Architected Agile Solutions for Software-Reliant Systems

Barry Boehm, Jo Ann Lane, Supannika Koolmanojwong
University of Southern California
{boehm, jolane, koolmano} at usc.edu

Richard Turner
Stevens Institute of Technology
rturner at stevens.edu

Abstract. Systems are becoming increasingly reliant on software due to needs for rapid fielding of "70%" capabilities, interoperability, net-centricity, and rapid adaptation to change. The latter need has led to increased interest in agile methods of software development, in which teams rely on shared tacit interpersonal knowledge rather than explicit documented knowledge. However, such capabilities often need to be scaled up to higher level of performance and assurance, requiring stronger architectural support. Several organizations have recently transformed themselves by developing successful combinations of agility and architecture that scale up to projects of up to 100 personnel. This paper identifies a set of key principles for such architected agile solutions for software-reliant systems, and illustrates them with several case studies.

Introduction

Systems are becoming increasingly reliant on software due to needs for rapid fielding of "70%" capabilities, interoperability, net-centricity, and rapid adaptation to change. This trend is shown in Figure 1, illustrating the percentage of aircraft functionality that relies on software vs. time, and the resulting system challenges as software was used to adapt to change [Van Tilborg, 2006].

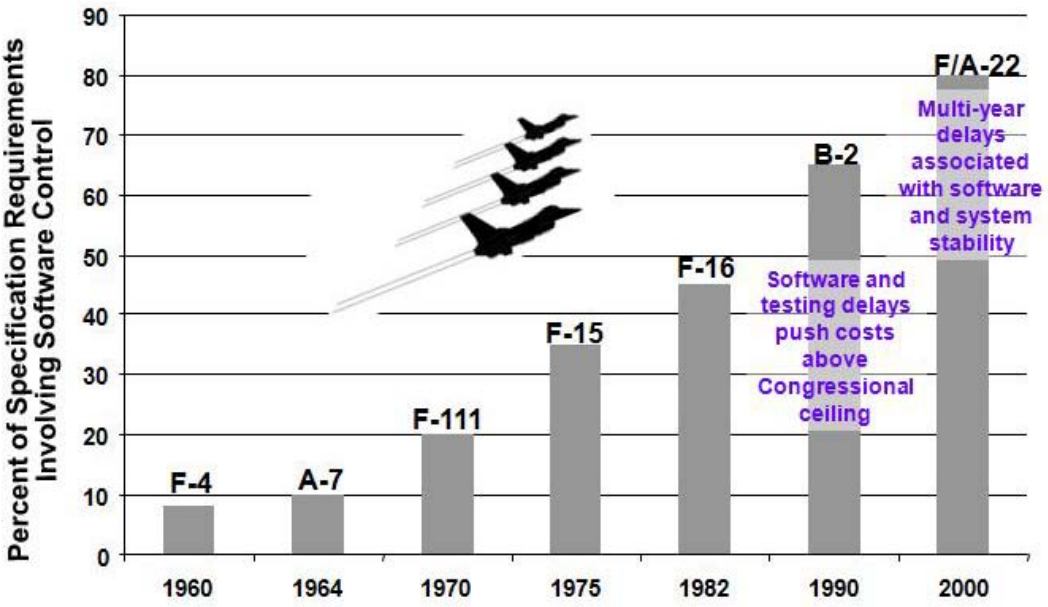

Figure 1. Trends in Software-Reliant Systems: Aircraft

Overall, small, less mission-critical projects with high rates of change are best accomplished by highly-skilled agile teams able to operate on shared, tacit knowledge with little documentation. Large, more mission-critical projects with less change and mixed developer skill levels are more successful using explicit documented knowledge such as architectural views and project plans to succeed. If the requirements of these large projects are relatively stable, architectures and plans will change infrequently, and a pure documented-architecture approach will succeed. However, as seen in Figure 1, large projects increasingly need product and process architectures that enable them to use agility to support more volatile requirements in areas like user interfaces, competition-critical features, or interfaces with independent, rapidly evolving external systems.

Also, our experiences in the commercial and public-service sectors indicate that a growing number of systems must integrate into larger enterprise software frameworks and systems of systems. This leads to complex interactions with evolving COTS products, legacy systems, and external systems, with the expectation that these systems "never fail." Emergent requirements, rapid changes, reused components, high levels of assurance, and dynamic market factors must also be addressed.

The relative importance of these different application sectors depends on what is counted. Organizations dealing mainly with large numbers of small, dynamic, less-critical projects will count the number of such projects. Organizations dealing with a mix of large and small projects will more likely count the investment costs and skilled effort that are consumed by their projects.

Figure 2 shows how the relative importance of agility and architecture varies when counting numbers of projects or percentage of costs. Drawn from Appendix E of [Boehm and Turner 2004] the information is based on data provided in [Highsmith 2002] of the relative number of projects in three size ranges across various business sectors. The particular data shown in Figure 2 is from the financial sector, with 65% of the projects having less than 10 people, 25% between 11 and 50 people, and 10% over 50 people.

