Relating Agile Development to Agile Operations

Rick Dove
Stevens Institute of Technology
Hoboken, NJ, USA
rick.dove@stevens.edu

Garry Turkington
Paradigm Shift International
Columbia, MD, USA
garry.turkington@gmail.com

Abstract

Agile Enterprise and Agile Software Development are concepts with some considerable exposure in the language. Agility in the enterprise context usually refers to a system operational characteristic, while agility in the software context usually refers to a system development characteristic. Both are concerned with systems that must deliver satisfaction in an environment characterized by uncertainty and change. Our interests are not with either specifically, but rather with the general concept of a system life cycle, for any type of system, that must deal throughout with uncertainty and change. The work described in this paper began with the observation that Agile Development processes generally ignore the agility of the resultant system; yet they exhibit a solid foundation in certain generic architectural concepts that have been shown applicable across all types of systems that would be or are agile. The investigation established certain commonalities across a predominate variety of Agile Development processes, then cast these common concepts in the domain independent Response Ability architecture that grew from work at the Agility Forum, and finally modeled how this architectural characterization of development could be seamlessly extended throughout the remainder of the system life-cycle, consistent with observed characterizations of operational agility. The result promises a graceful transition from development into and through operational phases, with a single stable process architecture throughout.

Introduction

The work reported here was inspired by a draft working paper (Turkington 2007) on agile-systems engineering and questions of its relationship to agile systems-engineering – the hyphenation making the distinction. Specifically, the relationships in question were those between Agile Software Development (ASD)[1] concepts, representing agile systems-engineering, and Response Ability Principles (RAP)[2], representing agile-systems engineering. ASD and RAP are employed for convenience of reference in this paper, not as acronyms.

RAP encompasses a set of domain independent architectural principles of structure and strategy that enable systems agility. ASD is domain focused on delivering customer satisfaction in situations of uncertain and changing software-system requirements. Central to both is the recognition that a system must continuously adapt to a changing and uncertain environment. Common to both is their employment of the word agile with meaningful intent.

[1] Agile Software Development is given definition and characterization by Manifesto for Agile Software Development (Beedle et al 2001) and accompanying 12 Principles.
[2] Response Ability Principles encompass research initiated at Lehigh University's Agility Forum in the '90s and later detailed in *Response Ability* (Dove 2001).

In 2006 the increasing interest in agility as a desired enterprise and system characteristic spawned a four-course graduate certificate in Agile Systems and Enterprises at Stevens Institute of Technology, and initiated related research. The Stevens program emphasizes fundamental domain independent architecture and design principles that enable agility in systems of any kind. ASD processes are included in case study work of domain-specific agile systems.

RAP and ASD have never claimed any common ancestry or inspiration, though both feature agility as a core concept. Each took shape independently. Bringing them together in a study of systems agility invariably raised the question of compatibility. Do the domain specific ASD processes fit the domain independent RAP model of an agile system – rephrased – Can domain dependent ASD concepts be cast as domain independent RAP architecture?

ASD processes appear on the verge of accelerated employment – one indication being the heated debates (Schwaber 2002) and rationalizations (Boehm and Turner 2004) over reconciling with CMM(I). Nevertheless an undercurrent of arguable ASD limitation is prevalent in the lore and literature, with themes such as inadequate progress visibility, can't be scaled for large projects, inappropriate for distributed teams, inappropriate for mission- and life-critical projects, and requires high expertise and talent. These concerns, entrenched CMM(I) cultures, and differing process-specific varieties, appear to make the decision to embrace ASD difficult for many. Casting ASD as a RAP architecture should provide a new viewpoint for understanding and evaluation, and is the basis of a second question. Could ASD cast as RAP architecture help reveal useful values and concepts?

