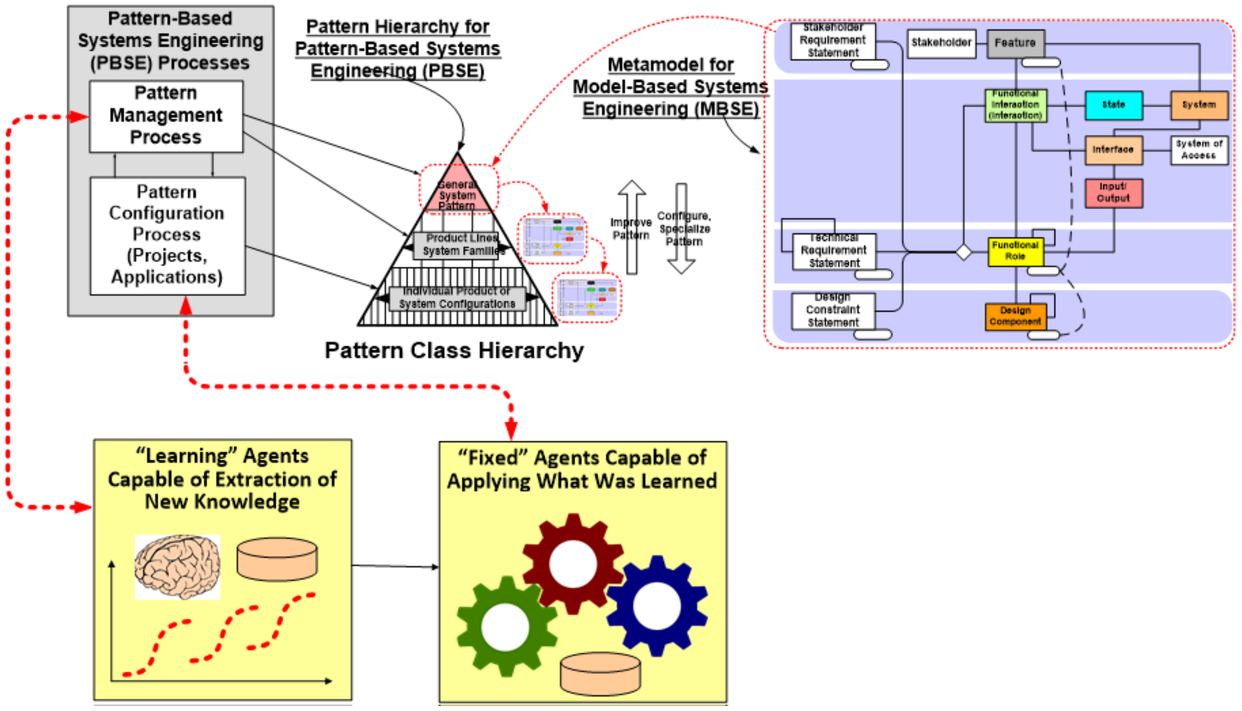

Pattern-Based Systems Engineering (PBSE) Processes
Pattern Management Process
Pattern Configuration Process (Projects, Applications)
Pattern Hierarchy for Pattern-Based Systems Engineering (PBSE)
Metamodel for Model-Based Systems Engineering (MBSE)
General System Pattern
Product Lines System Families
Individual Product or System Configurations
Improve Pattern
Configure, Specialize Pattern
Pattern Class Hierarchy
Stakeholder Requirement Statement
Stakeholder
Feature
Functional Interaction (Interaction)
State
System
Interface
System of Access
Input/ Output
Technical Requirement Statement
Functional Role
Design Constraint Statement
Design Component
"Learning" Agents Capable of Extraction of New Knowledge
"Fixed" Agents Capable of Applying What Was Learned

Figure 9: "Fixed" Agents apply past learned patterns from "Learning" Agents

Although agile methods often emphasize learning by human teams, in this project we are also examining the formalized accumulation of persistent knowledge that is not necessarily human-based in all cases. Compared to other, more product line-oriented, cases of System 1, the GCSS-J target information system was seen as not intended for deployment in multiple application configurations, and a higher balance of learned persistent memory about this single target information system was seen to be in the learned patterns of human team members. In Figure 6 and 9 the two "databases" shown represent (a) relatively fixed knowledge of System 1 and environment previously learned and available for use, versus (b) the more dynamic new learning also potentially occurring about that same domain.

