Software Engineering Institute

CMMI® or Agile: Why Not Embrace Both!

Hillel Glazer, Entinex, Inc.
Jeff Dalton, Broadsword Solutions Corporation
David Anderson, David J. Anderson & Associates, Inc.
Mike Konrad, SEI
Sandy Shrum, SEI

November 2008

TECHNICAL NOTE
CMU/SEI-2008-TN-003

Software Engineering Process Management

Carnegie Mellon

Table of Contents

Acknowledgments

The authors thank the following individuals for their significant contributions to this report: Mike Phillips, Sabine Canditt, Noopur Davis, Bill Peterson, Mary Van Tyne, and Barbara White. We greatly appreciate the insights and efforts of all of these individuals.

In addition, the authors would like to recognize those who attended Consultant's Camp 2007. Table 1 in Section 10 of this report builds on work done in a session at that event.

Abstract

Agile development methods and CMMI (Capability Maturity Model® Integration) best practices are often perceived to be at odds with each other. This report clarifies why the discord need not exist and proposes that CMMI and Agile champions work toward deriving benefit from using both and exploit synergies that have the potential to dramatically improve business performance.

1 Problem Definition

Agile development methods and CMMI best practices are often perceived to be at odds with each other. If these perceptions or their causes are not resolved, we are likely to see more confusion and conflict as the adoption of each approach increases. In addition, each approach includes principles of good software development often overlooked but needed by the other approach thus being knowledgeable in both is important to project success. In the long term, this discordant situation is not healthy for the software engineering profession.

Why the discord between Agile and CMMI camps? The purpose of this report is to clarify why the discord need not exist and to ask for your help in making the software development community aware that, when properly used together, CMMI and Agile can dramatically improve performance.

We believe there are two primary reasons for the discord between the Agile and CMMI camps:

1. Early adopters of both CMMI and Agile methods represent extreme examples of their software development paradigms. Early CMMI adopters were developers of large-scale, risk-averse, mission-critical systems, often with high levels of management oversight and hierarchical governance; whereas the early adopters of Agile methods generally focused on smaller, single-team development projects with volatile requirements in a software-only environment. These two extremes set the tone for all that followed.
2. The inaccurate information about CMMI and Agile and the misuse of both resulted in misperceptions in both camps about the other. These negative perceptions that position CMMI and Agile at odds with each other arose largely from the following factors:
    a. Misuse—two decades of experience, first with the Capability Maturity Model (CMM®) and then with CMMI models, in which practices were sometimes misused or applied to (i.e., overlaid on) development activities that may have already been perceived by software development teams as productive without them
    b. Lack of Accurate Information—a dearth of accurate information about CMMI in the Agile community and the corresponding dearth of accurate information about Agile methods in the CMMI community
    c. Terminology Difficulties—the use of terminology in CMMI (e.g., *discipline*, *quality assurance*, and *predictability*) and Agile methods (e.g., *continuous integration*, *test-driven development*, and *collective code ownership*) that carries context-specific connotations and is thus easily misunderstood and abused
    d. Top-Down Versus Bottom-Up Improvement Approach—the introduction of an approach that sometimes favors one "voice" (i.e., management versus practitioner) over the other, which neglects the other important voice in how to effectively run the business

In this report, we identify and discuss the misperceptions we consider most damaging to a correct understanding of both paradigms, and suggest ways to improve our understanding of both of these powerful approaches for effective software development. In hindsight, we acknowledge that the way in which CMMI was developed and introduced may have helped cause some users to misunderstand the true message and nature of CMMI[1]. Such misunderstanding may have led to inconsistent and ineffective use of CMMI.

Further, we identify common misperceptions in the Agile community about CMMI and common misperceptions in the CMMI community about Agile. In many ways, these misperceptions are related to the misuse of CMMI and Agile, but misperceptions can also be attributed to a shortage of accurate information as well as a persistent belief in notions and experiences that are not part of either approach.

Some misperceptions of CMMI in the Agile community stem from aspects of the CMM that are no longer found in CMMI. CMMI includes many improvements[2] that differentiate it from the CMM, which has not been updated since 1993[3]. Some in the Agile community use CMM concepts to judge CMMI unfairly. For example, incorrectly referring to the goal of maturity level 2 as creating *repeatable* processes persists to this day. In some recent posts to a CMMI-related online forum, a few participants have admitted to not knowing CMMI like they know the CMM. Several participants continue their arguments by writing, "…but if CMMI is anything like CMM, then…"

To complicate the problem, there are CMM users who never upgraded to CMMI and therefore persist in using a model that has a less flexible view of software and systems development than the newer CMMI models. There are also former CMM users who upgraded to CMMI but persist in holding on to some of the more dated views from the CMM.

An important aside is that when practitioners are naming their activities, the labels "CMMI" and "Agile" are often applied too freely when practitioners are not following either approach properly. These situations contribute to negative perceptions of both approaches.

We now know that CMMI and Agile can be used together successfully. Some of the references at the end of this report include experience reports about the successful use of these approaches together. The Software Engineering Institute (SEI^{SM}) continues to be interested in the development of Agile methods and in community experiences with both CMMI and Agile.

1 We can't speak for the Agile community given its diversity; but we feel that to some extent the same could be said for Agile methods.

2 Example improvements include more attention to risk management, integrated teaming, and the work environment and less attention to starting with a fixed set of requirements and doing things "according to a documented procedure."

3 An incomplete draft of CMM version 2 that included some of these same improvements was created in 1997 but was never formally released. Instead, it served as one of several sources for the initial draft of CMMI.

2 Origins from Two Extremes

The perceived incompatibility of Agile methods and CMMI best practices can, in large part, be attributed to their different origins. In this section, we describe the two extremes from which each approach sprung: Agile from fast-moving markets consisting of small organizations and CMMI from contractually-driven markets consisting of large organizations. These are generalizations, but when you see the details about how each was created, you begin to see why they approach software development from different perspectives.

2.1 THE ORIGINS OF AGILE METHODS

The cornerstone of Agile methods originated long before the World Wide Web and collaborative technologies (e.g., wikis and instant messaging). This cornerstone is iterative and incremental design and development (IIDD), a method adopted by engineers over 75 years ago[4].

Early adopters of IIDD included Department of Defense (DoD) engineers who engaged in propulsion-related research and development, which included engineering activities tied to hardware not software. An early progenitor of IIDD was Dr. W. Edwards Deming who began promoting Plan-Do-Study-Act (PDSA) as the vital component of empirical engineering. Early adopters of Deming's teachings in the aerospace industry include NASA (National Aeronautics and Space Administration**)** and the US Air Force, each of which developed entire systems using time-boxed, iterative, and incremental product development cycles.

As early as the mid-1950s, IIDD was used in software development resulting in business benefits such as "avoiding management discouragement" and "increasing customer satisfaction." In fact, a large number of early software development projects, which were often experimental and explorative, shared many of the attributes of today's Agile methods.

However, in a systems world dominated by mainframes, COBOL (Common Business-Oriented Language), and the demand to process large and complex datasets, procedural top-down design and development methods dominated and were perceived by many to be the standard. This situation was influenced by the procedural nature of DoD standards and the proliferation of fixed-price contracts[5] awarded to suppliers of complex DoD systems (the predominant consumer of computer software at that time).

In 1976, Tom Gilb argued that evolutionary development resulted in superior software delivery in his book *Software Metrics* and launched a movement toward agile, light, and adaptive development iterations that provided rapid results and more frequently visible business benefits.

4 This short history of Agile methods is provided for the benefit of those who might mistakenly believe that Agile methods are a recent innovation without deep conceptual roots. For a complete history, there are books that describe the events of the past seventy-five years that also contributed to the success of Agile methods.

5 By their nature, the parties to fixed-price contracts assume an unchanging project scope in unvarying development and use environments. This nature makes it difficult to later modify project direction (without high-ceremony activities) to take advantage of newly-discovered needs and constraints or new technologies.

As the state of software engineering matured, more formal applications of IIDD became available for example, in Barry Boehm's 1985 release of *The Spiral Model of Software Development and Enhancement*.

Throughout the 1990s, IIDD gained broad acceptance in the software community in various forms, including rapid prototyping, rapid application development (RAD), and rational unified process (RUP). The seeds of most modern Agile methods were sewn throughout this decade.

While most may not expect it, innovative and Agile methods began in the large information technology (IT) shops of several large companies, including an automotive manufacturer and an overseas bank. XP (eXtreme Programming) began at Chrysler Corporation in 1996 on a project staffed by IIDD advocates Ron Jeffries and Kent Beck, while feature driven development (FDD) started at United Overseas Bank in Singapore, one of Asia's largest banks. With *pair programming* and *refactoring* as XP's most celebrated features, XP became one of the most recognizable methods of the Agile family. Before the end of the decade, it was clear to many businesses and software engineers alike that in many settings, face-to-face communications, rigorous customer interaction, small rapidly moving teams, and frequent delivery of software ultimately produced superior software. This enlightenment was occurring simultaneously elsewhere and so-called *lightweight* methods proliferated with names such as *Scrum*, *Crystal*, *FDD*, and others.

With the proliferation of IIDD methods came the need to coordinate and compare these methods by those interested in their growth and sustainment. The result of this need was a "meeting of the minds" among leaders who were principally responsible for the theory and application of each method.