These initial questions become all the more interesting when it is understood that ASD is about the agility of the development process for creating software, and not about the agility of the resultant software system itself. ASD takes its need for agility from the premise that requirements are uncertain and subject to change, and the process therefore needs a fluid latitude for discovery and convergence. Though directly ignored by ASD, requirements don't cease changing when delivery is accepted and the development phase is ended. Borrowing from (Boehm and Turner 2004):[3] "Agile methods concentrate on delivering a specific software product, on time, that completely satisfies a customer. As such, the scope of concern is focused on the *product at hand* and generally ignores problems that may occur later." Their intent for this statement was different, but it is equally applicable here. This begs another question: Can domain dependent ASD concepts cast as domain independent RAP architecture provide a path for consistent agile response ability in both development and operational phases?

In summary, three questions drove the work reported here:

1. Can domain dependent ASD concepts be cast as domain independent RAP architecture?
2. Could ASD cast as RAP architecture help reveal useful values and concepts?
3. Could ASD cast as RAP architecture provide a migration path for extending development-phase agility into the operational phase?

Though 17 independent leaders managed to meet and agree on a 4-point manifesto (Beedle et al 2001) that gave legs and a name to the ASD movement, each has a deeply personal and strong procedural framework explained in multiple books for how agile development should occur. Published discussion from formal to blogish is largely a detailed focus on brand differences, and on the unacceptable clash with traditional planned sequential methods.

[3] Boehm and Turner 2004, pg 29

We will not enter that fray here, but instead hope to re-focus the ASD characterization on fundamentals, align those fundamentals with general RAP theory, and suggest that contrary to the belief of some, ASD processes do maintain a consistent architecture, and that progress toward completion can be monitored. We will show that this architecture is what enables ASD processes to deal with change, and that an altered strategic approach can extend the same architecture into operations and deliver the benefits of agility throughout an extended life cycle.

The next and second section describes the sources and nature of the tools and information employed in seeking answers to these questions. The third section describes the process and results of pursuing these questions. The fourth section discusses the results and implications. The fifth and final section presents some conclusions and suggests further work illuminated by this effort.

Experimental Setup

Answering the three questions was conducted much like an experiment. The outcome was not at all sure in the beginning, a literature search compiled characteristics of ASD processes and filtered for common fundamentals, some trial runs on concept mapping between ASD and RAP were performed to see if a fit was likely, and a suitable set of RAP modeling concepts was selected. This section will describe the source and nature of the modeling tools and the source and nature of the process characteristics chosen for the experiment. The next section will describe the mapping and casting processes.

RAP Modeling Tools. Agility as a system behavior is defined here in the broad terms intended by the 1991 Lehigh study (Nagel et al 1991, Dove 1992, Goldman et al, 1994) that put the word into play: effective response under conditions of uncertainty. Subsequent research recognized that practicing such agility requires at least three aspects: situational awareness, decisive choice making, and the ability to respond. This latter aspect appeared to be the principle stumbling block observed in enterprise systems at that time. This drove a research focus at The Agility Forum on discovering architectural structure and strategy principles that would enable highly adaptable enterprise systems. This aspect of agile systems was later dubbed Response Ability in (Dove 2000), and is detailed in domain independent terms in (Dove 2001). The work here deals only with this third architectural aspect of agile systems.

RAP based architecture currently encompasses seven thought-guiding frameworks: response requirements categories (2x4 elements), response performance metrics (4 elements), functional design principles (10 elements), design quality principles (3 elements), system integrity responsibilities (4 elements), an overarching architectural philosophy (3 elements), and a conceptual pattern. All except the recent addition of design quality principles are dealt with at length in (Dove 2001) and perhaps more accessibly and briefly in (Dove 2005).

