Critical Success Factors for Scaling Agile Development

Barry Boehm
University of Southern California
United States
boehm@usc.edu

Doug Rosenberg
Parallel Agile, Inc.
United States
doug@parllelagile.com

Neil Siegel
University of Southern California
United States
siegel.neil@gmail.com

***Abstract*—A number of agile practices work well for smaller projects, but encounter difficulties for projects involving multiple organizations, over-100 developers, safety- and security-critical systems, and interoperablity with independently-evolving systems. Examples of large-project critical difficulties are daily stand-up meetings, 2-week sprints, single co-located customer representatives, projects with high personnel turnover, reliance on tacit interpersonal knowledge vs. documents, and need for rapid, continuous development and deployment.**

At USC, we are experimenting with an approach called Parallel Agile, which has been successfully used to develop widely different applications such as for Location-Based Advertising, Picture Sharing, Bad Driver Reporting, and a Pokemon-Go type of video game. The performers have been groups as large as 50 USC MS-Computer Science students working part-time, with high personnel turnover from semester to semester. We are currently beginning a large project for a commercial organization to replace an aging business management system, using the Parallel Agile approach. In researching the scaling of Parallel Agile to very large systems, we have identified a set of Critical Success factors (CSFs) from some large TRW government system projects that were able to succeed in large-scale parallel development with a set of similar CSFs.

This paper summarizes the Parallel Agile CSFs, compares them with the large TRW projects CSFs, and also identifies similarities with the Incremental Commitment Spiral Process Model (ICSM) and with other large-project agile approaches such as the Bosch Speed, Data Ecosystems approach and the Leffingwell Scaled Agile Framework (SAFe).

***Index Terms*—Agile Methods, Scaled Agile, Parallel Agile, Continuous Software Deployment**

I. Introduction

Some forms of agile development have scaled up well, but some have not. Some early failures include the Chrysler Comprehensive Compensation (C3) project, which started out well, but failed later when the initial agile gurus left the project, along with the on-site customer representative, who had a strong understanding of the customer organization and of the software, and over-fulltime dedication to the project. However, she turned out to be a single point of project failure when she wore out from overwork and was replaced by a Chrysler employee who was competent but far less knowledgeable and energetic [1].

Another agile scalability failure was the Thought Works ATLAS lease management project, which started out well using Extreme Programming (EP) but found that when the project reached 50 people and 500,000 lines of code, EP practices such as daily standup meetings, shared tacit knowledge of the code, 2-week system sprints, and lack of a project architecture or Big Design Up Front, were leading to project failure [2].

Our Parallel Agile (PA) approach (called Resilient Agile at the time), was initially described in our ICSSP 2017 paper [3]. It began when Doug Rosenberg, a guest lecturer in our USC software engineering courses, wanted to develop a system for Location-Based Advertising (LBA), in which drivers crossing a geofence around a restaurant or store would be presented on their screen with a button to push to get a bargain certificate. He issued a challenge to the students in our software engineering project course to participate in the development of an LBA system by developing the LBA use cases in parallel.

It started with two homework assignments to give students a hands-on learning exercise with UML and use case driven development, with the first homework assignment called "build the right system" and the second assignment called "build the system right". Homework 1 (Build The Right System) consisted of each student identifying requirements, storyboarding screens, writing use case narratives, and disambiguating their use cases via "robustness diagrams" (conceptual model-view-controller decomposition of a use case), while Homework 2 (Build The System Right) consisted of detailed design using class diagrams, sequence diagrams, database schemas, and occasionally state machines.

While grading Homework 2, Rosenberg offered an extra credit assignment for students to write prototype code for their use case, and 29 out of 47 students accepted the challenge. Their PA process used storyboards and prototypes to define both sunny-day and rainy-day scenarios; to define requirements for each use case; and to decompose each use case into a conceptual Model-View-Controller (MVC) pattern. PA also uses code generation from UML models to rapidly construct a domain-driven microservice architecture at the inception of a project. This microservice architecture is then used to enable prototype code to interact with a live database during requirements definition. PA then uses automatic test case generation from the same UML model. The common MVC pattern enabled all of the resulting 29 microservices to successfully interoperate.

The results of the 29 students were interesting enough that we decided to continue the project using Directed Research

(DR) students as scenario developers, and work continued over a summer session, and the following fall and spring semesters, once again with students working in parallel. At the end of the exercise, we had evaluated multiple technical approaches and refined the prototype code into a fully functional system.