A group of leaders met, including Kent Beck, Ron Jeffries, Alistair Cockburn, Jim Highsmith, Bob Martin, Mike Beedle, Ken Schwaber, Jeff Sutherland, and others who represented the most successful of the new lightweight methods. Modeling their meeting after an earlier meeting of XP enthusiasts in Oregon the year before, these leaders gathered in the Wasatch Mountains of Utah to ski, relax, and ultimately author the *Manifesto for Agile Software Development.*

A subset of the *Manifesto* authors, together with others like Mary Poppendieck, went on to form the Agile Alliance, a not-for-profit organization dedicated to encouraging the adoption of agile methods. The Agile Alliance primarily focuses on organizing the Agile Conference in the United States every year.

While even the organizers of the Utah event expressed skepticism of its outcome, the sessions were a success. The *Manifesto* documented the guiding principles of Agile development and defined a philosophy around a set of existing methodologies. While the first *Manifesto for Software Development* focused on programming, three years later original *Manifesto* authors Jim Highsmith and Alistair Cockburn gathered a similar group of early Agile adopters, including David Anderson, Mike Cohn, Todd Little, and others to establish a set of six management principles known as the *Project Management Declaration of Interdependence (DoI)* [Anderson 2005b]. The 15 authors of the DoI subsequently formed the Agile Project Leadership Network (APLN), a not-for-profit organization dedicated to encouraging better leadership and management in the IT sector and software engineering profession.

The *Manifesto* and *Interdependence* publications, books written by their original authors, the formation of not-for-profit organizations to promote the Agile approach, and the widespread use of the internet for research by software practitioners, has resulted in the rapid growth and broad adoption of Agile methods throughout much of the software engineering profession. Some methods, most notably Scrum, continue to grow beyond the software industry into professions that desire the benefits provided by the same basic IIDD concepts first pioneered by Deming and his predecessors.

2.2 THE ORIGINS OF CMMI

Before receiving the "CMM" name, the first Capability Maturity Model-like framework was published in 1989 by Watts Humphrey in his book, *Managing the Software Process*[6]. A few years earlier, the U.S. DoD announced a request for proposals (RFP) to address the excessive amount of money being spent on software that was either never delivered or delivered late with little of its expected functionality.

The contract awarded on the basis of the RFP was to establish what we know today as the Software Engineering Institute (SEI) at Carnegie Mellon® University. The SEI brought together representatives from academia, research, and industry to expose and promote practices demonstrated to be successful at avoiding the failures so beleaguering DoD software acquisition efforts. Carnegie Mellon's framework of practices to address the DoD's issue became the CMM.

If we look at the genesis of the CMM, it predates the internet and nearly everything associated with internet technology. For that matter, CMM predates many software development, deployment, and infrastructure technologies, languages, and methods. We've all learned a lot in the past 20 years. When the DoD set out to address their "software dilemma," the software world was different than it is today.

To color this context even further, everyone working to develop the initial CMM was looking for the solution to the "software problem" from the perspective that software is a component of a larger system and that if it failed, lives would be lost (e.g., aircraft, ships, weaponry, medical devices). Systems were evolved using careful and deliberate development paths according to lower-risk, standardization-heavy and contractually-driven relationships between the developer and the customer.

In today's frequent discussions of increasing globalization and the important role played by *trust* (i.e., level of social capital[7]) [Fukuyama 1995] in making effective collaboration happen across stakeholders, one might describe such a development context as exhibiting *low trust*. Users were typically not direct contributors to the evolution of the end product prior to field testing. They instead had to depend on the contracting relationship, requirements, and standards to deliver the product they needed.

[6] This short history of CMMI focuses more on the past twenty years, but as in the case with Agile, the roots for many of the product development, project management, and process concepts found in CMMI have a long history.

[7] This definition of *trust* may be clearer but more elaborate: "the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party" [Mayer 1995]. For the purposes of this report, the particular question being asked is whether there is a level of trust between the project and its customer that will allow them to effectively negotiate scope as the project progresses, without the customer requiring detailed accounts of project effort.

These comments may be an over-generalization, but they are intended to summarize the DoD software acquisition environment that existed at the time. Further, these comments explain why the practices in CMMI sometimes exhibit some of these same *high ceremony* and *low trust* characteristics found in the high-risk, government-contractor environment in which software failure could equal lives lost.

Also, within the high-risk government-contractor environment at that time, the prevailing pattern of infrequent and monolithic deliveries contributed to the high costs associated with deployment, upgrade, and replacement (e.g., software embedded in fighter aircraft in the 1980s could not be upgraded over the air or over the internet). Hence, getting it right the first time was critically important. Furthermore, most of the organizations involved in this contractual environment were large organizations working on large complex projects.

Finally, the use of public money in government contracting (or similar high-visibility and high-stakes situations) requires a level of accountability by all those involved that often drives all parties toward risk-averse behavior bordering more on *protecting one's own interests*[8] than on *finding the most efficient win-win solution*. Ceremonial but perfunctory activities help address the often competing and incompatible self-interests of all parties, but make operating in an open and transparent manner challenging, and reinforce the perception of low trust[9].

Within a few short years, the CMM was expanded into several other models; these were *point solutions* developed to address non-software development projects. Also, the CMM and these variants increasingly became used internationally and by commercial industry. Organizations attempting to adopt more than one model on any given project quickly realized the challenges of doing so and petitioned the SEI to consolidate the various CMMs into one model, which in 1998 led to a joint industry, government, and SEI team to create CMMI.

Of course, over the years that the CMM and CMMI have been maintained, inputs on what constitutes good management and development practice have increasingly come from a wider variety of industries and from users around the world. As a result, new ideas, standards, and practices are continually being incorporated into what is now the CMMI Framework[10].

8 Two somewhat extreme examples of protecting one's own interests include (1) a contractor that under bids, hiding the real cost of the work to win the contract, then treating every change as an opportunity to reclaim some of the investment; or, conversely (2) a government program manager who strongly encourages "cutting corners" to meet aggressive cost targets, only to leave the resulting disaster-in-wait to fall on "the watch" of his or her replacement.

9 When taken to an extreme, ceremonial dependence on plans, processes, and standards can replace common sense, usurp accountability, and adherence to them can be used as an excuse for poor project performance and poor product functionality.

10 In August 2006, *CMMI for Development, Version 1.2* was released and in November 2007, *CMMI for Acquisition* was released. A third model covering service delivery is under development. Most of the comments in this report apply to the first of these, *CMMI for Development*, as there has been less exposure to the newer models in the CMMI family.

3 Factors that Affect Perception

Perceptions that position CMMI and Agile at odds with each other largely arise from factors present in the relatively more volatile commercial software development community. These factors may be generally attributed to market forces and human nature. We've described them as (1) misuse, (2) lack of accurate information, and (3) terminology difficulties.

3.1 MISUSE

Whether analyzing the CMM or CMMI, there is one thing shared by both works that makes them unique—they are *models,* not *standards,* for improving product quality and process performance.

However, for nearly two decades, the software industry has experienced the result of people misusing appraisal ratings as entry criteria, confusing appraisal ratings for measures of business performance, and misapplying a model as a standard in an environment in which products are created to meet contractual requirements.

Used in this way, CMM and CMMI best practices were misinterpreted and misused. It is not an exaggeration to say that any approach to improve an organization's achievement of business objectives in such an environment would have difficulty overcoming an emphasis on RFP requirements and keen competition for multi-year contracts.

Does this situation mean that CMMI is wrong for software? Not in the least. It simply illuminates the following reasons why some perceive CMMI to be incompatible with Agile ideals:

- The context from which the CMM and CMMI originated was specific to a particular customer base having unique challenges and characteristics of high risk and low trust.
- The CMM and CMMI were a new paradigm introduced into a large (and dominant) industry where paradigms, including the attitudes and beliefs associated with them, were in place for many years (e.g., command and control).
- Agile ideals developed as a backlash against the inefficient software development patterns that arose in this industry.

These points describe the context in which the CMM and CMMI were developed. This context enables us to understand some of the characteristics of the CMM and CMMI and how they have been used over the past two decades. While the language of CMMI, admittedly, may retain some of the flavor and phrasing of this context, each release of a CMMI model grows further away from these roots to embrace a richer and more dynamic set of contexts[11].

[11] Users of older model versions may not fully make the transition as new versions are released, and older beliefs and values may persist. Further, while CMMI and SCAMPI materials continue to evolve, it is simply not possible to bring all users rapidly forward to the newer versions, government edicts and SEI encouragement notwithstanding.

Further, this context is not the sole element determining how CMMI should be used, when and where it can be used, or what defines whether CMMI is being properly used. Nor for that matter, does this context determine its applicability in other contexts. The challenge is for the broader community to identify the practices and methods (or practice implementations) that enable organizational maturity in more dynamic contexts (e.g., internet commerce, social networking, and games development). An increasing subset of both CMMI and Agile method users are trying to do just that.

3.2 LACK OF ACCURATE INFORMATION

As we write this report, the availability of accurate information has improved. However, these improvements are new, thus, the pre-improved situation is still the current one for many. Internet searches for information (papers, presentations, trade articles, blog posts, discussion group posts, and wikis) on Agile and CMMI sorted by year yields telling results.

It's no surprise that in many of the writings of the original Agile Alliance (those who signed the famous *Agile Manifesto* and *Agile Principles*), much of the impetus for their collective philosophy and methods were a direct counter-point to what they (often justifiably) perceived as heavy-handed, wasteful, over-burdened and ceremonial processes. "Oil and water" is not an uncommon expression used to refer to the relationship between Agile methods and CMM/CMMI in the writings and postings of Agile supporters. If we account for the historical differences and challenges described above, this is no surprise.