The conceptual pattern of RAP architecture employed here[4] is one of drag-and-drop modules in a plug-and-play infrastructure. The overarching architectural philosophy is reusable modules reconfigurable in a scalable infrastructure. Modules are encapsulated one-to-one physical and functional units. Infrastructure has two parts: a passive part that provides standards (rules) for connectivity and interaction among modules, and an active part that consists of four specific in-

[4] (Dove 2007) distinguishes agile systems as reconfigurable (class 1) and reconfiguring (class 2). The drag-and-drop/plug-and-play pattern is associated most readily with class 1 (reconfigurable) agile systems, as the four integrity responsibility elements of the active infrastructure are typically peopled rather than systemic.

tegrity responsibilities for maintaining and sustaining readiness for unpredictable system response needs.

Integrity responsibility, the "active" part of the infrastructure, will play a key role in question number 3's extension of agility from the development phase into the operational phase. The four integrity responsibility elements are:

1) maintaining sufficient inventory of modules ready for use (development people, team leaders, engagement procedures, reusable code modules, reusable test suites, etc),
2) new module addition and upgrade as new capabilities are needed (new developer skills, newly developed code modules, new test suites for new code, new procedures as indicated by a changing situation, user representatives intimate with next stage feature development needs, etc),
3) infrastructure evolution (improvements to existing rules and standards, new rules and standards, etc), and
4) assembly of modules into on-demand system configurations suitable for changing response needs (successive iterations in the development process).

Three of the seven RAP frameworks are employed in the work reported here: the 3-element architectural philosophy, the 4-element integrity responsibilities, and the plug-and-play/drag-and-drop conceptual pattern. Thus, ASD characteristics will be cast in the conceptual pattern of drag-and-drop encapsulated modules in a plug-and-play infrastructure of passive rules and active responsibilities. Exploring application of the other frameworks appears promising, but is left to subsequent work.

ASD Characteristics. Agile Software Development consists of a family of processes with distinctly different approaches but certain underlying common themes. These common themes distinguish them from the traditional family of development processes often referred to as plan-driven.

At core, plan-driven and ASD processes differ on the nature of system requirements and how that affects their determination. Though many more differences are hotly debated, such as the timing and weight of documentation, the stability of architecture, the predictability of outcome, the necessary level of talent, and the scalability of process, all stem from the root of requirements determination. Plan-driven is predicated on providing external management and evaluation visibility through mechanisms such as documenting the development plan for agreement before commitment, and scheduling milestones for measuring progress toward completion.

Plan driven processes establish up front, for the most part, what must be done and how it will be done, and document this as the development plan. Agile processes, in contrast, are founded on the belief that requirements will necessarily reveal themselves during development, so a comprehensive document specifying what will be developed cannot be generated up front. At heart, the issue centers around the timing and characterization of requirements specification. Recent suggestions by (Boehm and Turner 2004) and others favor a recognition that both plan driven and agile methods have merit when they are matched with a compatible project need, and that every project can likely benefit from a custom blending of the two extremes. We are not addressing these issues in this study.

Succinctly captured in the Agile Manifesto of (Beedle et al 2001), ASD coalesced around perceived failings of traditional approaches to software system development. Advocates saw lengthy and formal requirements capture activities as inefficient at best, counter-productive at worst.

ASD advocates believe that up-front fixed requirements capture is flawed because the understanding of requirements evolves and changes. Evolution occurs as early projected understandings progressively gain feedback from development, testing, and usage experience. Change occurs throughout as the external situation which defines requirements continues to change. Both conspire against the shelf life of fixed snapshot articulation.

This viewpoint resulted in a number of processes often referred to as agile methodologies. These fall into two main types, those such as Extreme Programming (Beck 2000) and Feature-Driven Development (Palmer et al 2002) which present processes for actual software development; and those such as Scrum (Schwaber et al 2003) and Agile Unified Process (Ambler 2005) which focus more on the project context within which development occurs. Common across all the methodologies is a belief that requirements sufficient to create satisfaction must be discovered, that discovery will change the priorities, values, and activities of development as they are revealed, and that convergence on total customer satisfaction within time and budget constraints should be the driving objective of the development process.

We are interested in characterizing ASD in terms of resources and the rules (process standards) that constrain and enable the engagement and interaction of those resources.