Approximately 75 students overall worked on the LBA project, with a 90% staff turnover every 3 months. During this time we did extensive prototyping of different geofencing solutions, and introduced a test-team operating concurrently with the developers to evaluate the performance of the various prototypes and conduct in-the-field acceptance testing. DR students are given a time budget of 5 hours per week per unit, and most students take only 1 unit per semester. LBA was adopted for a time by the Santa Monica Chamber of Commerce, but the business volume was insufficient to continue.

Over the next few years we continued the experiment with three other student projects and began gathering productivity data on the various projects. This experimentation and data gathering is presently continuing with additional student projects as described below.

Our second attempt was with a photo-sharing app called PicShare, where we originally wanted to implement the same project with two independent teams, with one using Architected Agile and one using PA. Both teams produced good results, but the PA team took considerably less effort.

The third experiment was with a Crowdsourced Bad Driver Reporting System (BDR) which involved 15 students working in parallel for approximately 12 weeks at 5 hours per week per student. This was the first project where we deployed the database code generator. The BDR team developed a proof-of-concept system consisting of "dashboard camera" mobile apps for iOS (Swift) and Android (Java), and a web app implemented in Angular JS for filing, reviewing and querying video-centric bad driver reports. After 3 semesters of development the product is functional and has acquired a name, CarmaCam. Sample videos of bad driving incidents can be seen on www.carma-cam.com.

Our fourth PA experiment is a foray into game development, a Hawaiian-themed AR/VR territory game called TikiMan Go, where players throw lava fireballs at animated 3D tiki men in both VR and AR battle environments. The game is developed in Unity 3D with scripts in C that connect to Mongo DB via a Node JS API. Both the Tikiman game and the BDR/CarmaCam project have used the Executable Domain Model code generator with excellent results. Sample game videos can be seen at www.tikiman-go.com.

One interesting aspect of the DR-student projects is nearly 100% turnover every semester. Because we work one use case at a time, discovering requirements with prototyping and capturing requirements in the UML model we really don't lose much time when a new semester starts. Sometimes we throw the prototypes away and start the use case fresh with a better concept. This was the case with CarmaCam, where we spent two semesters prototyping voice activation and ultimately decided to switch to a one-touch video upload after we failed to meet reliability requirements with voice commands in a high ambient-noise environment.

One might think that the four diverse software projects described in the previous chapters would have had similar problems with a part-time project-set manager and up to 47 part-time USC MS students working in parallel. However, the projects satisfied a set of Critical Success Factors (CSFs), mainly because of Doug Rosenberg's CSF skills and experience in interactive system development and project management, and partly due to the similarities between the Parallel Agile CSFs and the CSFs in the Incremental Commitment Spiral Model (ICSM), which many of the student developers were familiar with as the ICSM book [4] was the textbook for USC's two primary software engineering courses.

The ICSM CSFs, presented in Section II, were strongly derived from a set of successful large interactive TRW command and control software projects, which also satisfied and extended the CSFs to address forms of Parallel Agile for very large systems. The first example was the 3-version, million-line Command Center Processing and Display SystemReplacement (CCPDS-R) project, described in Section III. The second example was a series of even larger TRW command-control-type projects that extended the set of CSFs and delivered highly successful results, described in Section IV.

Section V compares the PA approach and CSFs with other successful scalable agile approaches such as the Speed, Data, and Ecosystems approach [5], and the Scaled Agile Framework (SAFe) [6]. Section VI presents Conclusions to date.

II. Parallel Agile Critical Success Factors

The critical success factors for PA modify and extend the ICSM Three Team approach shown in Fig. 1, resulting in Fig. 2.

The main difference in the two approaches is that the 2014 ICSM three-team approach is organized around incremental development, while Parallel Agile is organized to support either incremental or continuous development and deployment.

The upper Architecture and Capabilities Evolution team has as its primary functions the determination of the win conditions of a system's primary success-critical stakeholders (clients, end users, safety/security regulators, maintainers, investors, etc.); the choices of system infrastructure, commercial services, and technology options, likely directions of future growth, and needed developer skills; and the development of prototypes such as with geofencing alternatives for Location-Based Advertising or voice vs. button-pressing for Bad Driver Identification (evaluated by the Continuous V&V team); and overall responsibilities of the Keeper Of The Project Vision (KOTPV) success-critical function identified in the [7] study of 19 large project practices and outcomes. Across the system life cycle, the Architecture and Capabilities Evolution team serves at the KOTPV.