What is interesting is that it is easier to find far more material on CMMI from the Agile perspective than on Agile methods from the CMMI perspective [Levine 2005, SEI 2008e]. This observation leads many to conclude that Agile methods were largely ignored by CMMI users, which is not far from the truth. Although two noteworthy articles were written from this perspective in 2001 (one by Hillel Glazer in *CrossTalk – The Journal of Defense Software Engineering*, and one by Mark Paulk [an author of the CMM], which was broadly distributed by the *IEEE Computer Society Dynabook*), with few exceptions, these perspectives were received with open skepticism from both camps, but otherwise largely ignored.

Discussion of Agile methods in CMM/CMMI settings was frequently peppered with anecdotes equating Agile with "no discipline." In fairness, the XP community (unintentionally) invited this censure. Even the name "extreme programming" conjures up an image of skateboard parks or the rule-breaking anarchy of off-piste snowboarding. Certain readings of the *Agile Manifesto* (excerpted in Section 5) might bear resemblance to this perception. With Agile proponents being perceived as openly "hostile" to CMM/CMMI, and CMM/CMMI being too often misapplied as described above, the few attempts to draw attention to the possibilities of a peaceful co-existence between CMMI and Agile were easily lost in the noise of other competing messages.

In searching conferences, little material on Agile and CMMI used together exists prior to 2005. In late 2005, the NDIA (National Defense Industrial Association) introduced an *Agile and Lean* track to their CMMI Technology Conference and User Group event, and in 2006, the SEI added a similar track to the SEPG^{SM} 2006 conference. In 2007, both events grew their respective Agile tracks, while similar tracks appeared at events of other process-oriented organizations. Since these events, material placing Agile and CMMI used together in a positive light began to grow, and at the most recent Agile conference (Agile 2007), a presentation was given on a successful introduction of Scrum in an organization that had achieved maturity level 5 [Sutherland 2007].

Agile and CMMI have been available simultaneously since 2001, and while participation at conference sessions has continued to grow in subject areas combining them, not before 2005 was there a major presentation on the successful merging of the two concepts[12] [Anderson 2005a]. It is also worth noting that the Agile and CMMI communities don't mix much. Few attendees of Agile conferences also attend CMMI conferences and vice versa. Even more surprising, rarely do the thought leaders in either community publish in the same sources.

Nonetheless, continuing research on the topic reveals that although momentum seems to be gaining on the side of the successful merging of Agile and CMMI, it has yet to overtake (let alone extinguish) the existing perception that the two ideas are incompatible.

It may be worth addressing the demographics in the software development industry. How much of the industry was Space and Defense in 1985? How much of it is Space and Defense today (measured by participation of the 15+ million software engineers worldwide)? Of those not in Space and Defense, how many are so young that they never used a third generation language, such as C, or worked on a true waterfall-style project? As a result of these demographics, a growing number of members in our profession demonize traditional methods (which for some, CMMI symbolizes) based on superstition or hearsay rather than experience.

We'll return to the topic of available information in section 6.3, *Current Published Research*.

3.3 TERMINOLOGY DIFFICULTIES

CMMI, Agile development, traditional development, and product development in general all have their own vocabularies. However, the overlap in these vocabularies includes terms that are given different or even niche meanings. More important is when terms introduced in one vocabulary have a history of use elsewhere and the term becomes associated with its previously-used context.

For example, *technical data package* (TDP) is a term used in CMMI that is intended to refer to the collection of product-oriented documentation associated with a product being developed that describes the product from a technical standpoint. It is typically used for further development, operation, installation, training, repair, support, troubleshooting, or maintenance of the product.

The TDP can include text, diagrams, drawings, designs, specifications, and/or data sets. (In a small Agile project, the software code itself may largely suffice.) However, the term TDP has a meaning in systems acquisition where it refers to a specific deliverable that includes specific documents. Not all projects create that specific set of documents, nor is it reasonable for them to do so.

Many people unfamiliar with the holistic purpose of CMMI can easily misapply their previously-learned systems acquisition definition of TDP and thus assume that CMMI requires, for example, a MIL-STD-498 version of a TDP. This assumption is not the case. In actuality, CMMI uses the term TDP far more generically—an interpretation more likely to match what most small or Agile projects actually do and, possibly, what most successful projects do whether or not they are under contract with a government agency.

[12] At the Agile conference in Denver in 2005, David Anderson recounted his experience developing Microsoft Solutions Framework for CMMI Process Improvement as an Agile method compatible with CMMI maturity level 3.

Also, the word *predictable* is a locus of the confusion and misunderstanding between the traditional and Agile development communities. Many in the Agile community believe that software projects cannot be predicted with any great precision. These believers say, “Perfect is the enemy of good enough, so don’t try to predict, just react and re-factor.” However, CMMI uses the word *predictable* in a more subtle sense[13]. Predictability does not arise from a detailed project plan up front that covers the entire project lifecycle (nor does CMMI require or generally expect this type of predictability to be possible).

The Agile community might find a lot of benefit and interesting results from experimenting with this higher-level concept of predictability. For example, David Anderson has achieved predictability in sustaining engineering at Corbis with a release every two weeks, but the content of the release is not bound until five days prior to release due to the unpredictable nature and dynamic character of the work scope [Anderson 2008]. Through better process management and improvement, he’s been able to meet CMMI rigor for predictability while simultaneously using an Agile approach that adapts to the unpredictability of the work and the market.

[13] In CMMI at maturity level 4, predictions are derived from an understanding of variation at the process-step level combined with probabilistic or statistical models that predict the ranges in quality and process performance that can be expected as project final or interim outcomes. For example, by investigating and characterizing the defect injection and removal behavior of individual design, coding, and verification subprocesses as a function of team skill and product complexity, and statistically modeling the relationship among these attributes and project final or interim outcomes, a project can periodically evaluate whether it is still on track to achieving its quality and process performance objectives. In a similar manner but by considering team velocity and the attributes affecting team velocity, an Agile project can be “predictable,” even though the scope, schedule, and budget of the project are changing and reacting to feedback and market conditions.

4 The Truth About CMMI

In CMMI, *process management* is the central theme. It represents learning and honesty as demonstrated through work according to a process. Process also enables transparency by communicating how work should be done. Such transparency is within the project, among projects, and being clear about expectations[14]. Also, measurement is part of process and product management and provides the information needed to make decisions that guide product development.

However, if CMMI is used in the pursuit of maturity level numbers, the resulting process improvement efforts can sometimes lose sight of customer value, product, project value, and practical business goals.

Also, when CMMI users establish (sets of) standard processes for use by the organization and its projects, they sometimes fail to ensure that these processes are (1) successfully deployed to all new projects; (2) periodically revised based on lessons learned from use; (3) compatible with individual and team practices and flexible enough for teams to adapt the processes to their needs according to their experience; and (4) written using language and formalisms that practitioners understand. The result can be rejection of the organization's process improvement efforts by practitioners.

There is a balance to achieve across organizational, project, and individual prerogatives and responsibilities that are a function of risk, task maturity, trust, and other business, fiduciary, and cultural considerations. If the balance is weighted too heavily in favor of the organization, projects and practitioners may lack the flexibility they need to be successful and motivation fails. On the other hand, too much flexibility can expose the organization to excessive risk (e.g., from mis-aligned teams) and missed opportunities for organizational learning that over the longer term might lead to improved product quality and productivity. It is difficult to achieve the right balance.

4.1 CMMI IS A MODEL, NOT A PROCESS STANDARD

A plausible rationale for many of the challenges faced by organizations using CMMI is likely the fact that most processes implemented using CMMI (as witnessed by the authors and garnered from feedback from others who use CMMI) fail to distinguish that CMMI is fundamentally a *model*. Instead of working with CMMI as a model, they work with CMMI as a *standard*. A *standard* is an auditable, testable, *compliable* work with a narrow field of *distinct*, acceptable, and demonstrable outputs with little variation from one implementation to the next.

This view of CMMI as a *standard* is a complete misuse of a model. To reiterate from the model itself, "CMMI contains neither processes nor procedures;" the lists of *typical work products*, for example, are *examples* of process outputs. They are not exhaustive lists of required process outputs nor are they mutually exclusive of other possible process outputs. The practices are not steps in an organization's set of standard processes, and they are not activities that necessarily occur neatly within a specific business process.

[14] In reality, CMMI and Agile share many of the same values.

They are process-oriented activities that when used collectively can attain the process area and business goals to improve the real-life business activities of an organization, wherever and whenever those activities may exist. CMMI's practices are also meant to encourage an organization to experiment with and utilize other models, frameworks, practices, and process improvement approaches, as appropriate, based on the organization's needs and priorities.

In some important ways, CMMI is an "ivory tower" of theoretical concepts born of decades of research and practical application. It is a collection of the activities to be expected of an organization as it sets out to improve its processes. CMMI is not and never was meant to be a replacement or a definition of anything in the real world. That is what a *model* is. It is an ideal from which we are meant to learn and relate to actual situations.

All too often, CMMI has been *applied* rather than *implemented*. The standards-centric *application* of CMMI has contributed to some spectacular failures and losses of time and money. The key difference between *applying* and *implementing* CMMI is that applying usually appears as a superimposition (or overlay) of model practices onto existing activities with an expectation of producing the example work products found in the model, rather than seeking the natural products of the organization's processes. This misplaced focus is often the by-product of an overly strong focus on achieving a particular appraisal rating.