Our first attempt to identify common process standards across ASD methods stemming from this core of requirements-uncertainty proposed five rules:

1. Let architecture emerge. The changing understanding of the requirements for satisfaction may be incompatible with a fixed initial architecture. Some approaches, for instance, don’t start with an architecture, but rather with the intent to confirm isolated important feature concepts through customer tests that will help shape an appropriate architecture.
2. Deliver early and often. The time between a requirement identification and its confirmation by customer use should be minimized. The system will be developed iteratively and delivered incrementally, with an aim to continuously increase functionality and customer satisfaction.
3. Employ frequent customer input. Customer involvement is key. Through exposure to the latest delivered iteration the customer can refine, augment, and replace guiding requirements as interaction with results reveal misunderstandings and new understandings.
4. Build for change. Implicit in the process is the expectation that continuous requirements change must be reflected in and supported by continuous system change, development process change, developer team change, and a general ability to fit the next activity and goal in all respects to the latest understandings. This expectation shapes the nature of ASD processes to minimize cost and lost effort associated with change.
5. Empower development teams. Because the ability to respond quickly and accurately is paramount there is no room for gratuitous bureaucracy and rigid organizational hierarchies. Team members are encouraged to accept as much delegated responsibility as possible and to resolve issues at the lowest practicable organizational level. Effective communication and decisive individual responsibility are important means to harness change beneficially.

A search of the literature reveals similar attempts to characterize ASD by common concepts across ASD methods (Boehm and Turner 2004, Cockburn 2005, Coffin et al 2006, Highsmith 2002, Strode 2005, Theunissen 2003). No real contradictions were found among any of them. Differences appear to be the level of abstraction, or put another way, concepts in one that are inclusive of concepts in another.

Upon reflection, rather than have this work predicated on yet another mincing of concept categorization begging justification, we choose to adopt the four abstractions concluded by (Boehm and Turner 2004), which seem both comprehensive and succinct:

> "A truly agile method must include all of the following attributes:
> iterative (several cycles),
> incremental (not deliver the entire product at once),
> self-organizing (teams determine the best way to handle work), and
> emergence (processes, principles, work structures are recognized during the project rather than predetermined)."[5]

Our interest here is not on the precise nature of common procedural characteristics across ASD processes, but rather where suitable abstractions of those characteristics might fit into a common process architecture – one which doesn't emerge or change as development proceeds, but remains stable throughout.

(Cockburn 2005) addresses the employment of abstractions (properties in his case) as ASD guidance over procedures: "The procedures may not produce the properties. Of the two the properties are the more important. Other procedures than the ones I choose may produce the [same] properties for your particular team."[6]

Procedures and Results

This section presents the procedures and results in pursuing each of the three questions.

Pursuing Question 1: Can domain dependent ASD concepts be cast as domain independent RAP architecture? To answer this question we explored three paths: 1) how similar are the meanings of agility in ASD and RAP, do ASD concepts and principles map clearly into RAP concepts and principles, and is ASD agility enabled by a RAP architecture?

First comparing the meanings of agility in ASD and in RAP for congruence, we look at words from two prominent members of the ASD community. Kent Beck explaining the essence of his Extreme Programming methodology: "XP is a...methodology for...developing software in the face of vague or rapidly changing requirements.[7]" James Highsmith explaining the essence of the methodology he is known for: "Adaptive Software Development has five primary goals [defining] the essence of building better software in a world where high speed, change, and uncertainty are key characteristics of its intensifying complexity.[8]" Wrapping it up for the community at large, Highsmith states elsewhere: "Agility is the ability to both create and respond to change in order to profit in a turbulent business environment."[9] Two people and three quotations are offered here as representative indications of how the word agility is characterized across the greater ASD community. This characterization is identical to the RAP employment of the term, said in as many different ways as found in ASD, but succinctly stated earlier as: "effective response under conditions of uncertainty." Highsmith's recognition of response to change having both reactive and proactive elements is a core concept throughout the history of RAP development. Agility has a remarkable consistency of meaning for both ASD and RAP considering the plethora of vague and different interpretations given the word elsewhere in systems literature.