Many KOTPV functions require more than one person, often a combination of a domain expert and a technology expert. In some domains, a single person can provide such a combination, but it is rare to find a single person who can perform the KOTPV function across several domains, as Doug

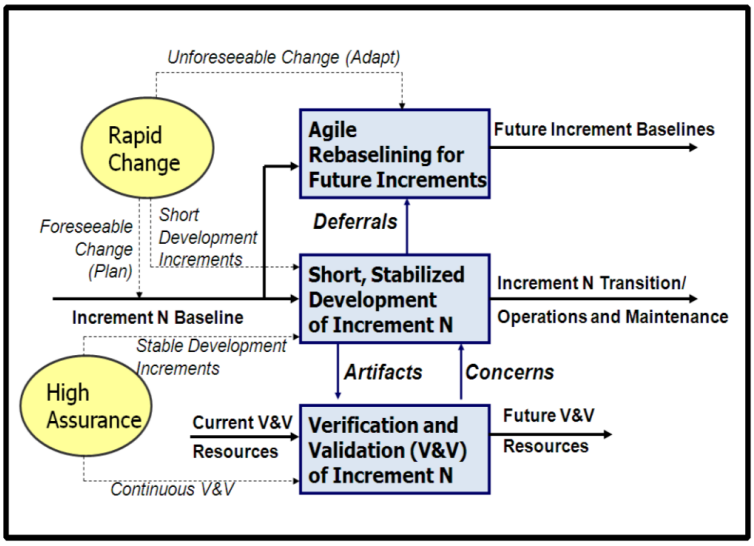

Fig. 1. ICSM 2014 Three-Team approach

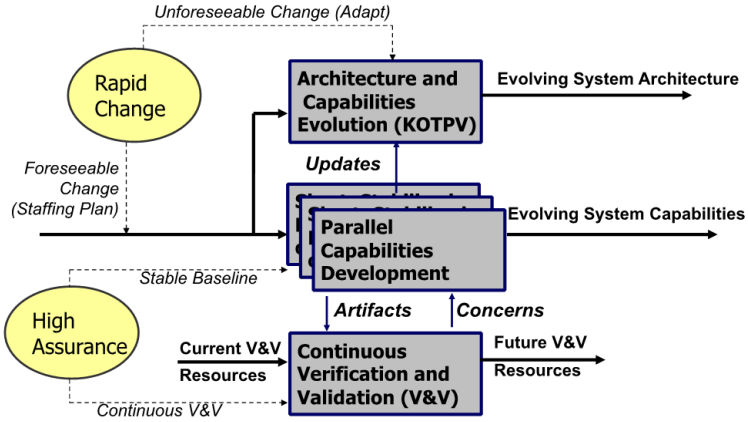

Fig. 2. Parallel Agile Three-Team Approach

Rosenberg did for the current 4 projects. For the large-scale TRW projects described in Sections III and IV, the KOTPV function is provided by a team of experts, although there will generally be a KOTPV team leader such as Walker Royce for CCPDS-R and Neil Siegel for the series of large command-control systems.

The central Parallel Capabilities Development team generally develops the desired capabilities in a series of increments determined by the stakeholders, while providing updates to earlier increments as needed. Its Critical Success Factors include the skills and facilities needed to accomplish the work, plus negotiation with the Architecture and Capabilities Evolution team on the priorities of evolving desired system capabilities vs. availability of developers with the required skills.

The Continuous Verification and Valuation (V&V) team does not wait for some code to be tested. It starts at the beginning of the project getting ready to support the ICSM principle and PA Critical Success Factor "Evidence and Risk-Based Decisions" by evaluating the feasibility of the initial decisions of the first two teams, such as the prototypes to use off-the shelf geofencing algorithms in Location-Based Advertising and voice reporting of bad driving in Bad Driver Reporting. As shortfalls in feasibility evidence are uncertainties and probabilities of loss, and Risk Exposure is calculated as (Probability of Loss) times (Impact of Loss), such evaluations enable the first two teams to explore less risky alternative solutions. The Continuous V&V team continues to prepare for and perform testing and evaluation of the first two teams' later artifacts.

III. TRW-USAF/ESC CCPDS-R Parallel Development

The Command Center Processing and Display System Replacement (CCPDS-R) project was to re-engineer the command center aspects of the US early missile warning system. It covered not only the software but also the associated

system engineering and computing hardware procurement. The software effort involved over 1 million lines of Ada code, across a family of three related user capabilities. The developer was TRW; the customer was the Air Force Electronic Systems Center (ESC); the users were the U.S. Space Command, the U.S. Strategic Command, the U.S. National Command Authority, and all nuclear-capable Commanders in Chief. The core capability of 355,000 lines of Ada code was developed on a 48-month fixed price contract between 1987 and 1991. While this was admittedly a long while ago in software time, the project closely mirrors current systems being developed in government and the private sector, and so is relevant as an example. A more detailed description of the CCPDS-R project is provided in pages 299-362 (Appendix D) of [8].