Resources and Attention in System 2

The ASELCM Pattern described in (Schindel and Dove, 2016) provides Attention Management Features for the responsive dynamic (and potentially concurrent) allocation of limited System 2 resources to dynamically changing environments. In analyzing GCSS-J it was noted that this particularly applied in responding to high priority system security related threats.

States, Modes, and Learning in System 3

Agile Scrum includes the idea shown in Figure 8 as the interactions in the Conducting Sprint Process Retrospective state, during which the status and performance of System 2 (GCSS-J development and other life cycle management processes) are analyzed and updated. While shown within the sprint model, this is actually the role of System 3 in Figure 6. A number of GCSS-J practices and systems were developed and improved through this process, improving process performance to a high level of maturity. Indeed, the GCSS-J project adoption of the agile systems engineering approach itself was a leading example of this type of activity.

Concluding Remarks

The ASELCM project reviews well-working agile systems-engineering processes – so this case study may sound a little rosy, but it is factual and intended to be instructive. This case study reviews the process that developed and perpetually evolved a web-based software product, which in some circles is assumed to be the ideal environment for agile software development concepts: small feature chunks are natural, a loosely coupled product architecture is inherent, continuous delivery is possible, and DevOps is easily accommodated. But the analysis-team's eventual task is to find necessary principles with universal agile-system-engineering application. This and other ASELCM case studies are intended to provide a foundation for eventual extraction of those universal principles.

Initially the analysis team viewed the GCSS-J process as a maintenance process, rather than a systems-engineering process. The distinction was rooted in a belief that systems engineering is about developing and deploying a new system for the first time, and that subsequent upgrades would simply be maintenance of that original system. But a revelation occurred. Before the analysis workshop ended we realized we were analyzing an agile process that evolved an agile system, perpetually. When a deployed systems' environment continues to evolve it is necessary that systems engineering is also a continuous process, punctuated by a series of next generation evolved system instances. This is system life extension, a core conceptual view for systems expected to live effectively in today's rapidly evolving environments, and requires a life-extending agile architecture pattern as the basis of the deployed system's structure and design. It is not uncommon for web-based software systems to enable and exhibit continuous evolution, but the lesson of import is the general concept of perpetual evolution applied to all system types that need to provide sustained service.

This article introduced a circular asynchronous and simultaneous agile-life-cycle model framework, featuring the addition of a Research seventh stage. Unlike sequential-stage waterfall, agile systems engineering processes may be conducting activities in any and all of the stages simultaneously and without progressive sequencing. The newly acknowledged Research stage attends to active external and internal situational awareness – the critical driver of agile capability expression in all of the other stages. Active awareness and enabled mitigation of situational threats and opportunities to process and product alike, throughout the perpetual systems engineering process and the target system's life cycle, is a distinguishing feature of effective agile systems engineering – previously unrecognized explicitly for its fundamental driving role. The active Research awareness and mitigation activity breaths life into the agile systems engineering process, taking it beyond the repetitive execution of traditional development-sprint procedures.

This article illuminated a process of optimal control that contends effectively with potential disruptions to the integrity of a system-of-systems composed of independent constituent systems that change without notice; and contends effectively with a centralized hub that must give priority to rectifying frequent short-notice security vulnerability bulletins and frequent obsolesce of COTS/OSS elements. These disruptive situations impact the ability to execute new capability development planned for sprints, mitigated by frequent real-time re-prioritization of target system release evolution. The agility of the GCSS-J process to sustain continued service with integrity under frequent disruption has not appeared in the literature to the authors' knowledge.