[5] Boehm and Turner 2004, pg 17
[6] Cockburn 2005, pg 17.
[7] Beck 2003, pg xv Preface.
[8] Highsmith 2000, pg xxiv Preface.
[9] Highsmith 2002, pg xxiii Preface.

Next we attempted various mappings of principles and concepts between the two. This proved to be problematic and reached no useful end. Not because contradictions or discontinuities surfaced, but rather because the articulation of objectives, principles, and practices in one mapped inconsistently across the stated objectives, principles, and practices of the other. Clean one-to-one mappings were not to be found.

The initial failed mapping attempt drove a search for an alternate integration mechanism, which led to the mapping of ASD characteristics onto RAP architecture, as depicted in Figure 1. Space limitations here do not permit a textural explanation of figure one so the reader is left to examine it directly.

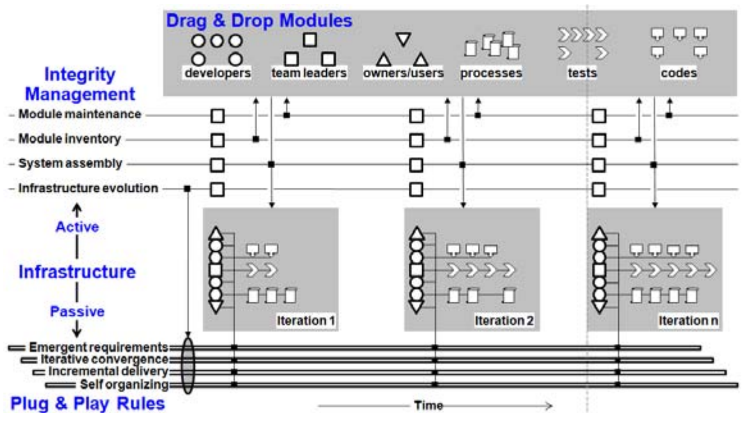

Figure 1: Mapping ASD Characteristics onto RAP Architecture

Pursuing Question 2: Could ASD cast as RAP architecture help reveal useful values and concepts? A diligent exploration of this question must be left for another place with pages to spare. Here we will simply suggest that the depiction in Figure 1 is that of a stable architecture, one that does not change as requirements are discovered which cause a rethinking of necessary functions and functional interactions. This architectural depiction also cuts through all of the various choices among different ASD processes and facilitates the custom configuration of an ASD process fit to the project and fit to the culture. Stability of architecture and perplexity of choice are two problems that may be mitigated with this RAP view.

Pursuing Question 3: Could ASD cast as RAP architecture provide a migration path for extending development-phase agility into the operational phase? ASD is predicated on the continuous change of requirements. Requirements change for at least two primary reasons: initially vague or ambiguous needs become crystal clear in necessary nuance as solutions are tested and employed, but more perniciously, aspects of and in the environment the system must serve and compete with continue to change independently. This latter type of change does not stop with delivery. It's effect on the ASD objective of optimal customer satisfaction begins virtually the day after delivery celebration.

Figure 2 depicts the three phases of development, commissioning, and operation in a purposely but not contrived mirror image of Figure 1's iterative development process. Nothing fundamentally changes with delivery: new modules must be built when new requirements surface, and existing modules must be upgraded when they cease to perform effectively due to requirements change.