The project had numerous high risk elements. One was the extremely high dependability requirements for a system of this nature. Others were the ability to re-engineer the sensor interfaces, the commander situation assessment and decision-aid displays, and the critical algorithms in the application. Software infrastructure challenges included distributed processing using Ada tasking (a major problem with previous Ada projects was the incompatibility of software control threads caused by unrestricted use of Ada tasking) and the ability to satisfy critical-decision-window performance requirements. Many of these aspects underwent considerable change during the development process. The project involved 5 administration and 77 software personnel, most of whom had training in Ada programming but had not applied it.

CCPDS-R used standard Department of Defense (DoD) acquisition procedures, including a fixed-price contract and the documentation-intensive DoD-STD-2167A software development standards. However, by creatively reinterpreting the DoD standards, processes, and contracting mechanisms, USAF/ESC and TRW were able to perform with agility, deliver on budget and on schedule, fully satisfy their users, and receive Air Force awards for outstanding performance.

A. CCPDS-R Evidence-Based Decision Milestones

The DoD acquisition standards were acknowledged, but their milestone content was redefined to reflect the stakeholders' success conditions. The usual DoD-STD-2167A Preliminary Design Review (PDR) to review paper documents and briefing charts around Month 6 was replaced by a PDR at Month 14 that demonstrated working software for all the high-risk areas, particularly the network operating system, the message-passing middleware for handling concurrency, and the graphic user interface (GUI) software for commanders. The PDR also reviewed the completeness, consistency, and traceability of all of the Ada applications software interface specifications, as verified by the Rational Ada compiler and R-1000 toolset. Thus, a great deal of system integration was done before the software was developed, and enabled the 47 sequential-Ada programmers to develop their functions in parallel.

TRW invested significant resources into a package of message-passing middleware that handled the Ada tasking and concurrency management, and provided message ports to accommodate the insertion of sequential Ada packages for the various CCPDS-R application capabilities. For pre-PDR performance validation, simulators of these functions could be inserted and executed to determine system performance and real-time characteristics. Thus, not only software interfaces, but also system performance could be validated prior to code development, and stubs could be written to provide representative module inputs and outputs for unit and integration testing. Simulators of external sensors and communications inputs and outputs were developed in advance to support continuous testing. This created an executable architecture skeleton, within which hardware and software component simulators could be replaced by completed components for performance and interoperability assurance (more recently called a Digital Twin). Also, automated document generators were developed to satisfy the contractual needs for documentation.

Evidence of achievable software productivity was provided via a well-calibrated cost and schedule estimation model, in this case an Ada version of the Constructive Cost Model (Ada COCOMO), which was available for CCPDS-R. It was used to help developers, customers, and users better understand how much functional capability could be developed within an available budget and schedule, given the personnel, tools, processes, and infrastructure available to the project. Another major advantage of the Ada COCOMO cost/performance tradeoff analyses was to determine and enable the savings achieved via reuse across the three different installations and user communities.

Since the CCPDS-R plans and specifications were machine processable, the project was able to track progress and change at a very detailed level. This enabled the developers to anticipate potential downstream problems and largely handle them via customer collaboration and early fixes, rather than delayed problem discovery and expensive technical contract-negotiation fixes. Fig. 3 shows one such metrics-tracking result: the cost of making CCPDS-R changes as a function of time.

For CCPDS-R, the message-passing middleware and modular applications design enabled the project to be highly agile in responding to change, as reflected in the low growth in cost of change shown in Fig. 3. Further, the project's advance work in determining and planning to accommodate the commonalities and variabilities across the three user communities and installations enabled significant savings in software reuse across the three related CCPDS-R system versions.