By % of Projects

Criticality, Size / Stability	Low (78%)	High (22%)
High	Either	Arch
Low (80%)	Agile	Both

By % of Costs

Criticality, Size / Stability	Low (28%)	High (72%)
High	Either	Architecture
Low (80%)	Agile	Both

Figure 2. Relative Importance of Agility and Architecture

The boundary between Low and High Size projects is taken as 25 people. In 1999, eXtreme Programming (XP) innovator Kent Beck said, "Size clearly matters. You probably couldn't run an XP project with a hundred programmers. Not fifty. Not twenty, probably. Ten is definitely doable." [Beck 1999]. Since then, 20-25 person pure-agile projects have succeeded; for larger projects, investments in architecture are needed [Elssamadisy and Schalliol 2002]. The relative stability data come from cost model Requirements Volatility parameter trends, in which High stability means less than 5% change in requirements per year.

In this paper, we will identify key principles for balancing agility and architecture in the largest-cost sector where both agility and architecture are needed. In this sector, the software development activities must be lean and agile but also need a strong flexible architecture undergirding software functionality. If either is not adequately addressed, the software will not be viable over the long term. It will be late to market (or late to the war), too difficult to evolve, quickly reach a dead-end, and suffer a relatively short life.

We will also provide a risk-driven process framework, the Incremental Commitment Model, that enables projects to determine what mix of agile and architected approaches bets fits their system situation, and will provide quantitative results that show how this mix is generally a function of the system's size, criticality, and requirements volatility. We will then analyze case studies of one of the most successful mixed approaches, the Architected Agile approach, and present critical success factors for applying the approach.

Key Principles

Several analyses of successful programs have been performed to determine the kind of processes that satisfactorily address current trends and challenges [OUSD AT&L 2008; Pew and Mavor 2007] The strengths and difficulties of current process models have also been analyzed [Pew and Mavor 2007], finding that while each had strengths, each needed further refinements to address all of the identified challenges. The most important conclusion, though, was that there were key process principles that address the challenges, and that the form of the process models was less important than the ability to adopt the principles. These key principles are:

1. Commitment and accountability of success-critical stakeholders
2. Stakeholder satisficing based on success-based negotiations and tradeoffs
3. Incremental and evolutionary growth of system definition and stakeholder commitment.
4. Iterative system development and definition
5. Concurrent system definition and development allowing early fielding of core capabilities, continual adaptation to change, and timely growth of complex systems without waiting for every requirement and subsystem to be defined
6. Risk management – risk driven anchor point milestones which are key to synchronizing and stabilizing all of this concurrent activity.

A new process model framework, the Incremental Commitment Model (ICM) [Boehm and Lane 2008], was developed to build on the strengths of current process models: early verification and validation concepts in the V-model, concurrency concepts in the Concurrent Engineering model, lighter-weight concepts in the Agile and Lean models, risk-driven concepts in the spiral model, the phases and anchor points in the Rational Unified Process (RUP), and recent extensions of the spiral model to address SoS acquisition. The model framework, illustrated in Figure 3, explicitly:

- Emphasizes concurrent engineering of requirements and solutions
- Establishes Feasibility Rationales as pass/fail milestone criteria
- Enables risk-driven avoidance of unnecessary documents, phases, and reviews

- Provides support for a stabilized current-increment development concurrently with a separate change processing and rebaselining activity to prepare for appropriate and stabilized development of the next increment.