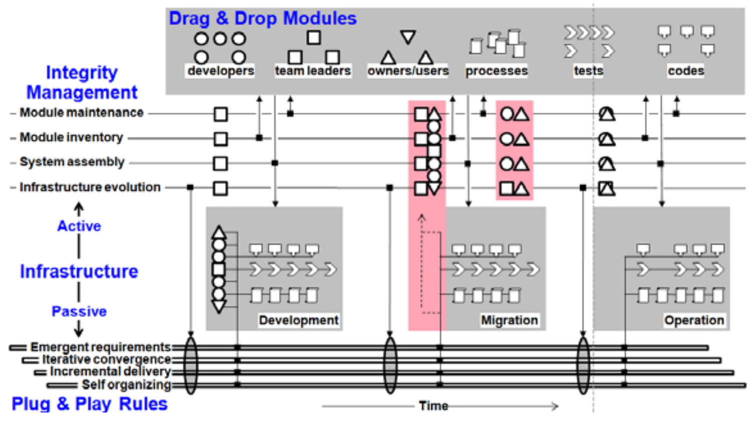

Figure 2: Graceful Migration from Agile Development to Agile Operations

It matters not if the modules in question are people with certain skills, code that executes on a computer, tests that verify a new addition doesn't cripple prior capability, procedures that discipline the system sustainment activity, and so on. What does change is a lessening of development intensity, and an adjustment in who carries out the responsibilities for maintaining and sustaining system integrity.

The commissioning phase depicted in Figure 2 is labeled as migration. It is meant to signify a smooth and natural transition. Suggested in this depiction is the concept that owner/users, so vitally central in ASD as active participants, evolve toward greater and more intimate operational responsibilities as development migrates into commissioning. This would mean that the customer representatives chosen for frequent requirements-verification at development time are selected with an eye to their eventual and longer term operational responsibilities. The depiction indicates that they gradually take on front line responsibilities with development people playing a supporting rather than front-line role in the operational phase.

Discussion and Implications

Does it work? Chapter 8 of (Dove 2001) includes an in-process (at that time) description of an agile ERP system design and a plan for implementing it with a process for development and operations that mirrors the suggestion here, including the employment of owner/users during development that would go on to become the integrity managers of the system. (Dove 2005) written after the fact discusses the actual results of that design and implementation process, with lessons learned and noted concerns about long term sustainment. The design, strategy and process carried out at that time did not have knowledge of the ASD processes referenced here, and largely unfolded naturally from the knowledge of domain independent agile systems concepts. It was a success in all respects, under management that demanded an agile ERP operating result.

ASD proponents stress "simple design" concepts that meet only what is explicitly needed at the time. Stated in various ways are the instructions to avoid anticipating future needs in the work being done at the immediate moment. The authors wish to clarify our interpretations of those instructions, and suggest that they are not a prohibition on explicitly enabling ease of continuous change in the future, but rather are aimed at surmised or anticipated future functional needs which should not be attended to until those functional needs become irrefutable. Thus, there is no contradiction of principles to suggest that ASD should specifically enable operational agility.

Furthermore, the drag-and-drop/plug-and-play reconfigurable patterns depicted here are consistent with actual ASD practice and do not represent additional project cost to realize the long term continuation benefit in the operational phase. Realizing the benefit is a matter of selection criteria in choosing appropriate user/owners for development-phase participation. We do not mean to make light of the difficulties associated with gaining any kind of useful owner/user participation, only to illuminate necessary selection criteria consistent with the customer's real need for an ROI during the operational phase.

With these concepts in mind, it is suggested that a strategy for continuing the ASD approach pattern as depicted here, into operations, will lower the barriers for gaining ASD acceptance. ROI in the operating phase is a requirement of all systems, whether explicitly recognized in system requirement "shall" statements or not, and savvy management responds to value propositions that provide reasons for believing ROI will occur. At today's technology pace a system that cannot incorporate new technology or respond to new competitive needs and customer demands becomes a liability often before ROI is realized. Corporate management and government acquisition are both showing increasing sensitivity to this reality.