References

Dove, R. and R. LaBarge. 2014. Fundamentals of Agile Systems Engineering – Part 1 and Part 2. International Council on Systems Engineering, International Symposium, Las Vegas, NV, 30Jun-3Jul. www.parshift.com/s/140630IS14-AgileSystemsEngineering-Part1&2.pdf

Dove, R., W. Schindel, and C. Scrapper. 2016. Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group. Proceedings International Symposium. International Council on Systems Engineering. Edinburgh, Scotland, 18-21 July. www.parshift.com/s/ASELCM-01SSCPac.pdf

Dove, R., W. Schindel. 2017. Case Study: Transition to Scaled-Agile Systems Engineering at Lockheed Martin's Integrated Fighter Group. Unpublished working paper. www.parshift.com/s/ASELCM-04LMC.pdf

Dove, R., W. Schindel, R. Hartney. 2017. Case Study: Agile Hardware/Firmware/Software Product Line Engineering at Rockwell Collins. Proceedings 11th Annual IEEE International Systems Conference. Montreal, Quebec, Canada, 24-27 April. www.parshift.com/s/ASELCM-02RC.pdf

GAO. 2011. GAO-12-7, Report to Congressional Committees: Information Technologies – Critical Factors Underlying Successful Major Acquisitions. US Government Accountability Office. October. http://www.gao.gov/assets/590/585842.pdf

ISO/IEC/IEEE. 2015. Systems and Software Engineering — System Life Cycle Processes. ISO/IEC/IEEE 15288:2015(E) first edition 2015-05-15. Switzerland.

ISO/IEC. 2010. Systems and Software Engineering — Life Cycle Management — Part 1: Guide for Life Cycle Management. ISO/IEC TR 24748-1:2010(E) first edition. Switzerland.

Schindel, W. and R. Dove. 2016. Introduction to the Agile Systems Engineering Life Cycle MBSE Pattern. Proceedings International Symposium. International Council on Systems Engineering. Edinburgh, Scotland, 18-21 July. www.parshift.com/s/160718IS16-IntroToTheAgileSystemsEngineeringLifeCycleMBSEPattern.pdf

US Department of Defense. 2014. Strategy for Improving DoD Asset Visibility. Department of Defense. Washington, DC (US): Assistant Secretary of Defense for Logistics and Materiel Readiness. January. http://gcss.army.mil/Documents/Articles/Strategy_for_Improving_DoD_Asset_Visibility.pdf

Acknowledgements

The agreement to host the workshop, choice of process to analyze, and arrangements for enabling the workshop were the efforts of Emmet (Rusty) Eckman III, Director, Business-Facing, Engineering and Sciences, Northrop Grumman, Sector Technical Fellow, and INCOSE ESEP.

The SE process analysis described is this article is the collective output of a workshop team that includes (alphabetically): Judith Dahmann (MITRE), John Davidson (Rockwell-Collins), Rusty Eckman (Northrop Grumman), Rick Dove (Paradigm Shift International), Tim Fulcher (Northrop Grumman), Kevin Gunn (MITRE), Suzette Johnson (Northrop Grumman), Mark Kenney (Northrop Grumman), Ken Laskey (MITRE), Sunil Lingayat (Northrop Grumman), David Lempia (Rockwell-Collins), Chris McVicker (Northrop Grumman), Erin Payne (Northrop

Grumman), Avinash Pinto (MITRE), Jack Ring (OntoPilot), Bill Schindel (ICTT Systems Sciences), and Chris Scrapper (US Navy).

Biographies

Rick Dove is CEO of Paradigm Shift International, specializing in agile systems research, engineering, and project management; and an adjunct professor at Stevens Institute of Technology teaching graduate courses in agile and self-organizing systems. He chairs the INCOSE working groups for Agile Systems and Systems Engineering, and for Systems Security Engineering, and is the leader of the INCOSE Agile Systems Engineering Life Cycle Model Discovery Project. He is an INCOSE Fellow, and the author of *Response Ability, the Language, Structure, and Culture of the Agile Enterprise*.

Bill Schindel is president of ICTT System Sciences. His engineering career began in mil/aero systems with IBM Federal Systems, included faculty service at Rose-Hulman Institute of Technology, and founding of three systems enterprises. Bill co-led a project on Systems of Innovation in the INCOSE System Science Working Group, co-leads the Patterns Working Group, and is a member of the lead team of the INCOSE Agile Systems Engineering Life Cycle Project.