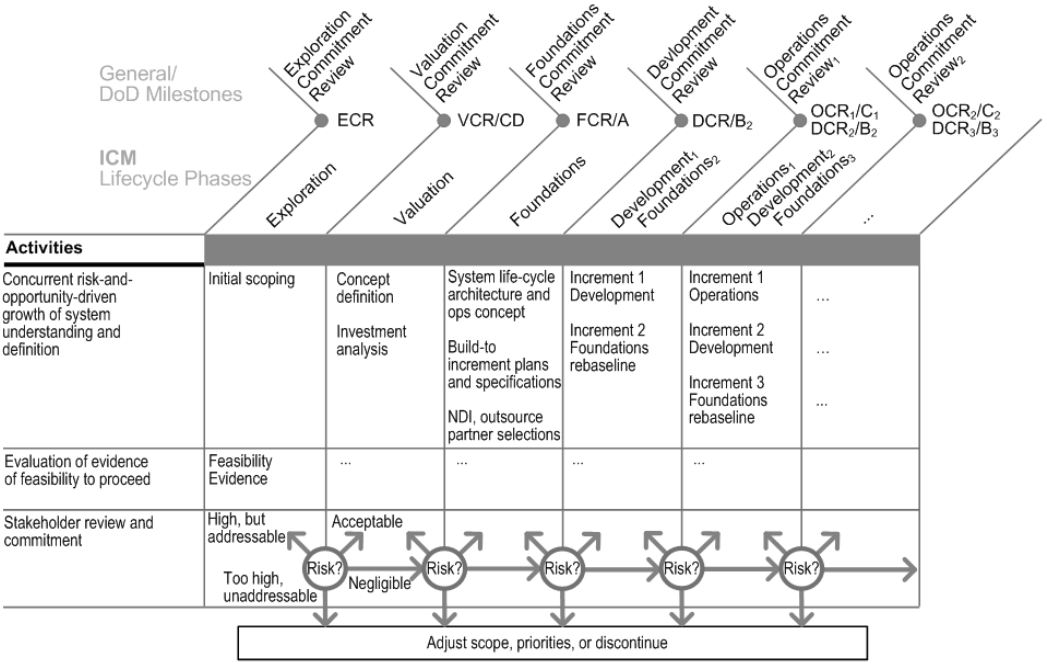

Figure 3. ICM Overview

Figure 3 shows the relationship between the concurrently engineered life cycle phases, the stakeholder commitment review points with their use of feasibility rationales to assess compatibility, and the resulting risk assessment to decide how to proceed at each commitment review point. There are a number of alternatives at each commitment point, leading to many risk-driven paths through the life cycle. These are:

(1) the risk is negligible and no further analysis and evaluation activities are needed to complete the next phase;

(2) the risk is acceptable and work can proceed to the next life cycle phase:

(3) the risk is addressable but requires backtracking; or

(4) the risk is too great and the development process should be rescoped or halted.

Risk is assessed by the system's success-critical stakeholders, whose commitment will be based on whether the current level of system definition gives sufficient evidence that the system will satisfy their value propositions.

The ICM is a hybrid agile/plan-driven process that integrates a) agile processes for assessing the system environment and user needs and then planning for the implementation of new and modified system capabilities, b) plan-driven (often time-boxed) processes to develop and field new capabilities, and c) continuous verification and validation (V&V) to provide high assurance of the requisite system qualities. Figure 4 shows one of the risk-driven process patterns of the ICM that best fits the need for simultaneously accommodating rapid change and the need for high assurance on larger projects. High assurance is achieved by stabilizing a build-to-specification development team, anticipating foreseeable change and architecting for its easy

accommodation, providing continuous V&V to ensure rapid and efficient defect fixes, and diverting unforeseeable changes to the concurrently-operating agile team assessing the changes and rebaselining the plans and specifications for development of the next increment by the build-to-spec team.

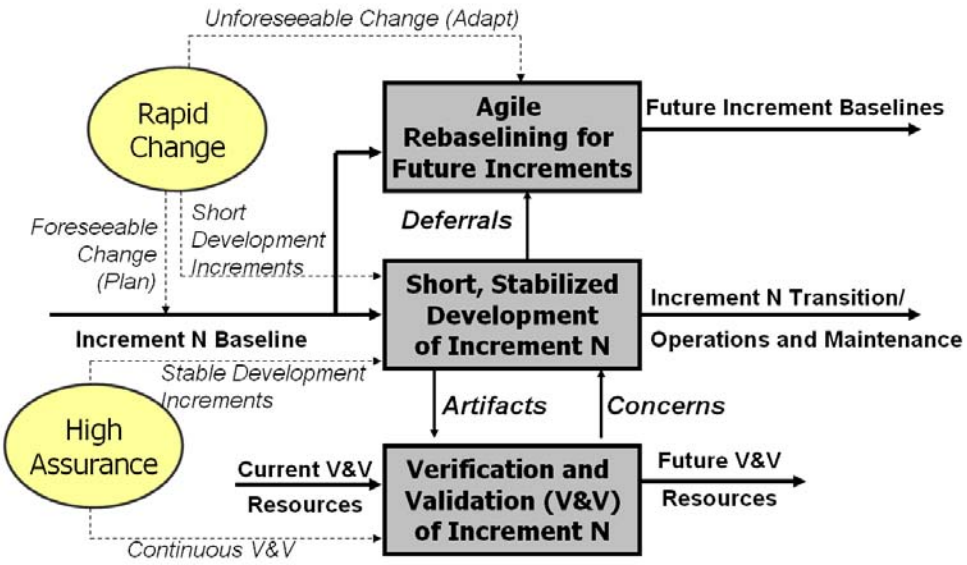

Figure 4. Hybrid Agile/Plan-Driven Process [Boehm and Lane 2008]

The ICM framework acknowledges there can be no "one size fits all" process model and that a key feature of any successful framework is the ability to tailor processes based on key project characteristics and risks. The ICM uses project characteristics and risks to determine how much process agility or rigor is enough to satisfy the system's objectives subject to its constraints. As a result, several common types of software-intensive systems have been identified along with guidance for tailoring the ICM framework for each case [Boehm and Lane 2008]. These common types range from small agile types of projects, through the larger-project pattern in Figure 4, to more complex, adaptive processes for nondirected systems of systems. The goal of the ICM tailoring process is to establish the right amount of rigor, assurance, and predictability, while remaining agile and timely.