In contrast, *implementing* CMMI is using the model in the same way that engineers and architects use models: as a learning tool, a communication tool, and a means of organizing thoughts. The more implementation-oriented an organization is, the more improvement-centric it is; thus, the focus is on maturing and growing process capabilities rather than ratings. The ratings are the by-product of the improvements in these implementations; and the model serves as a tool to holistically examine the organization's processes and performance and determine possible areas for improvement.

When viewed holistically, CMMI's ultimate goal (i.e., continuous process improvement) is to cause an organization to become less wasteful, leaner, and more in touch with actual development progress. Ultimately, both Agile and CMMI, especially in high-trust environments, expect organizations to see gains in productivity by eliminating unnecessary effort. It's true that implementing Agile methods will often eliminate many unproductive efforts and behaviors at the project level. However, even with Agile retrospectives, what CMMI offers beyond Agile is an infrastructure of organizational learning and improvement that benefits the projects even before they begin.

When using CMMI as a model for process improvement rather than a standard for it, organizations are free to implement whatever applications of process improvement suit their needs and they can see organization-wide progress beyond the project level and beyond immediate projects and developers. CMMI provides an approach to improvement in which processes and improvements to them are supported by the organization, persist beyond their original implementation date, and do not encumber development teams with waste (e.g., from re-inventing a process already known to work well in similar circumstances).

4.2 PROCESS AREAS, NOT PROCESSES

CMMI is comprised of *goals* and *practices* organized into *process areas*. To simplify this discussion, no distinction is made between the types of goals or practices—suffice to say that each goal is composed of several practices.

An example goal (SG 1) and practice (SP 1.2) from CMMI for Development, Version 1.2 reads as follows:

> *SG 1 Establish Baselines*
>
> *Baselines of identified work products are established.*
>
> *SP 1.2 Establish a Configuration Management System*
>
> *Establish and maintain a configuration management and change management system for controlling work products.*

Process areas are not processes themselves. The processes themselves could be operating wherever and whenever and in whatever sequence necessary to perform the work of the business. Satisfying (i.e., achieving) goals implies that those areas of activities where processes are happening are being improved to some degree.

At this point, you easily can see that misinterpreting CMMI's process areas as actual processes can cause an enormously misleading effort and thus waste and abuse. Waste is found when organizations apply practices not intended to be performed as written, and abuse is found when evidence is not accepted in an appraisal because it lacks of conformity with the process areas, which are considered defined processes.

Goals are the only required components of CMMI. The practices within the goals are *expected*, but not *required*. What this statement means is that to satisfy a goal, some form of activity must take place that generates a state, output, or persistent condition in which a goal is satisfied. In the absence of any such activity, the practices themselves suggest what might be a suitable approach to satisfying a goal. Again, these are not necessarily practices to be incorporated *as is* within a given organization's processes, but instead are *recommended starting points* for the purpose of improving an organization's satisfaction of the goals of a process area.

Performing these practices in the absence of a business value for doing so is as unlikely to improve an organization's processes as not performing the practices at all. What is lacking in rote practice performance is the understanding of the rationale or motivation for the practice and the potential value the practice has toward achieving a business objective important to the organization.

Although CMMI allows the satisfaction of *alternative practices* that meet the goal but do not explicitly implement the practices, this concept is poorly understood by users. Further, there is a fear among users that the use of alternative practices will be viewed suspiciously in an appraisal.

Within the practices there is a tremendous amount of *informative material*. Including this informative material is the only way to describe a *model*[15]. Narrative is necessary to understand, interpret, and implement the model's practices and goals. On the plus side, informative material helps to explain the purpose of the practice and provide considerations relevant to how it might be implemented.

However, for people and organizations that do not have the skills or experience to understand, interpret, and implement the practices and goals to their specific contexts, the informative material takes on the mantle of *requirements*. This situation is also true for appraisal teams and lead appraisers. Thus, the over-emphasis on specific model examples, typical work products, and subpractices often results in prescriptive processes with little gain in quality or organizational performance.

This concept-based disconnect is compounded by an approach toward CMMI use that can be aptly described as *pathological box-checking* and an over-zealous contracting or "small-picture" auditing attitude that seems willing to sacrifice what is good for people, projects, customers, products, and the long-term health of each in favor of supporting the generation of appraisal evidence.

4.3 SCAMPI APPRAISALS

With the proper understanding that CMMI is a *model*, the SCAMPI^{SM} (Standard CMMI Appraisal Method for Process Improvement) appraisal method is designed to determine whether *process improvement* (i.e., process maturation) is occurring as a reflection of model (e.g., CMMI model) practices having been *implemented*. Even the term used for CMMI artifacts points in the direction of practice *implementation* rather than practice *application*.

SCAMPI appraisals are not audits or tests. Instead, a SCAMPI appraisal is a *process story* in which the elements of the story include the following:

- the CMMI model
- the organization's development practices
- acculturation of those development practices
- the improvement activities within the organization's broader set of practices
- the ability of project participants to articulate their development practices

[15] Another way to express the need for informative material is that the value in a CMMI practice does not lie only or even principally in the statement itself. Rather, the statement is a summary of a larger concept comprised of activities or operations that will achieve a particular process area goal. To quote from the SEI website (i.e., http://www.sei.cmu.edu/cmmi/adoption/cmmi-informative.html), "[I]nformative material supports correct interpretation and implementation of CMMI practices, but is neither required nor expected, nor is it to be used as part of a checklist in an appraisal." The informative material "takes on a critical role in explaining …the intent of the practice, [and]… the practice's dependencies with other parts of the model." If the intent is met in some other way or if the goal can be achieved through activities different from those specified in the CMMI practice statements, then the CMMI model material associated with that practice or goal need not be further pursued from a maturity rating standpoint. (Of course, it is possible that further process improvement is possible through further consideration of informative material, but such pursuance must be weighed against higher priorities and better opportunities for improvement to the business.)

It is the job of an appraisal team, led by a lead appraiser, to discern the improvement activities of the organization and to determine whether the practice implementation indicators[16] objectively indicate whether CMMI was implemented. It is how the above *process story* elements *hold together* that determine the outcome (often a rating) of the SCAMPI appraisal.

If an organization has been engaged in an effort to improve its processes, Agile or otherwise, the SCAMPI method is used to ascertain which process area goals are being met. It then becomes incumbent on the appraisal team to understand the context of the organization's practices and the relevance of their activities toward process improvement. In many cases, these activities—based on the organization's context—may not appear in a form or format familiar to those whose experiences are limited to non-Agile development approaches. Nonetheless, it is the appraisal team's responsibility to interpret CMMI model practices and the organization's artifacts of process improvement so that the appraisal team can correctly determine the overall process maturity and/or capability profile of the organization being appraised.

Agile methods are not addressed specifically in either the model or the SCAMPI method because Agile is a particular approach to development activities that contrasts with other approaches to development. CMMI, not being a development approach, and SCAMPI, being agnostic on the subject of development approaches, are not irrelevant or of no value because an organization chooses to use Agile principles. An organization's development approach is its choice.[17]

---

[16] The fundamental idea of practice implementation indicators (PIIs) is that the conduct of an activity or the implementation of a practice results in "footprints" (i.e., evidence that provides a basis for verification of the activity or practice implementation).

[17] Likewise, the SEI has been asked over the years why CMMI does not promote specific methods, tools, or techniques such as function points, RUP, or Six Sigma (or you name it). Again, by being a *model* that is *implemented* as opposed to a *standard process* that is *applied*, CMMI, by its nature, must remain agnostic to such selections, leaving the organization free to select the methods, techniques, and tools, most appropriate for its use.

5 The Truth About Agile

The *Agile Manifesto* reads as follows:

> *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value [the following]:*
>
> - *individuals and interactions over processes and tools*
> - *working software over comprehensive documentation*
> - *customer collaboration over contract negotiation*
> - *responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more."*

The *Agile Manifesto* is frequently read in such a way that ignores the last line (by too many Agile proponents and detractors alike) and where the "things on the right" (items commonly found in too many plan-driven, contractually-driven, standards, and audit-driven environments) are not merely valued less, they effectively have *zero* value.

The *Agile Manifesto* has been frequently cited by Agile proponents as justification for not having processes, for not documenting their work, and for not having plans. This interpretation gives justifiable cause to Agile detractors to accuse proponents of being "undisciplined lackeys." Similarly, Agile detractors abuse the *Agile Manifesto* by using the same exact mechanism. By attributing a value of zero to the "things on the right," detractors assume the worst of Agile proponents.

However, these interpretations of the *Agile Manifesto* are incomplete and disingenuous.

At a genuine level of discourse, who wouldn't value the things on the left more than the things on the right? Keep in mind the paradigm from which the *Agile Manifesto* is trying to depart. Even without researching the discussion among the signers, the source of the *Agile Manifesto* should not be difficult to guess. In too many implementations there were plans, processes, and standards that were overly detailed, rigid, and just plain abused to the detriment of people, projects, products, customers, and technology.

Common practices associated with Agile approaches include the following:

- Iterative and short incremental time-boxed development is used.
- There is frequent and ongoing customer feedback. Customers are commonly embedded with developers. In fact, extensive use of tacit knowledge is a key in Agile development and practices. Use of tacit knowledge is of most benefit in a high-trust environment that can directly benefit from the low transaction costs of software engineering activities. The risks from the use of tacit knowledge, though often not explicitly managed, are mitigated by the following nature of the domain:
  - the length of the project (and the expected lifecycle of the product)[18]
  - the self-documenting nature of modern code

[18] Agile teams and projects often assume low staff turnover (rightly or wrongly) because of the short nature of projects and the small size of teams.