B. CCPDS-R Parallel Development

The 48-month CCPDS-R core capability schedule included 20 technical personnel and 14 months of system definition and user prototyping (13 people), infrastructure definition and initial development (4 people), and initial testing and full-scale test planning (3 people). After a successful Preliminary Design Review, the full-scale core capability development included 77 technical personnel and 34 months of extending and evolving the infrastructure (11 people), having parallel sequential-Ada

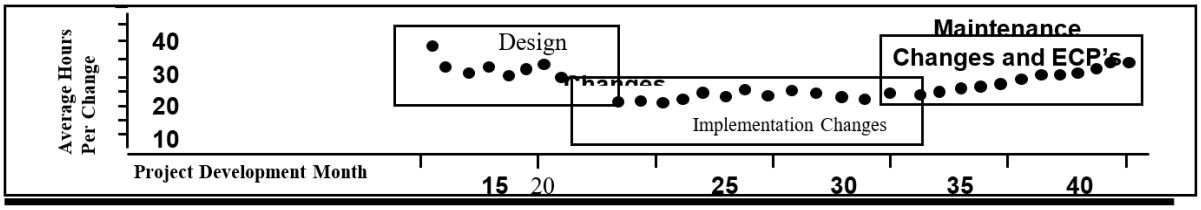

Fig. 3. Cost of changes vs. Time: CCPDS-R

programmers developing and evolving the system capabilities (47 people), and continuous V&V, test preparation and testing (23 people). It also included the full team adapting to organizational and mission changes during development. CCPDS-R also satisfied the CSF of low personnel turnover by having the fee for on-time delivery split 50-50 between corporate profit and bonuses for performers staying through project completion.

IV. Parallel Development of Even Larger TRW Defense Software Systems

Dr. Neil Siegel spent several years as the designated "project fix-it person" at TRW, during which time he actually did "fix" several big DoD software programs. During this hands-on experience, he developed a theory of the root cause of the problems that affected these particular programs, and worked with some very smart people to develop a "fix" based on that theory. Over the next 20 years, as an executive at TRW (and at Northrop Grumman, after Northrop acquired TRW in 2002), he had the opportunity to impose this "fix" on a dozen or so major software-intensive development programs. All finished on time and on budget, and all are liked by their users. Many are still in operational use 25 years later, and have proven not only to be good products, but highly maintainable and adaptable.

Siegel's theory postulated that the main source of large software system overruns, crashes, and expensive and ineffectual fixes were due to the degree of entanglement between the product's software control structure and its functional mission capabilities. (Note that the CCPDS-R project in Section III was aware of previous projects' difficulties with mixing functional software and control software, particularly Ada tasking, and created an architecture and control software that kept them separate). Siegel extended this by creating a general architecture for separating these concerns, generalizing the CCPDS-R system architecture skeleton approach to apply to a wider variety of applications, including weapon systems, sensors, and information systems as well as command and control systems, and supplementing this design pattern with a design-based methodology for implementing this design pattern. One important aspect of this method was his use of the design as a method explicitly to separate the most-difficult design and implementation tasks from those that required only more ordinary levels of skills and experience. One striking result that he achieved was the extreme difference in size achieved between the tasks deemed the most-difficult and the tasks deemed more-ordinary; the most-difficult tasks typically comprised less than 2% or 3% of the final lines of source code in the system. The small size of these most-difficult tasks made it feasible to find and recruit a small group of true experts for these tasks, while significantly decreasing the average difficulty of the remaining (e.g., 97% or 98%) of the work. These projects typically involved many hundreds of people; it was simply not feasible to try to recruit experts for all of these positions; in such large teams, most people are of average skill.

Siegel attributes this success at such separation as one of the key factors in the success of his methods. He also developed a set of objective technical metrics (such as the need to specify every independently-schedulable software entity within the system, and to severely limit the number of such entities) for the purpose of measuring progress during the design activity. He notes that metrics commonly used during the design phases of most engineering projects are managerial metrics (e.g., documents are produced, reviews are held, etc.), rather than technical metrics; one cannot, of course, measure actual technical progress without technical metrics.

In his Ph.D. research, he used data from several real DoD software-development programs to retrospectively establish a controlled observational case study, comparing programs performed using his design pattern and methodology with those that did not. He also had data from programs that started without his design-based technique, and switched to it mid-course, and even data from one program that started with the technique (and used it to successfully deliver an initial capability), then did a next capability without the technique (and had problems), and then re-adopted the technique for successful third and subsequent deliveries [9].

Fig. 4 shows a monthly measure of quality for each of 3 projects each that did, and 3 projects that did not, use the technique. It is clear that the design-based technique resulted in far higher quality, which was a major factor in keeping those projects that did use the technique on budget and schedule, while those projects that did not use the technique experienced cost and schedule erosion, at the same time that they experienced the quality problems directly indicated in the Figure.