Successfully satisficing with respect to a diverse set of stakeholders in a multi-team development environment is critical to larger software projects. Studies [Carlile 2002] that have focused on knowledge management and transfer between different groups such as stakeholders and product developers and distributed development teams have identified key "boundary objects" that are used to guide the development and integration of new products such as software. These include key models such as architecture and data models that allow the different developers to quickly communicate with non-technical stakeholders and other developers and to coordinate work so that pieces come together quickly in the integration environment [OUSD (AT&L), 2008]. As shown in the next section, cost estimation models can support mutual understanding of cost implications of project decisions -- such as how little or how much to invest in up-front architecting.

How Much Architecting is Enough?

Size, criticality, and volatility are key decision drivers for focusing on agile or architected approaches. But critical questions remain about how much architecting is enough for a particular project. Here we provide a quantitative approach that has helped projects address this question. It extends the ROI of SE analysis described in [Boehm 2008].

The graphs in Figure 5 show the results of a risk-driven "how much architecting is enough" analysis, based on the COCOMO II Architecture and Risk Resolution (RESL) factor. This factor was calibrated along with 22 others to 161 project data points. It relates the amount of extra rework effort on a project to the percent of project effort devoted to software-intensive system architecting. The analysis indicated that the amount of rework was an exponential function of project size.

A small (10 thousand equivalent source lines of code, or KSLOC) could fairly easily adapt its architecture to rapid change via refactoring or its equivalent, with a rework penalty of 14% between minimal and extremely thorough architecture and risk resolution. However, a very large (10,000 KSLOC) project would incur a corresponding rework penalty of 91%, covering such effort sources as integration rework due to large-component interface incompatibilities and critical performance shortfalls.

Actually, the RESL factor includes several other architecture-related attributes besides the amount of architecting investment, such as available personnel capabilities, architecting support tools, and the degree of architectural risks requiring resolution. Also, the analysis assumes that the other COCOMO II cost drivers do not affect the project outcomes.

The effects of rapid change (volatility) and high assurance (criticality) on the sweet spots are shown in the right hand graph. Here, the solid black lines represent the average-case cost of rework, architecting, and total cost for a 100-KSLOC project as shown at the left. The dotted red lines show the effect on the cost of architecting and total cost if rapid change adds 50% to the cost of architecture and risk resolution. Quantitatively, this moves the sweet spot from roughly 20% to 10% of effective architecture investment (but actually 15% due to the 50% cost penalty). Thus, high investments in architecture and other documentation do not have a positive return on investment due to the high costs of documentation rework for rapid-change adaptation.

The dashed blue lines at the right represent a conservative analysis of the effects of failure cost of architecting shortfalls on the project's effective business cost and architecting sweet spot. It assumes that the costs of architecture shortfalls are not only added rework, but also losses to the organization's operational effectiveness and productivity. These are conservatively assumed to add 50% to the project-rework cost of architecture shortfalls to the organization. In most cases for high-assurance systems, the added cost would be considerably higher.