  - the use of tools that can reverse engineer designs[19] and other artifacts as well as continuously integrating code
  - the small size of the team.
- Each increment potentially delivers value (e.g. working code). Value can also be *what not to do*.
- The delivered product is fully tested for *each* delivery. Products are often created by writing rigorous tests first, then evolving the product to pass the tests.
- Everyone owns quality, including the customer (or customer representative). Development teams are often multi-disciplined, in which all developers are generalists and collectively "own" the product. Customers are equally accountable to ensure the product meets their expectations. Products are prevented from advancing until a common understanding of the product's functionality, inclusions, and exclusions are established. There is a *fail early* attitude with an underlying assumption that the cost of failure is low to negligible.
- Detailed requirements are developed *just in time* and evolve along with the understood product. Effort is not invested (i.e., wasted) by speculating on requirements or expectations. As the product to be delivered evolves, the requirements become more and more detailed[20].
- Change is expected, embraced, welcomed, and/or accepted at any time. Acknowledged sources of changes in traditional (i.e., serial) development are changes to requirements that were baselined too early (from an Agile or Lean perspective). Furthermore, the expectation of changes drives design decisions. By design, the product and architectural components are not locked in so that developers can perform continuous trade-offs until all the component requirements are available[21].
- Self-directed, empowered teams are used. Task-mature, well-trained, cohesive, high-trust, small teams who take ownership of the product need little direction, management, or task-feeding. Broad product, project, and business goals are shared with the team so that informed managerial, organizational, and technical decisions can be made by the team to determine what must be done, by when, by whom, and to what standards. Teams are allowed to respond to current conditions rather than adhering to over-emphasized, over-detailed plans. Thus, plans and planning are actually updated frequently and communication with superiors (whether organizational or team) often takes place daily or nearly daily. Highly ceremonial

19 Again, there is a bias against documentation. People assume that documentation is complete and up to date, which is often not the case. Without sustained effort over time to keep it up-to-date, it can become misinformation. Documentation is often a poor substitute for communication, especially for a co-located team.

20 There is a recent movement toward articulating this *just-in-time requirements* method through the use of Real Options Theory. The key proponents are Chris Matts (from London), Olav Maassen (from Amsterdam), and Kent McDonald (of the Agile Project Leadership Network, from Iowa).

21 Traditional assembly line production of code assumes a cost of change curve that accelerates with late discovery. Hence, early discovery through analysis methods and inspections is desirable [Boehm 1976]. Some Agile proponents (e.g., for eXtreme Programming) have argued that the cost of change rises quickly and asymptotically levels out [Beck 2000]. Actually, they are both often wrong assumptions—cost of change is contextual and depends on where the capacity-constrained resources lie in the process flow [Anderson 2003]. Cockburn showed that slack resources in development can be used to fix defects at no cost to productivity (i.e., throughput) and hence, in some situations the cost of change in coding is as Beck suggested negligible [Cockburn 2006].

status meetings with management or reviews with customers are replaced with frequent interactions with these stakeholders.

- The process is periodically evaluated and adjusted. Task-mature teams take ownership of the process they employ and adjust it based on feedback on what works and what does not.

In general, attributes of successful Agile implementations include the following[22]:

- small teams consisting of approximately ten people
- an involved customer
- "rolling wave" or continual planning
- co-located and cross-functional teams
- organizations that are not in the habit of breaking up teams until each member is individually competent at Agile

Agile is not successful when the following is true:

- lack of processes
- lack of discipline
- absent a role for plans or planning

The *Agile Manifesto* has a list of *general principles* attached to it.

> *"We follow these principles:*
>
> - *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
> - *We welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
> - *We deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
> - *Business people and developers must work together daily throughout the project.*
> - *We build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.*
> - *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
> - *Working software is the primary measure of progress.*
> - *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
> - *Continuous attention to technical excellence and good design enhances agility.*
> - *Simplicity—the art of maximizing the amount of work not done—is essential.*
> - *The best architectures, requirements, and designs emerge from self-organizing teams.*

[22] Some organizations, which are adopting both CMMI and Agile methods, characterize the situations that are more suitable for the implementation of Agile methods according to the level of trust, customer/end user availability, project scope, scale, expected lifespan of the product, cost of delay, value of early (partial) delivery, and cost of failure present in the situation.

- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."*

Each word in the *Agile Manifesto* and the *guiding principles* was attained via consensus—a process familiar to anyone having been on a SCAMPI appraisal. Each word carries meaning. Note the use of *priority*, *most*, *primary*, *sustainable*, *excellence*, and *continuous*. These are conditional and qualifying words, not absolutes. Anyone who interprets these words as absolutes are misusing Agile ideas just as they would if they were to interpret of CMMI's *typical work products* as absolutes.

6 There Is Value in Both Paradigms

CMMI and Agile are compatible. At the project level, CMMI focuses at a high level of abstraction on *what* projects do, not on what development methodology is used, while Agile methods focus on *how* projects develop products. Therefore, CMMI and Agile methods can co-exist[23].

There can be much value gained from Agile and CMMI synergies. Today, many CMMI-adopting organizations have Agile development teams. Conversely, CMMI can be effectively introduced in an Agile setting where an iterative, time-boxed approach is used, which is perfectly compatible with CMMI.

CMMI and Agile can complement each other by creating synergies that benefit the organization using them. Agile methods provide software development how-to's that are missing from CMMI best practices that work well—especially with small, co-located project teams. CMMI provides the systems engineering practices that help enable an Agile approach on large projects. CMMI also provides the process management and support practices that help deploy, sustain, and continuously improve the deployment of an Agile approach in any organization.

6.1 CHALLENGES WHEN USING AGILE

The major challenge when using an Agile approach[24] in a large project is keeping the small teams aligned and coordinated for the duration of the project to ensure its success while adhering to Agile's team-focused principles and values. Maintaining alignment and coordination across a distributed project requires that someone (possibly a team) or a mechanism maintain coherence (i.e., unity, logic, and consistency) of the following:

- overall system capabilities to be developed, including non-technical requirements
- scope, quality, schedule, cost, and risk tradeoffs
- product (or service) architecture

If coherence is not achieved, alignment becomes divergent on multiple fronts. For example, one common Agile practice, re-factoring, often doesn't scale up well when multiple development teams must cooperate on the same product. Through a combination of activities such as release planning, test planning, continuous integration, and employing feature-oriented rather than component-oriented teams, Agile approaches have been scaled to work in larger projects.

From a systems engineering perspective, these activities are challenging for large, complex projects to perform well:

- establishing overall product objectives and a project vision
- managing the allocation of requirements to teams

[23] The references at the end of this report include publications and presentations that also characterize CMMI and Agile as complementary and synergistic [Anderson 2005, Boehm 2003, Canditt 2008, Santos 2007, and Sutherland 2007].

[24] Of course, this is a challenge common to all large, complex projects.

- defining and maintaining interfaces and constraints among teams (for both the product and the process)
- maintaining effective product integration, verification, and validation strategies for the overall product (or service)
- coordinating risk management across the project

These alignment and coordination activities, necessary in larger, complex projects, are described in the systems engineering practices found in the Engineering, Risk Management, and Integrated Project Management process areas of CMMI. Thus, CMMI provides a “safety net” for large projects that helps reduce the risk of something going very wrong.

Some Agile proponents acknowledge that many Agile methods have yet to be proven effective or possible as the core product development process on large projects[25]. However, there has also been increasing success scaling Agile approaches for use on large projects by introducing a top layer of coordination (e.g., a Scrum of Scrums) and by providing explicit and early attention to non-functional requirements and product architecture[26]. Alternatively, Agile methods are also in use on features or components of larger systems in which the Agile team is, in effect, isolated from the rest of the product development team. The software development community is increasingly learning how current Agile methods can be effectively extended to work on large-scale, complex projects.

Agile methods generally lack practices and guidance for implementing and supporting an Agile approach across the organization. While some larger companies (e.g., CapitalOne, BMC Software, and Yahoo!) are pursuing large-scale enterprise adoption of Agile, the level of documentation and implementation guidance available to support such adoption is relatively modest[27]. Agile implementations will not “stick” or improve without an organizational context that supports process definition, measurement, feedback, training, and improvement—mechanisms that are described in CMMI. CMMI practices also address institutionalizing and improving deployed processes, be they Agile or not; therefore, CMMI supports a long-term view of process adaptation and improvement across the enterprise as a whole.

Of course, many organizations have identified and implemented project management, software development, and process improvement practices without the benefit of CMMI. The book *Microsoft Secrets* discusses Microsoft’s seven-year journey to improve its product quality [Cusumano 1995].

---

25 A lot depends on the definition of *large* and which Agile method is being used. The first FDD project (1997-1999) was characterized by 50 people, 18 months, and 1.5 million lines of Java. Two years later, Jeff DeLuca led a 250 person project in New Zealand. Hence, there is relatively early literature and evidence that Agile techniques can scale up [Highsmith 2002].

26 A continuing challenge experienced in many Agile implementations is ensuring adequate and early attention to non-functional requirements and product architecture [Cao 2008]. Practices that address this challenge are found in the Engineering process areas of CMMI. Relative to its own Agile approach, the SEI is currently collaborating with NAVAIR on extending the Team Software Process^{SM} (TSP^{SM}) to address systems engineering and is also experiencing promising results from doing so [Carleton 2006, SEI 2008d].