The Force XXI Battle Command, Brigade and Below (FBCB2) project was the one that used the technique in period I, did not use it in period II, and returned to its use in period

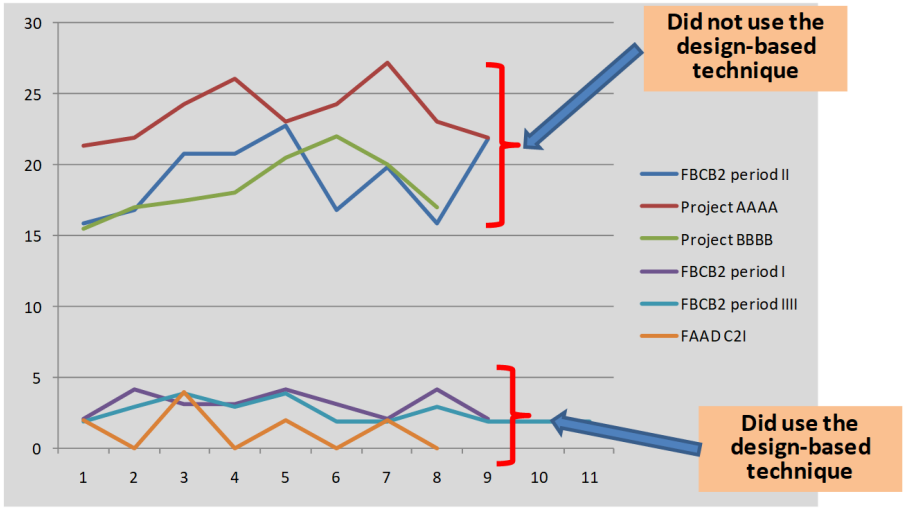

Fig. 4. Monthly overruns of using and not using the design-based technique

III. As with CCPDS-R, the design-based technique had an infrastructure team and a functional capabilities team operating in parallel.

Fig. 5 and Fig. 6 provide additional detailed perspectives of FBCB2's monthly mean time between failures and quality measures for the periods using and not using the design-based technique.

Clearly, the FBCB2 results were markedly better when the infrastructure team and the functional capabilities team were operating in parallel, and using Siegel's design pattern and design-based methodology.

V. Comparisons Between the PA Approach and CSFs With Other Successful Scalable Agile Approaches

A. *The Speed, Data, and Ecosystems Approach [5]*

In today's and tomorrow's fast-moving and competitive world, a key concept is John Boyd's OODA Loop, for Observe-Orient-Decide-Act, shown in Fig. 7. If your OODA Loop is slower than your competitor's, you are going to lose business. Over 50% of the 2000 Fortune 500 companies are now gone, largely due to slow responsiveness to change. Software is increasingly critical to this need for speed, not only in information-based companies like Amazon with a new release every 11 seconds, but also in industries like autos, where software provides over 80% of the auto's functionality.

For an organization's Speed, Data, and Ecosystems improvement, the Bosch approach provides a 5-step Stairway to Heaven sequence. For Speed, the steps are Traditional Development, Agile Development, Continuous Integration, Continuous Deployment, and RD as an Innovation System. For the Continuous Integration step, there is a three-part organizational model similar to the Parallel Agile three-part model. It is called ART, for Architecture, Requirements, and Testing. It differs from the PA Architecture and Capabilities Evolution part by preceding Requirements by Architecture.

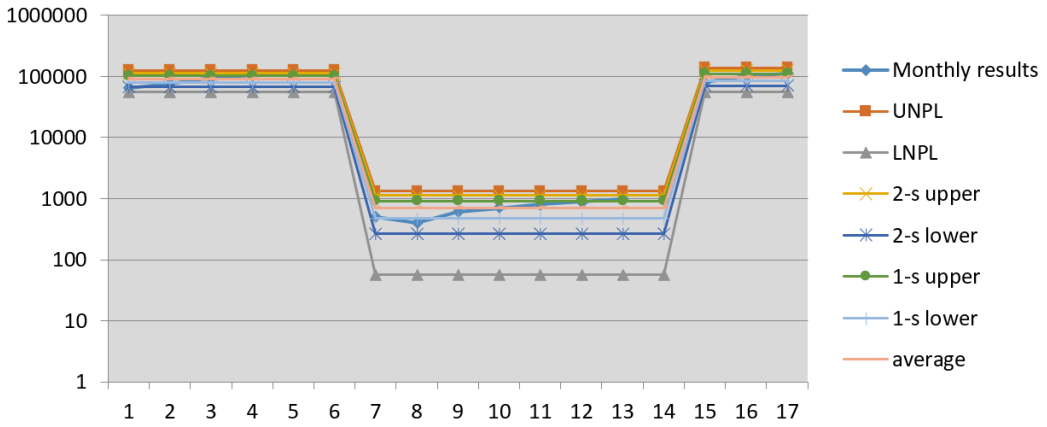

Fig. 5. Monthly FBCB2 mean time between failures in hours with and without the design-based technique