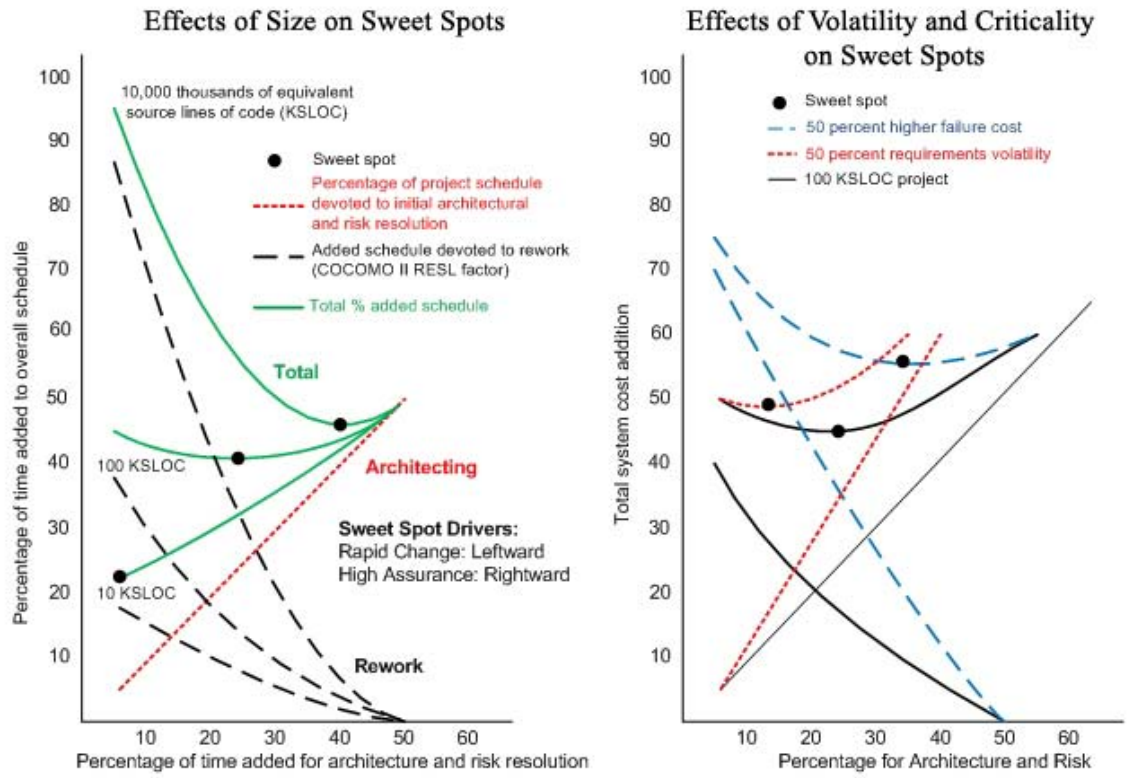

Figure 5. How Much Architecting is Enough?

Quantitatively, this moves the sweet spot from roughly 20% to over 30% as the most cost-effective investment in architecting for a 100-KSLOC project. It is good to note that the "sweet spots" are actually relatively flat "sweet regions" extending 5-10% to the left and right of the "sweet spots." However, moving to the edges of a sweet region increases the risk of significant losses if some project assumptions turn out to be optimistic.

Early Architected-Agile Success Cases

Most successful large scale projects have learned to balance architecting and agility. When these projects find a good balance, they can set up a battle-rhythm using a process somewhat akin to the hybrid agile/plan-driven process described in Figure 4. They welcome change and manage it through a product backlog, planning capabilities for multiple future increments, based on stakeholder priorities. They stabilize development for the current increment by defining the changes to be incorporated into the update cycle. Development is supported by continuous V&V, often with mature integration and test environments that support experimentation as well as quality assessments.

A comparison of the 6 key ICM principles [Boehm and Lane 2008; Poppendieck and Poppendieck 2003; Agile Manifesto 2009] with characteristics of lean and agile principles was conducted and a summary of the results are shown in Table 1. Although there are differences in the level of detail in the way each set of principles is specified, there are no substantial differences with respect to architecting. All focus on efficiently performing value-adding

activities at the appropriate point in the development life cycle and eliminating activities that don't add value.

Table 1. Key Principles Comparison

ICM Principles [Boehm and Lane 2007]	**Related Lean Principles** [Poppendieck and Poppendieck 2003]	**Related Agile Principles** [Boehm 2007; Agile Manifesto 2007]
(1) Commitment and accountability of success-critical stakeholders	• Respected leaders and champions • Team commitment • Master developers to guide decisions, make rapid progress, and develop high-quality software	• Business people and developers must work together daily throughout the project • Provide the developers with environment and support they need
(2) Success-critical stakeholder satisficing	• Joint customer-developer iteration planning • Value stream mapping	• Joint customer-developer iteration planning • Satisfy the customer through early and continuous delivery of valuable software
(3) Incremental growth of system definition and stakeholder commitment (4) Iterative development cycles	• Balance experimentation with deliberation and review • Iteration planning with negotiable scope and convergence	• Deliver working software frequently • Working software is the primary measure of progress
(5) Concurrent engineering	• Decide as late as possible to support concurrent development while keeping options open • Ensure emergence of a good architecture through reuse, integrated problem solving, and experienced developers	• The best architectures, requirements, and designs emerge from self-organizing teams.
(6) Risk-based activity levels and milestones	• Eliminate waste • Value stream mapping	• Team reflects periodically on how to become more effective, then tunes and adjusts its behavior accordingly • Simplicity--the art of maximizing the amount of work not done--is essential.

This section provides five examples. First, we summarize three architected-agile case studies involving multinational US and European corporate transformations. In each, top management commissioned an internal technical and management expert to lead a corporate-wide effort to transform the company's software development practices into a more agile Scrum of Scrums approach. All used a transformed corporate information architecture and framework that enabled compatible agile development while satisfying various corporate assurance and governance needs such as medical safety, physical platform safety, always-on availability, and US Sarbanes-Oxley corporate information management accountability [Highsmith 2002]

Next, we summarize a Scrum of Scrums approach evolved by a large US aerospace company to enable more rapid and cost-effective logistics support for an evolving product line of aerospace vehicles. Finally, we describe the Composite Health Care System (CHCS) [Lane and Zubrow 1996, MHS Help Desk 2009] a large Government health care system, that evolved a more lean and architected-agile approach as part of a major process improvement, while providing continuity of service across over 700 Department of Defense (DoD) hospitals and clinics

worldwide. Each of these case studies has risk-driven approaches similar to the Supply Chain Management example in Boehm-Turner [Boehm and Turner 2004], but with additional considerations of transforming their enterprises across multiple user sites and applications.