27 A notable exception is [Schwaber 2006], which provides guidance and case studies on how to introduce SCRUM to an enterprise; but may have applicability to other Agile approaches and to large projects as well.

Arguably, in some of its divisions Microsoft put in place many of the practices required to achieve maturity level 3 in a manner reflecting the challenges of their environment. Therefore, given enough time, organizations can address what is missing from their business and implement a solution without the benefit of a model such as CMMI. However, when used with care, the appropriate model can certainly make the journey easier.

Impediments to introducing an Agile approach also include factors common to the introduction of any new technology: lack of management support and general resistance to change. CMMI and the process improvement experience that has grown around it can address these issues. For example, with the introduction of an Agile method, management often fears a loss of control. Blending a top-down approach (i.e., CMMI) with a bottom-up approach (i.e., Agile) may go a long way in helping to reassure management that its concerns are being addressed in the improvement effort.

6.2 CHALLENGES WHEN USING CMMI

There are challenges to successfully using CMMI best practices as well. No development approach or methodology can effectively address all difficult challenges or situations [Elm 2007]. Just because an organization has been appraised at a particular CMMI maturity level is no guarantee that a particular project in the organization will succeed. However, organizations using CMMI can fail (and some are failing!) because they misuse the model or pursue process improvement and subsequent appraisal with misguided motivation or with imprudent leadership.

Sometimes, in the rush to achieve a maturity level, the focus on improving organizational performance is lost. The wrong standard process and tailoring guidelines may be imposed. For example, the standard process may be over-specified and thereby over-constrain projects to the detriment of project success.

The tailoring guidelines may not allow projects the flexibility they need to tailor standard processes to address their project-specific needs and priorities [Charette 2004]. In particular, these guidelines may not allow the process dexterity[28] required to effectively address difficult project challenges and risks. To find ways to more effectively address risk (and opportunity) was one of the main motivations for the development of iterative and spiral approaches to product development—and more recently—of Agile methods. In today's increasingly dynamic world, CMMI-based organizational process improvement approaches cannot rely exclusively on traditional project management approaches, waterfall-based project lifecycles, and heavyweight analysis methods.

Conversely, standard processes may be under-specified and omit proven organizational and project practices. Similarly, the definition of standard processes may tend to treat all CMMI practices equally (to earn a maturity level) and fail to recognize that some specific practices are critical to the business. Proper attention to these practices may add immediate value, but these practices can be lost in the noise of charging toward a maturity level rating.

CMMI and Agile each brings something to the table on how to run the business that the other side should listen to and learn from. More dialogue would be healthy and beneficial for users, the paradigms, and the broader community.

[28] We use the word "dexterity" in this sentence because we are trying to use a term other than "agility."

6.3 CURRENT PUBLISHED RESEARCH

There is a large and growing body of research on the adoption of CMMI [Elm 2007]. As was the case in the early days of CMMI (e.g., [Goldenson 2003]), the nature of the published research on the efficacy of Agile methods is largely anecdotal. Also similar to CMMI, there is no one way to use Agile methods. As already mentioned, CMMI addresses the *what*. Therefore, the studies of the costs and benefits experienced by organizations using CMMI actually report on how those organizations implemented a set of processes to meet the requirements and expectations set forth in selected CMMI process areas and not on the effectiveness of individual CMMI practices per se.

Likewise, there are multiple ways to implement Agile methods. Because the methods offer somewhat different how-to's, some methods clearly work better in some situations than others. Well-intentioned implementations of Agile methods can fail to deliver a good product and uphold the people-centric values held dear by Agile and CMMI champions alike. In a case study written by Khaled El Emam, the employee morale at one organization actually became worse following the introduction of Agile methods until more explicit attention was given to addressing selected CMMI process areas [Chrissis 2007].

Many are too quick to pass judgment, declare victory, or declare failure. They expect instant results and often assume that those results will persist ad infinitum. The first project using Agile or CMMI has to be a success or the newly tried approach is labeled a failure; equally, one success is assumed to represent an organization-level, institutionalized pattern. The reality is that neither is generally the case. There are few case studies in which the organization still exhibits the same success a few years later after the environment, projects, personnel, and management have changed.

While formal studies of impact and benefits have been clearly solicited from and by the Agile community, there has been less energy behind *systematic* attempts to examine impacts (both successes and failures) that result from introducing Agile methods. It will take years to understand the efficacy of individual methods and the contexts in which there is high return on investment. The same understanding has taken years for CMMI and that journey continues.

The problem with some of the existing research for both CMMI and Agile methods (or of new technologies in general) is that success stories are more likely to be reported at conferences than are failures and that research or reports based on such samples of convenience are therefore likely to be favorably biased.

Furthermore, it is easier to claim success than to attribute success to specific, discrete practices separate from other environmental factors such as team composition, organizational culture, and customer interfaces[29]. Much like the adage, "It is impossible to prove a negative," project failure can happen for many more reasons than those controllable by a project. Conversely, a project can claim that it did nothing more than change some practices or introduce some methods and thereby justify its claims that what it changed or introduced is the source of its success. In either case, the proof is not robust; therefore, most such research should be considered anecdotal.

[29] Elm 2007 may be a notable exception. By design this recent research study targeted a broader range of programs (including less successful programs) and tried to account for challenges in the program environment. An important finding of this study was that while higher process capability (or maturity) results in markedly better project performance, *project challenge* (i.e., technical, stakeholder and contract complexity; requirements, scope, and funding volatility) lessens the benefits of higher capability.

6.4 THE HOPE IN RECENT TRENDS

The public relations problem that CMMI (or any process or development model) faces is that each organization that uses CMMI affects many project participants (e.g., product developers) who are able to quickly spread the word of their painful experiences through existing developer communication channels, irrespective of any benefit attained, whether immediate or eventual[30].

Nonetheless, recent experience reports indicate that CMMI and Agile methods can work well together and result in business value [Sutherland 2007]. These reports provide one basis for the synergies discussed in this report.

Jeff Dalton (a CMMI appraiser and Agile leader [Dalton 2008]) remarked, "I've observed Agile organizations who fiercely apply little process nor produce valuable documentation in the name of being "Agile." Their rationale for the absence of process is found in the *Agile Manifesto*, which places a higher priority on *working code* than on project documentation. This prioritization, which is intended to shift the balance away from docu-centric traditional software development, is often interpreted as "no process or documentation" and can result in confusion and failure as systems become larger and more complex. When CMMI is applied appropriately and integrated successfully, it can provide the infrastructure required to successfully scale the management and delivery of complex systems while maintaining the benefits of iterative and incremental development that Agile provides. Also, it's stunning how few "Agile organizations" use Agile methods to design and deploy their set of processes" [Dalton 2008].

David Anderson (author of [Anderson 2004] and co-author of the APLN's *Declaration of Interdependence*) observes, "It's a fair comment that many in the Agile community tend to lump traditional and CMMI in the same category and use the terms synonymously. I was probably guilty of this back in 2002. However, I've come to the opinion that traditional really means waterfall or V-model software engineering with traditional project management (i.e., estimating, and tracking tasks in a dependency graph). In my view, CMMI transcends the definition or difference between Agile and traditional. But this view is not widely held or understood at this time."

An interesting recent development is the work of Jeff Sutherland (another Scrum founder) and others who at Agile 2007 presented how Scrum could build on an organization's successful achievement of maturity level 5, and how CMMI and Scrum together may be the "magic potion" of software. High maturity organizations were able to manage change better and institutionalize the change. Hence, a change to Scrum worked better in a high maturity organization. The transition was faster and institutionalized better.

It would be fair, then, to assert that the perceived problems between Agile and CMMI stem not from Agile and CMMI being inherently inconsistent, but rather from a combination of misperceptions and the negative experiences with some organizations that imposed overly-heavy and restrictive standard processes on all projects as part of using CMMI—approaches that may be effective in larger, low-trust, and high-assurance contexts—but burdensome and paralyzing when used in smaller project situations. CMMI encourages organizations to use processes that best meet their business objectives. However, the business objectives of organizations vary widely.

30 Gary Wolf of Raytheon recounted, "When we first heard our organization had to get to maturity level 3, we were filled with negative feelings. Change is painful unless you are stuck in a fire. But if we had to improve our processes, we were resolved to do it the right way. After several months, somewhat to our surprise, life got better in the projects: there was less rework, fewer late nights, and more time for the right kind of work."

What is needed when using CMMI are standard processes (or sets of standard processes) that recognize the broader spectrum of situations that may arise and that provide process solutions that collectively address all relevant project situations[31].

Were DeMarco and Lister in their 1999 book *Peopleware: Productive Projects and Teams* [DeMarco 1999] correct in asserting that high maturity organizations (in the era of CMM use) become conservative and risk averse? It requires a broadminded attitude in regards to the software engineering process to enable Agile method adoption. In general, Agile values align with a culture in which anyone can suggest a change (i.e., improvement) and there is less attachment to the status quo or established way of doing things. For example, some organizations place tremendous emphasis on level ratings and are loath to change anything perceived to challenge appraised processes upon which their ratings were based.

Barry Boehm and Rich Turner describe the need for hybrid approaches that combine and balance strengths from both plan-driven and Agile approaches based on a risk assessment of a particular project's characteristics (e.g., high risk of failure vs. an overwhelming need for speed to market) [Boehm 2002, Boehm 2003].

In his foreword to Boehm 2003, Alistair Cockburn (founder of the Crystal family of methodologies [Cockburn 2004]) provides this summary of the book's message, "If one has strong discipline without agility, the result is bureaucracy and stagnation. Agility without discipline is the unencumbered enthusiasm of a startup company before it has to turn a profit[32]."