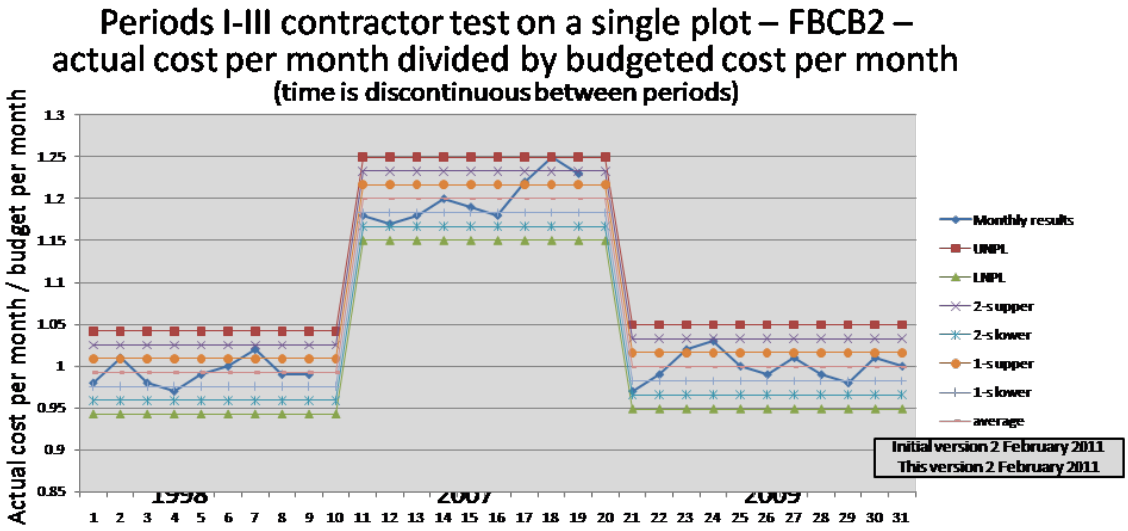

Fig. 6. Monthly FBCB2 actual vs. budgeted cost with and without the design-based technique

Testing is about the same for both, although PA includes V&V of the Architecture and Capabilities.

For Data, which is critical for the Observe and Orient parts of the OODA Loop, the five steps are Ad-Hoc, Collection, Automation, Data Innovation, and Evidence-Based Organization. Evidence-Based is similar to the PA and ICSM principle of Evidence and Risk-Based Decisions, while the Stairway is stronger in data collection for improvement experiments.

For Ecosystems, the five steps are Internally Focused, Ad-Hoc Ecosystem Engagement, Tactical Ecosystem Engagement, Strategic Single Ecosystem Engagement, and Strategic Multi Ecosystem Engagement. PA does not have a counterpart, although the Architecture and Capabilities Evolution part includes active search and evaluation of commercial services and products.

B. The Scaled Agile Framework (SAFe) Approach [6]

For sizable products and services, the SAFe Approach has another ART construct, the Agile Release Train. ARTs are long-lived, cross-functional teams of agile teams. Typically, they are virtual organizations (50-125 people) that create a solution or capability. Types of agile teams participating in an ART could include Business, Product Management, Hard-