Architected Agile Corporate Transformations. The first of these corporate transformations involved a US medical services company with over 1000 software developers in the US, two European countries, and India. The corporation was on the brink of failure, due largely to its slow, error-prone, and incompatible software applications and processes. Top management commissioned a senior internal technical and management expert in both medical safety-critical applications and agile development, to organize a corporate-wide team to transform the company's software development approach to address its agility, safety, and Sarbanes-Oxley governance and accountability problems.

Software technology and project management leaders from all of its major sites were brought together to architect a corporate information framework, and develop a corporate architected-agile process approach. The resulting Scrum of Scrums approach was successfully used by the team in a collocated pilot project to create the new information framework in a way that would accommodate continuity of service in their existing operations.

Based on the success of this pilot project, the team members returned to their sites and led similar transformational efforts. Within three years, they had almost 100 Scrum teams and 1000 software developers using compatible and coordinated architected-agile approaches. The effort involved their customers and marketers in the effort. Expectations were managed via the pilot project. The release management approach included a 2-12 week architecting Sprint Zero, a series of 3-10 one-month development Sprints, a Release Sprint, and 1 to 6 months of beta testing; the next release Sprint Zero overlapped the release Sprint and beta testing. Their agile Scrum approach involved a tailored mix of eXtreme Programming (XP) and corporate practices, 6-12 person teams with dedicated team rooms, and global teams with wiki and daily virtual meeting support, working as if located next-door.

The second and third corporate transformations were similar. One involved a World-100 European products company with major software sites in Europe, the US, India, and China. The third involved a large European IT company with major development centers in Europe, India, and Israel. Each applied the six key principles above of (1) stakeholder commitment (top management, developers, external stakeholders); (2) stakeholder satisficing (e.g.,via Scrum prioritized backlogs); (3,4,5) incremental, evolutionary, iterative, concurrent development starting with a talented early-success core team from all major sites; and (6) risk-driven commitment milestones for expanding the number and distribution of the Scrum teams (over 30 in three years for company-2, including development of a new corporate architecture; over 40 in one year for company-3, which already had a strong corporate architecture).

Automated Maintenance Support System. This project was done by a major aerospace company on a government contract. It involved an advanced, highly automated vehicle maintenance application with extensive on-board trend and anomaly analysis on each vehicle, and net-centric communication of impending service needs to the nearest relevant vehicle maintenance center. The multi-mission vehicle versions and components can encounter many unanticipated interaction effects, leading to a high rate of change in the maintenance and diagnostic software.

Originally, the project used a Scrum of Scrums organization similar to those involved in the three corporate-internal case studies above. But the project encountered numerous coordination challenges across the multi-mission, multi-owner vehicle versions. These included decision priorities for common components and product backlog, multi-team requirement implementations, and needs for teams to interrupt their progress to help other teams. The number and magnitude of these challenges caused the classic single-owner Scrum of Scrums architected-agile approach to lose momentum.

The project recognized this and evolved to a more decentralized Scrum-based approach, with centrifugal tendencies monitored and resolved by an empowered Product Framework Group (PFG) consisting of the product owners and technical leads from each development team, and the project systems engineering, architecting, construction, and test leads. The PFG meets near the end of an iteration to assess progress and problems, and to steer the priorities of the upcoming iteration by writing new backlog items and reprioritizing the product backlog. A few days after the start of the next iteration, the PFG meets again to assess what was planned vs. what was needed, and to make necessary adjustments.

This approach has been successful in keeping the project's momentum at a high level and pointed in the right direction. This example and similar multi-mission, multi-stakeholder net-centric systems of systems have led to somewhat different combinations of agility and architecture. But overall, each has found ways to succeed by applying the six critical success factor principles discussed above. It is also important to remember that it takes good people to understand when and how to apply the principles to determine, tailor, and apply processes or architectures to complex system situations.

Composite Health Care System (CHCS) [Lane and Zubrow 1996, MHS Help Desk 2009]. CHCS is an automated health care system that in 1996 was installed in about 500 Department of Defense (DoD) hospitals and clinics worldwide. The application consists of several subsystems: patient administration, patient appointments and scheduling, managed care program, clinical, laboratory, radiology, pharmacy, dietetics, quality assurance, workload accounting menu, medical services accounting, ambulatory data menu, and medical records tracking. In a study of process improvements initiated in the mid 1990s, Lane and Zubrow [Lane and Zubrow, 1996] documented key attributes of the development process as well as the successes in improving software quality, time to market, and customer satisfaction. The process improvement activities were implemented in response to customer dissatisfaction with increasing development cycle durations for new software releases and unacceptable software quality when a release was installed at an alpha site for operational test and evaluation. Root cause analysis showed that these problems were primarily due to requirements creep late in the development cycle, causing both a slip in the schedule and inadequate testing before the software was delivered.

The improvements initiated on this program were all related to key ICM principles and architected agile practices: committed and accountable stakeholders and development team, more continuous V&V (peer reviews and testing) throughout the development cycle, agile analysis of incoming changes requests and deferring as much as possible to future releases, and stabilization of the current development cycle. This stabilization of the development cycle resulted in development teams working in parallel on the current major release and two maintenance updates while planning the next major release instead of the previous approach of continuing to slip new features into the current release right up until delivery.