Concluding Remarks

RAP includes a design ethos that a “good” system satisfies three quality principles: Requisite Variety, Parsimony, and Harmony. These are part of the art and practice. Requisite variety would seem to be the driving force behind ASD: the development process must be able to respond to requirements as they become evident and as they change – and it must be proactive in discovery, reactive in accommodation. Parsimony is seen throughout the ASD varieties, especially in minimal documentation, deferring action on “possible” future needs, and “lean” avoidance of anything that doesn’t add immediate value. Harmony has been the rub: ASD appears in contradiction with acceptable best management practices as measured by plan-driven values.

ASD receives some common disharmonious pronouncements: No stability, as the architecture is vague and subject to whimsical change; and no way to measure progress, as the goals keep changing. The RAP view depicted here provides a solid unchanging basis to the ASD processes that should counter the lack of stability perception. As to the need for visible progress measurement, the core metric can only be the nature and speed of convergence on customer satisfaction – an issue we’ve not addressed overtly here.

It is regrettable that space limitations do not permit a deeper discussion of results, implications, and revelations. The work reported here is inspiring a number of additional projects that will continue, and augment this beginning. For an obvious one, a fuller treatment of what was learned and what is implied is indicated.

ASD is seductive yet perplexing to many who are having difficulty understanding what it is really about. Suggestions to mix and match procedures from one ASD process variety with another, and fold in what seems right from plan-driven approaches, abound in the literature. When this suggested custom-configure-process advice is employed by the very savvy, it works; but the target audience of the advice are not the savvy. A new level of understanding is needed.

RAP offers ten design principles that might help fill this need. For instance, pair programming is a well known XP procedure, where one programmer writes code as another looks over the shoulder. Simply plopping this concept among some other borrowed procedures into a process strategy will not likely deliver the promised benefits. Two working like this as a team deliver value through the synergy of redundancy (two of them) and diversity (with complimentary but not identical thinking). The savvy know this and configure pairs accordingly. Those reading the advice literature need to learn this. RAP provides a framework for this elucidation. Redundancy and Diversity is one of RAP’s ten design principles, and would be employed as one of the fundamental principles behind the pair-programming tactic. This path begs more exploration and will be pursued in future work.

With earned respect we have referenced and quoted from both (Boehm and Turner 2004) and (Highsmith 2002) to underscore shared beliefs and relate the work reported here to voices that may be respected and familiar to the reader. We do not agree, however, with what appears to be a core underpinning for (Boehm and Turner 2004) as stated on page 1: “Agility is the counterpart of discipline.” We hold, instead, that agility is obtained only through discipline, and is a fleeting competency if that discipline is not vigilantly maintained by the four integrity responsibilities. (Highsmith 2002) on page xxxi tributes this same contrast somewhere in the middle: “Agility means balancing between structure and flexibility, so rigor is a vital part of any development process. Agility focuses on the flexibility side of the definition and rigor focuses on the structured side." We suggest the "focus" is a temporary artifact of introducing ASD into a plan-driven world, and that ASD as espoused by its leaders is extremely rigorous.

This investigation was about agility in both development and operational phases of a system sharing a common architectural pattern. ASD provided a concrete working-example in the software domain. The authors believe the conceptual pattern application is domain independent, has implications on system life-cycle models, and are pursuing further work in this direction.

References

Ambler S., Agile Unified Process, http://www.ambysoft.com/unifiedprocess/agileUP.html, 2005.

Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000, ninth printing July 2003.

Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Schwaber, K., Sutherland, J. and Thomas, D., with additional signatories Beck, K., Grenning, J., and Mellor, S., *A Manifesto for Agile Development* (http://www.agilemanifesto.org) and "Principles behind the Agile Manifesto" (http://www.agilemanifesto.org/principles.html), Agile Manifesto, 2001

Boehm, B. and Turner, R., *Balancing Agility and Discipline – A Guide for the Per*plexed, Addison-Wesley, 2004.

Cockburn, Alistair, *Crystal Clear – A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2005

Coffin, R. and Lane, D., "A Practical Guide to Seven Agile Methodologies," *Part 1, XP, Scrum, Lean, and FDD* (http://www.devx.com/architect/Article/32761/) and *Part 2, AUP, Crystal, and DSDM*, http://www.devx.com/architect/Article/32836, 2006.