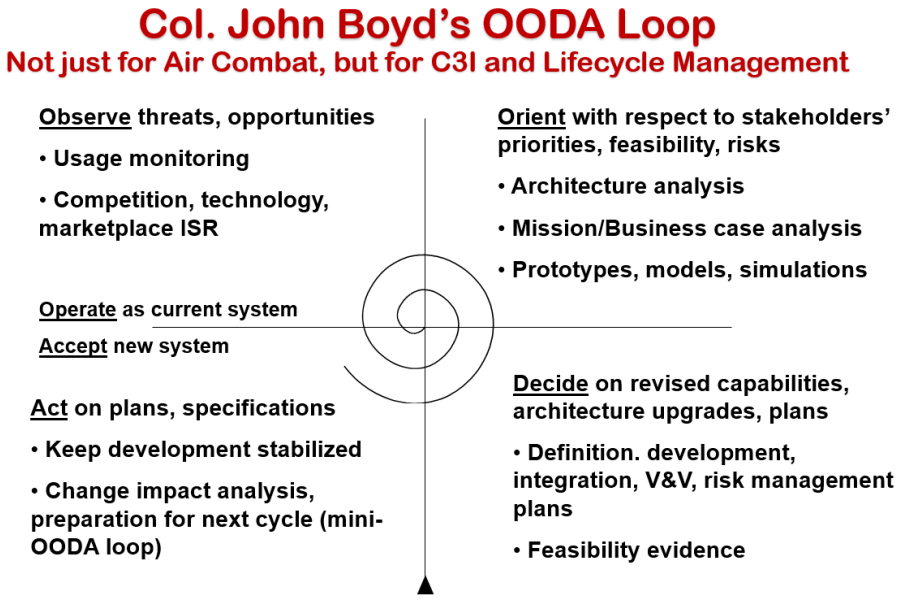

Fig. 7. Col. John Boyd's OODA Loop

ware, Software, Quality, Testing, Compliance, Security, or other domain-specific specialties. For scalability to very large systems, SAFe also has Solution Trains, which coordinate the work of several ARTs, and Super Trains, which coordinate the work of several Solution Trains.

SAFe has 9 fundamental principles: Take an economic view; Apply systems thinking; Assume variability, preserve options; Build incrementally with fast, integrated learning cycles; Base milestones on objective evaluation of working systems; Visualize and limit work-in-progress, reduce batch sizes, and manage queue lengths; Apply cadence, synchronize with cross-domain planning; Unlock the intrinsic motivation of knowledge workers; Decentralize decision making.

The SAFe principles are similar to the four ICSM principles. Stakeholder value-based guidance is partially addressed by Take an economic view and Apply systems thinking. Incremental commitment and accountability is addressed by Build incrementally. Concurrent multi-discipline engineering is addressed by the ART construct. Evidence and risk-based decisions are addressed by Base milestones on objective evaluation. SAFe and PA also both use the Unified Modeling Language (UML) and the Systems Modeling Language (SysML) for software and systems modeling.

SAFe also incorporates other Lean/Agile methods. For example, the SAFe principle, Visualize and limit work-in-progress, reduce batch sizes, and manage queue lengths, basically adopts the Kanban Approach [10].

VI. Conclusions To Date

"Scaling up Parallel Agile (PA)" could also be called "Speeding up the Incremental Commitment Spiral Model (ICSM)." Both address three-team approaches for developing and evolving a software product's architecting, developing, and V&Ving in parallel. The ICSM has provided PA with a method for large-scale software development, while PA has provided the 2014 incremental development version of the ICSM with capabilities to support increasing demands for rapid, continuous system development and deployment.

Both PA and ICSM include a set of Critical Success Factors (CSFs) in terms of three team responsibilities: Architecture and Capabilities Evolution using a Keeper of the Project Vision (KOTPV); Parallel Capabilities Development; and Continuous Verification and Validation (V&V). Evidence of the scalability of PA and Continuous ICSM was provided by a set of very large TRW projects that used the CSFs to succeed in rapid and reliable development, including one project where the CSFs were cut off, speed and reliability seriously fell, and were only restored when the CSFs were restored.

Based on the CSFs, PA and a continuous-delivery version of ICSM are comparably scalable to the two main scalable agile approaches: Speed, Data, and Ecosystems and the Scaled Agile Framework (SAFe).

References

[1] Doug Rosenberg and Matt Stephens. *Extreme programming refactored: the case against XP*. Apress, 2008.

[2] Amr Elssamadisy and Gregory Schalliol. Recognizing and responding to bad smells in extreme programming. In *Proceedings of the 24th International conference on Software Engineering*, pages 617–622. ACM, 2002.

[3] Doug Rosenberg, Barry Boehm, Bo Wang, and Kan Qi. Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process. In *Proceedings of the 2017 International Conference on Software and System Process*, pages 60–69. ACM, 2017.

[4] Barry Boehm, J Lane, Richard Turner, and Supannika Koolmanojwong. The incremental commitment spiral model, 2014.

[5] Jan Bosch. *Speed, data, and ecosystems: Excelling in a software-driven world*. CRC Press, 2017.

[6] Dean Leffingwell. *SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering*. Addison-Wesley Professional, 2016.

[7] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[8] Walker Royce. *Software project management*. Pearson Education India, 1998.

[9] Neil Gilbert Siegel. *Organizing complex projects around critical skills, and the mitigation of risks arising from system dynamic behavior*. University of Southern California, 2011.

[10] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.