To quantify the impact of these planned changes, the organization collected a set of measurements with respect to the previous software release and established this as their baseline (V4.11). These sets of measures were customer satisfaction, developer productivity (average number of logical source statements (LSS) output per month during the development cycle), cost per LSS (average number of dollars expended to develop each LSS), cycle time (number of months from software release start to delivery at an alpha site), and error rates (number of defects detected during the first 30 days of operation and the number detected during the first year of operation). The measures where then captured for the next release (V4.2) in which the process improvements had been implemented and compared to the baseline. Analysis of the new release measures showed that in the first increment after the process changes were implemented, the cost per LSS dropped 29%, cycle time dropped 46%, and error rates at the alpha site dropped 90%. Additional improvements in later releases were achieved with further development environment tool upgrades and the application of more advanced processes, but none of the subsequent results were as dramatic as the first set.

Key to continuing to deliver new, high quality capabilities every year were a stable but flexible software architecture and a core database managed by a single database team, continual stakeholder involvement, managing the contents of each upgrade, frequent status checks with committed and accountable stakeholders to manage obstacles and make timely decisions, minimal documentation, and continuous V&V. High priority features were assigned to each major increment and lower priority features in the product backlog were incorporated when the associated code was undergoing change for a high priority feature. As would be expected for a health care system, patient safety was a top priority and continuous testing was necessary to meet this goal.

Implications for Practice and Future Research

The above cases illustrate successes in the use of architected agile and show different approaches for using an architected agile process as well as the business drivers that led to the migration of an architected agile process. One would like to have a set of criteria for determining when to use an architected agile process versus a pure agile process versus a pure architected process. In addition, one would also like a set of criteria for determining which of a number of common-case system and software engineering processes to use, including cases where multiple systems form a system of systems (SoS). This set of criteria could then also provide guidance for selecting the process type(s) best suited for the various SoS constituent systems, or for parts of very diverse individual systems.

Our recent research in this area has identified a set of common-case processes and risk-driven criteria that indicate the homeground for each process, and also provide examples and likely build and release durations that are used in these common-case processes. Over time and with experience in their use, we have evolved this set of common-case processes and their associated attributes and decision criteria. The information presented in Table 2 below is a summary of version 4. The top row provides the criteria; the left-hand column names the common cases. In general, the special cases are listed in order of complexity and rigor. More detail is provided in Chapter 5 of [Boehm and Lane, 2008].

Table 2. Homegrounds for Common Software-Intensive System Processes

Common Case	Size, Complexity	Change Rate (%/Month)	Application Criticality	Available NDI Products	Organizational and Personnel Capability
Use NDI				Complete	
Agile	Low	1-30	Low-Med	Good; in place	Agile-ready Med-high
Architected Agile	Med	1-10	Med-High	Good; most in place	Agile-ready Med-high
Formal Methods	Low	0.3	Extra High	None	Strong formal methods experience
HW with embedded SW component	Low	0.3-1	Med-Very High	Good; in place	Experienced; med-high
Indivisible IOC	Med-High	0.3-1	High- Very High	Some in place	Experienced; med-high
NDI- intensive	Med-High	0.3-3	Med-Very High	NDI-driven architecture	NDI- experienced; med-high
Hybrid agile/ plan-driven	Med-Very High	Mixed parts; 1-10	Mixed parts; Med-Very High	Mixed parts	Mixed parts
Multi-owner system of systems	Very High	Mixed parts; 1-10	Very High	Many NDIs; some in place	Related experience, med-high
Family of systems	Med-Very High	1-3	Med-Very High	Some in place	Related experience, med-high
Brownfield	High-Very High	0.3-3	Med-High	NDI as legacy replacement	Legacy re-engineering
Net- Centric Services—Community Support	Low-Med	0.3-3	Low-Med	Tailorable service elements	NDI- experienced
Net-Centric Services—Quick Response Decision Suppport	Med-High	3-30	Med-High	Tailorable service elements	NDI- experienced
Legend : HW: Hardware; IOC: Initial Operational Capability; NDI: Non-Development Item; SW: Software.					

These criteria and the additional common-case information provided in [Boehm and Lane, 2008] can be used by organizations today. In those cases where an organization needs to use a certain process because of contractual constraints or business drivers, mismatches with the criteria and risks can be identified and appropriate mitigations can be taken to foster success.

As indicated by the evolving nature of the decision criteria and number of common-case processes, much further research will be needed to continue to evolve these and add detail about their characteristics and implementation sub-processes. We are continuing to address these, but welcome further improvements based on others' experience and research.

Conclusions

There are no single one-size-fits-all process or product models that can be applied to the wide variety of systems needing to be addressed now and in the future. However, there are key principles (stakeholder commitment and accountability; stakeholder satisficing; incremental and evolutionary growth of system definition and stakeholder commitment; iterative system

development and definition; and risk management) for determining appropriate process and product models for different system situations.