Dove, R., "What's All This Talk About Agility – The 21st Century Manufacturing Enterprise Strategy", Prevision, Japan Management Association, 1992, pre-translation English http://www.parshift.com/Files/PsiDocs/Rkd92Art6.pdf

Dove, R. *Response Ability – The Language, Structure, and Culture of the Agile Enterprise*, Wiley, 2001

Dove, R., "Fundamental Principles for Agile Systems Engineering," Conference on Systems Engineering Research (CSER), Stevens Institute of Technology, Hoboken, NJ, March 2005

Dove, R., SDOE 678 and 683 course materials, Stevens Institute of Technology, School of Systems and Enterprises, 2007.

Goldman, S., Nagel, R., Preiss, K., *Agile Competitors and Virtual Organizations*, Van Nostrand Reinhold, 1995.

Highsmith, J., *Adaptive Software Development: A Collaborative Approach to Managing Complex Software*, Dorset House Publishing, 2000, pg xxiv Preface.

Highsmith, J., *Agile Software Development Ecosystems*, Addison-Wesley, 2002.

Nagel, R., Dove, R., Goldman, S., Preiss, K., *21st Century Manufacturing Enterprise Strategy, Volumes 1 and 2*, Coolingdale, PA: DIANE Publishing Company, 1991.

Palmer S. and Felsing J., *Practical Guide to Feature-Driven Development*, Prentice Hall, 2002.

Schwaber K., Cohn, M. and Darby, E., 2003, http://www.scrumalliance.org.

Schwaber, K., "Agile Processes: Ken Schwaber's letter to IEEE Computer", Jeff Sutherland's Scrum Log, June 16, 2002, http://jeffsutherland.com/scrum/2002/06/agile-processes-ken-schwabers-letter.html

Strode, D.E., *The Agile Methods: An Analytical Comparison of Five Agile Methods and Investigation of Their Target Environment*, Masters Thesis, Massey University, New Zealand, 2005, http://digitalcommons.massey.ac.nz/context/dissertations/article/1048/index/1/type/native/viewcontent

Theunissen, W.H.M, *A Case Study Based Assessment of Agile Software Development*, Masters Thesis, University of Pretoria, 2003, http://upetd.up.ac.za/thesis/available/etd-07152004-084708/unrestricted/00dissertation.pdf

Turkington, G., "Agile Development of Agile Systems," SDOE 678 working paper, November 2007, www.parshift.com/AgileSysAndEnt/WrkPap/WrkPap678Turkington070716.pdf

Rick Dove was co-Principle Investigator on the 1991 OSD/Navy-funded, industry-led study that identified and defined enterprise agility as a critical pending need. As Director of Strategy for the subsequent DARPA/NSF funded Agility Forum he organized and led the development and execution of an industry-collaborative research agenda to identify the nature of agile systems and enterprises. He is author of two books dealing with agile systems and decision making. Currently he is co-director of the Agile Systems and Enterprise graduate certificate at Stevens Institute of Technology with course development and teaching responsibilities, and pursues research in the area of self-organizing systems-of-systems. He has 25 years of start-up, turn-around, and interim executive management, including responsibilities for most C-level positions, R&D, and major enterprise IT projects. He holds a BSEE from Carnegie Mellon University and did graduate studies at UC Berkeley in computer science.

Garry Turkington earned his BS and Ph.D degrees in computer science from the Queens University of Belfast, Northern Ireland. Since 1999 he has worked for the United Kingdom civil service in postings in both the UK and the USA. His work has focused on the design and implementation of large-scale distributed systems and the establishment of system engineering processes. His research interests are in systems and organizational agility, systems of systems, networks and distributed computing.

Acknowledgement: Dr. Turkington wishes to thank the United States Department of Defense for funding that assisted in this research.