We largely agree with these messages, but pragmatically it is difficult to know how much planning and architecting (and related development activities) are needed to minimize project risk exposure; the answer would seem to be situation dependent. Add subsequent product evolution (and attendant product releases and associated stakeholder costs) and it would seem some significant discipline (in the sense of self-control) and experimentation is needed to properly assess which hybrid approaches work in which contexts and to generalize to software-related industries as a whole.

We hope this report encourages both CMMI and Agile proponents (ideally, everyone in software-related industries) to do the following:

1. Recognize the value of each paradigm.
2. Resist common misperceptions.
3. Continue experimenting, learning, and reporting on what works and in which contexts.

[31] There is also a need for more explicit coverage of Agile approaches within CMMI itself and in the guidance provided within CMMI training and for appraisal purposes. In particular, SCAMPI Lead Appraisers need to have a more versatile understanding and appreciation for Agile approaches, what they are, and how they can benefit the organization. This report can be considered a step in that direction.

[32] Unfortunately, *Balancing Agility and Discipline: A Guide for the Perplexed* [Boehm 2003] is a misleading choice for a book title. The choice *is not* Agile or discipline because how can you continuously produce tested, working software in short time intervals without discipline? Perhaps a more accurate title might be *Balancing Agile and Plan-Driven Approaches*. CMMI can be used to address the full spectrum, with the risk assessment of a project's characteristics described in [Boehm 2003] being the basis for creating the project's defined process through appropriate process tailoring.

The tendency to form *camps*, such as separate CMMI and Agile camps, is an inherent human characteristic, even when it comes to software development. What is needed is a *supertribe* toward which all camps can feel an affinity [Immelman 2003]. This *supertribe* can facilitate continued experimentation, learning, and results reporting. Creating a supertribe that encompasses CMMI and Agile is the shared goal we have for this report—and we encourage you to join us in pursuing this goal.

7 Problems Not Solved by CMMI nor Agile

Part One of each CMMI model describes three aspects of development projects as (1) processes, (2) technology, and (3) people. CMMI makes no secret that it focuses on processes. Meanwhile, Agile methods clearly focus on people and allow people to determine technology and processes.

Regardless of the focus of CMMI or Agile, neither can entirely prevent simple human error, departure of key personnel, impact of incompetent personnel, active or passive insubordination, or deliberate sabotage. Neither can prevent the effect of an employee's personal life on his or her job performance. Each body of knowledge can have mechanisms to catch and manage such matters[33], but neither can prevent them any more than either can prevent the local economy from creating a scarcity of qualified employees.

Most organizations have experienced ill-fitting employees and the corresponding challenges in placing them or keeping them employed or happy. Most organizations have had to deal with employee losses, loss of project funding, evaporation of technology resources or suppliers, supply and demand in the marketplace, premature obsolescence of core architectural components, and other issues beyond their immediate control.

While derived and adapted abilities can positively influence the non-development aspects of product and service development, there are many aspects to development that neither CMMI nor Agile, as a first tier implementation, are meant to affect. The conclusion here is that neither CMMI nor Agile are panaceas to all that may hinder development.

To address the issue of how to most effectively recognize and credit Agile approaches within a SCAMPI appraisal bears further research. Perhaps simply a special guidebook for appraising Agile organizational units or projects or a set of Agile alternative practices associated with CMMI goals accompanied by an overview of Agile values and practices would prove useful.

Fundamentally, misuse is a problem for both CMMI and Agile. One of the original signatories of the *Agile Manifesto* recently wrote that he was hired to perform *due diligence* by a venture capital firm that had decided to invest only in companies practicing Agile methods. He found that of the companies he visited and examined, only three percent of those claiming to use Agile methods were actually doing so. (Whether another signatory would judge a similar percent is of course a different, though related issue.)

[33] Such workforce-management problems may be better addressed by the People CMM [Curtis 2001], which contains practices for improving the selection, retention, training, motivation, alignment, and skills of an organization's personnel.

One of many forms of misuse includes aspects of practice introduction entirely unrelated to either Agile or CMMI. Introducing new practices poorly will likely result in failed implementations regardless of what is being introduced. During a panel session on Agile at SEPG 2006, an audience member asked the panel about an Agile implementation failure at his company. The context of the question was that the adoption of Agile was an executive edict with unrealistic expectations of return on investment, disconnected goals, absent the Agile principles, and devoid of the cultural necessities of long-term change success[34]. This situation was an organizational behavior matter that neither CMMI nor Agile could immediately resolve.

[34] For articles about these kinds of situations, see www.Agilemanagement.net/Articles/Weblog/ThoughtsonEnterpriseAgile.html

8 Conclusion

Agile methods provide software development how-to's, purposely absent from CMMI, which work well on small co-located projects. Meanwhile, CMMI provides the systems engineering practices often required on large, high-risk projects. CMMI also provides the process management and support practices (and principles) that help deploy and continuously improve the deployment of Agile methods in an organization regardless of organization or project size.

A scaling limitation of several Agile methods has yet to be overcome with any consistency while still adhering to Agile principles. Adhering to Agile principles is inherently challenging given the context of large, long-term projects with geographically and organizationally dispersed project teams (a situation whose prevalence may be increasing). The mode of communication on large projects—in terms of spanning distance, time, and audiences—is necessarily a slower more involved endeavor than is possible with face-to-face, real-time, kinetic presence among project team members.

Scrum provides an Agile "wrapper" for many project entities that allows for "Scrums of Scrums." However, this approach still requires a strongly cohesive project team and can even wrap around non-Agile projects, thus begging the question as to the true agility of any project work being performed behind a "curtain" of an Agile-recognized approach.[35]

Although the complexities of large-scale projects require a more involved infrastructure, this fact is not a license to create unnecessary bureaucracies and unbalanced production at the expense of productivity. This creation of unnecessary bureaucracy occurs especially, but is not limited to, when CMMI is misused and a level rating is the principal expected outcome of a CMMI-based process improvement effort. As a result, over-engineering within process improvement activities is a common issue.

Using process experts to create and deploy process improvement activities is not unusual, and often has the advantage of allowing these experts (and the project teams) to focus on their respective tasks. But what makes this approach risky is that project personnel are frequently left out of process design activities[36] and are disinclined or openly skeptical toward the adoption of process improvement activities. This situation is typical of some Six Sigma style approaches to process improvement as well. Quality process teams are created and given the responsibility for quality. As a result, developers abdicate their responsibility for improvement (or worse, for quality).

[35] Some Agilistas question the inclusion of Scrum as an Agile development method as opposed to, perhaps, an Agile project management method. However, we have included it here because of its widespread recognition.

[36] Whether due to the design of process improvement activities or due to project personnel being too busy to provide meaningful inputs, each has the same negative effect.

Using Agile principles when designing and selecting CMMI practices can create more acceptable and appropriate process definition activities. Furthermore, the incorporation of CMMI goals into the project activities of Agile teams can help make these teams more mature and/or capable at handling the continuity of projects. Implementing CMMI practices in an organization that uses Agile methods, while remaining faithful to Agile principles, should also yield well-conceived Agile practices that maintain a lean, high-trust set of activities sufficient for the development of the product.

The following scenario suggests a hypothetical approach to using Agile ideas in CMMI-based process improvement.

Someone must be identified as the *product owner* for any product produced by a development team. Thereafter, no artifact of process improvement should be generated without an explicit owner for that artifact. The appraiser is off-limits as a product owner (i.e. for every CMMI-inspired artifact, there must be a real owner on a project). Every task or practice on a project produces an output. All process improvement-identified tasks and practices are an assumed part of the project and must be addressed as project activities. Either the output of a task serves process improvement purposes as is (i.e., an existing project activity already meets the need) or the output is specifically for process improvement—in which case someone must take ownership of the output. Depending on the project's budget structure, someone may need to "pony-up" money to create the work-product. An output with no owner means there is no value in the output. Why should any project spend energy on outputs that have no value? CMMI does not require it either. Therefore, effort would need to be spent re-engineering the process until someone "cares" about the output. Until then, the output is waste.

If those of us in both the Agile and CMMI camps would understand and accept our differences and explore the advantages of the other, we will find new ways of combining the ideas of both to bring improvement to a whole new level. Our challenge to CMMI and Agile proponents alike is to learn the value of the principles and technology of the other and begin to use them in their work.

9 Epilogue: A Call to Action

As authors and contributors to this report, we represent both sides of the perceived dichotomy and have deep experience in both camps, yet we universally agree that Agile methods and CMMI cannot only co-exist, but successfully integrate to bring substantial benefits to both Agile and traditional software development organizations. In these closing paragraphs, we issue a “Call to Action” to experts of each approach.

9.1 A CALL TO ACTION FOR CMMI EXPERTS

The SEI has always maintained that CMMI is *methodology neutral* and that, as a process model (and not a process in and of itself), CMMI can be applied to virtually any situation that involves the design, development, and delivery of systems and software.

Information about CMMI has always been freely available to all those who have an interest, although the methodology, manifested in training classes, white papers, and technical reports, has not always been effectively presented with the appropriate messages to each constituent group. This presentation of information has sometimes created confusion and misperceptions within the different methodology camps.

This confusion is caused, in part, by the biases of those who deliver the message, and those who receive it, as well as the methods used to present the information. In short, the message is just not being communicated or received clearly and all participants in the process share the burden (and consequences) of the inevitable result.