Table 1 showed how these principles are addressed by three approaches: lean development, agile development, and the Incremental Commitment Model (ICM). Each of these has strengths; the ICM provides the most explicit risk-based decision criteria for determining which overall process category fits which particular situation, and for tailoring of variations within and among the process categories, as with the Corporate Transformations, AMSS, and CHCS examples.

Another key set of decisions involves how much of agility and architecting is enough. A multivariate analysis of 161 projects, represented in Figure 5, provides additional guidance for such decisions, depending primarily on project size, criticality, and volatility. Finally, it is important to remember that good people are essential in determining how to determine, tailor, and apply process or architectures to complex system situations.

References

Beck, K. (1999); *Extreme Programming Explained*. Reading, MA: Addison-Wesley.

Boehm, B. (2007); "Agility and quality." IBM Agile Conference, May 15, 2007; ICSE Workshop on Software Quality, May 21, 2007.

Boehm, B. and Turner, R. (2004); *Balancing agility and discipline: a guide for the perplexed*. Addison-Wesley, Boston.

Boehm, B. and Lane, J. (2008); *Incremental Commitment Model Guide*, version 0.5, Center for Systems and Software Engineering Technical Report, http://csse.usc.edu/csse/TECHRPTS/2009/usc-csse-2009-500/usc-csse-2009-500.pdf.

Boehm, B., Valerdi, R., and Honour, E. (2008); "The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems," *Systems Engineering*, Vol. 11, No.3, Fall 2008, pp. 221-234.

Carlile, P. (2002); "A pragmatic view of knowledge and boundaries: boundary objects in new product development." *Organization Science* Vol. 13: 442-455.

Elssamadisy, A. and Schalliol, G. (2002); "Recognizing and Responding to 'Bad Smells' in Extreme Programming," *Proceedings, ICSE 2002*, pp. 617-622.

Highsmith, J. (2002); *Agile Software Development Ecosystems*. Boston: Addison-Wesley.

Lane, J. and Zubrow, D. (1996); "Metrics focus brings about improvement in health care software development. *Proceedings, SEI Software Engineering Process Group Conference*, March 1996.

Military Health System (MHS) Help Desk, System: CHCS, http://www.mhs-helpdesk.com/Pages/chcs.asp , accessed on 8/4/2009.

Office of the Under Secretary of Defense for Acquisition, Technology and Logistics (OUSD AT&L), (2008); *Systems engineering for systems of systems, Version 1.0*. Washington, DC: Pentagon.

Pew, R. and Mavor, A. (2007); *Human-system integration in the system development process: a new look*. National Academy Press, 2007.

Poppendieck, M and Poppendieck, T. (2003); *Lean software development, an agile toolkit*. Addison Wesley.

Van Tilborg, A., Acting DUSD(S&T), (2006); "Advancing Software-Intensive Systems Producibility: Charge to NRC Committee," September 27, 2006 (citing Defense Acquisition University as source).

Principles behind the agile manifesto, http://agilemanifesto.org/principles.html. accessed on 8/4/2009.

BIOGRAPHIES

Dr. Barry Boehm is the TRW professor of software engineering at the University of Southern California; and the Director of Research of the DoD-Stevens-USC Systems Engineering Research Center (SERC). He was previously in software engineering, systems engineering, and management positions at General Dynamics, Rand, TRW, and DARPA, where he managed the acquisition of more than $1 billion worth of advanced information technology systems. Dr. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation. He is a Fellow of INCOSE, ACM, AIAA, and IEEE, and a member of the US NAE. Contact him at boehm@usc.edu.

Dr. Jo Ann Lane is a Research Assistant Professor at the University of Southern California. She is currently working on process and cost models for system of systems engineering and studying the application of lean principles in the SoS engineering environment. Prior to this, she was a key technical member of Science Applications International Corporation's Software and Systems Integration Group responsible for the development and integration of software-intensive systems and systems of systems. Contact her at jolane@usc.edu.

Supannika Koolmanojwong is currently a PhD student at the University of Southern California Center for Systems and Software Engineering. Her primary research area focuses on software process improvement especially in Architected Agile, NDI-Intensive, and Net-Centric Services areas. Prior to this, she was a lecturer at Assumption University, Thailand and a RUP/OpenUp Content Developer at IBM Software Group. Contact her at koolmano@usc.edu.

Dr. Richard Turner is a Distinguished Service Professor at Stevens Institute, a Visiting Scientist at the Software Engineering Institute of Carnegie Mellon University and a respected researcher and consultant with thirty years of international experience in systems, software and acquisition engineering. Dr. Turner is co-author of three books: Balancing Agility and Discipline: A Guide for the Perplexed, CMMI Distilled, and CMMI Survival Guide: Just Enough Process Improvement. Contact him at rturner@stevens.edu.