The SEI cannot directly influence the biases of the receivers, but it can prepare the presenters and monitor the reactions to messages by various constituent communities. The presenters that can be influenced by the SEI include CMMI Instructors, SEI Partners, and SCAMPI Lead Appraisers^{SM}. These presenters all can play a proactive role in the delivery of the message and the monitoring of reactions by the current and prospective CMMI user community.

CMMI Instructors

Members of the CMMI instructor community, particularly SEI staff and authorized CMMI instructors, are often the “first line of defense” during an attendee’s sojourn into the world of CMMI. Authorized CMMI instructors are well trained in the CMMI model itself, but it is impossible for them not to bring their own biases and experiences (for good and ill) into the selection and presentation of examples, ancillary discussions, and commentary. This additional material, at least based on feedback received from attendees, is often found to be as valuable or even more valuable as the authorized material provided by the SEI.

CMMI instructors are encouraged[37] to further engage the Agile community in their training by including examples from multiple types of organizations, including those that deploy systems using Agile, waterfall, and other relevant methodologies, in (1) the eight existing *Introduction to*

[37] A change request has been submitted asking the CMMI Product Team to include coverage of Agile principles and approaches, as appropriate, in CMMI models and in CMMI and SCAMPI training; and in its CMMI qualification, certification, and Partner program selection processes. (CRs are submitted as explained at http://www.sei.cmu.edu/cmmi/models/change-requests.html.)

CMMI course exercises and (2) the course content itself. (This material is particularly relevant in the course modules related to Project Planning, Project Monitoring and Control, Measurement and Analysis, Requirements Development, Technical Solution, Verification, and Validation.) Instructors are further encouraged to emphasize the distinction between an organization's processes intended for use in product development (or project management) from model process area practices intended as improvement practices to achieve improvement goals.

CMMI Partners

Firms that are SEI Partners are often the "face" of CMMI for many organizations, particularly those smaller, Agile companies who are considering the adoption of CMMI. SEI Partners often are capable of delivering multiple services, including those related to CMMI, and may already be working in some capacity with an organization when the desire to adopt CMMI first appears. In fact, they often play a key role in the discovery of that desire through the types of services they are marketing to their clients. They are thus in a unique position to nurture prospective CMMI user in their understanding of how the model can be implemented in their specific business environment.

CMMI Lead Appraisers

SCAMPI Lead Appraisers and Team Leaders have the greatest impact on acceptance of the message, and probably have a greater degree of responsibility in causing, though often inadvertently, some of the confusion in the software industry between the Agile and CMMI camps. Because appraisals are often conducted after an organization has spent time and money building and deploying their set of organizational processes, a lead appraiser who does not have a solid understanding of Agile methods could, by virtue of conducting an appraisal in a way not conducive or sensitive to use of Agile methods, cause a great deal of harm to an organization adopting an Agile approach.

This problem, though uncommon, could be one reason an Agile organization may avoid CMMI altogether. The perception is that to be successful with CMMI, an organization must meet the *appraiser's* expectations, not the CMMI model's specific and generic goals. While this is an incorrect assumption, it nevertheless persists in the Agile community.

SCAMPI Lead Appraisers and Team Leaders are encouraged to further engage the Agile community by being knowledgeable of Agile methods now in use,[38] including examples of these methods (where appropriate) when training appraisal teams and when discussing CMMI with sponsors and others in the organization.

What About the SEI and the CMMI Model?

Finally, the SEI itself can play a key role in clarifying and eliminating the perceptions that led to the confusion that exists in various systems and software development communities.

The first place this clarification should occur is in the CMMI models themselves. Since the inception of CMMI, and in all subsequent releases, models have included robust sets of examples in the informative material. These examples include suggested work products, subpractices, elaborations, and amplifications that, justifiably, seek to enhance the users' understanding of CMMI, and to assist them in the practical implementation of CMMI specific and generic practices.

[38] Agile methods now in use include Scrum, XP, FDD, and the SEI's TSP.

However, much of the informative material is steeped in the language of traditional systems development. As new informative material is added to the model in subsequent releases of CMMI, the CMMI Product Team will improve the communication and guidance provided by ensuring that the new informative material includes work products, elaborations, subpractices, and amplifications illustrative of and compatible with Agile values and principles.

9.2 A CALL TO ACTION FOR AGILE EXPERTS

Agile experts should recognize that the SEI acknowledges that “level mania” and imposing processes is misuse. Agile experts are encouraged to learn more about CMMI, to seek out and engage in productive conversations with CMMI experts who understand CMMI as it was intended. In doing so, Agile experts will learn the following about CMMI:

- When introduced correctly, CMMI provides transparency, learning, and reuse of what works.
- CMMI focuses on “what” not “how.” This short statement means that a software development approach must be introduced for an implementation of CMMI to be effective.
- CMMI is method and tool agnostic. Also, the use of iteration is just as consistent with CMMI as is the use of a waterfall approach.
- CMMI embodies sound systems engineering and software engineering principles applicable in project contexts that Agile approaches were not intended to address.
- CMMI provides an infrastructure for organizational learning and systematic improvement.
- CMMI provides a path for the effective use of processes, measurement, training, and improvement.
- CMMI uses variation and modeling to reduce waste.
- CMMI fosters a routine that helps to identify and address gaps and lapses in attention to processes.

Agile experts should become more familiar with CMMI practices that complement Agile practices addressing the following:

- non-functional requirements
- product architecture
- measurement and analysis
- planning for non-product oriented tasks
- risk management
- organizational learning

By becoming more familiar with these practices, you can bring about organizational maturity and a broader and more effective deployment of Agile practices.

Finally, Agile experts should help organizations implementing Agile do so in a way that discourages individual heroics by encouraging management of processes, organizational maturity, and long-range value. When faced with a CMMI implementation, Agile experts are encouraged to work to ensure that the organization's product development and project management practices are well defined and have strong ties to business and customer value. In essence, Agile experts are called to participate in ensuring that CMMI implementations in their organizations involve the appropriate staff and are relevant to their context.

9.3 THE BOTTOM LINE

The first real step in the process of reconciliation has been taken and manifests itself in this report. We, as this report's authors and contributors, represent those professionals in the CMMI and Agile communities who are passionate about the subject of Agile methods and CMMI and are concerned with the perceptions that the software industry holds on both sides of the issue. By chartering this group to collaborate and write this report, a demonstrable, concrete step into the exploration and understanding of the issue has been taken, and this *Call To Action* outlines steps that can be taken by experts in both communities that can begin to repair the perceptions and reconcile our two communities. The next step is up to you. All practitioners (whether working with an Agile approach or CMMI) have a responsibility to learn about other technologies (and not just CMMI or Agile) and make the best use they can of the ideas they encounter for the betterment of their project, their organization, and the larger software engineering community.

10 CMMI and Agile Paradigm Comparison

The following table compares and contrasts the CMMI and Agile paradigms along multiple dimensions. This table provides a nice summary of concepts in a way that makes it easier to understand the viewpoints of both CMMI and Agile.

Table 1: CMMI and Agile Paradigm Comparison

Dimension	CMMI Paradigm	Agile Paradigm
Organizational focus of attention	The focus is on the organization or enterprise. Most benefit occurs when CMMI is implemented at organizational level so that all functions and capabilities contributing to the development of products and services are addressed by the process improvement effort.	The focus is on the project and team. Agile methods can isolate (i.e., insulate) the project/team from the organization and still be effective.
Management	Management plays an important role in ensuring project success. There is much attention to project management, including ensuring that plans affecting the project are integrated with the project plan, dependencies are managed, coordination issues are resolved, there is a shared vision for the work, and risk management is performed.	Management is a coaching function (as opposed to traditional command-and-control) that helps to eliminate barriers to progress. This view of management may be expanding as Agile approaches are extended to address larger project contexts.
Trust	Some CMMI practices assume the need to compensate for a low-trust environment (a key concern of the Agilistas). A low-trust environment is often characterized by (1) safety and mission critical objectives, (2) high risk of failure, and most of all (3) multiple stakeholders who cannot be totally transparent relative to their intensions and commitments.	Agile methods originated from the recognition that teams work best when they are composed of task-mature individuals operating in high trust groups. An Agile environment fosters high trust.
Planning	CMMI promotes macro (project-level) planning with an emphasis on establishing a suitable defined process enabling the project to achieve its objectives. Traditional planning approaches assume a long time horizon. Detailed plans are not required by CMMI but many of the examples encourage such an interpretation. Nevertheless, many who use CMMI also use “rolling plans” (detailed only to the next iteration or quarter). There is emphasis on replanning and conditional change.	In Agile methods, there are multiple levels of planning, including high-level product planning (including release planning) and at the beginning of each iteration, more detailed planning around the features to be addressed in that iteration. There is a strong emphasis on flexibility and replanning as conditions change. Use of Gantt charts (that map tasks to calendar time) and graphs of task networks is discouraged because the requirements on which the tasks are based change frequently.

Market/User Assumption	CMMI is broadly beneficial but particularly so when the target market becomes more mature and process innovation becomes a more important differentiator to organization success.	Agile methods have the most benefit in an emergent and not well-understood target/market.
Design Presumptions	CMMI presumes the product architecture is selected or created in the early stages of a project and is revisited when it becomes clear the selected architecture is no longer valid or when using an iterative lifecycle.	Projects are most successful when corporate standard architectures are adopted with flexibility applied as the project progresses. Some Agile methods downplay the importance of architectures on the general principle that such are often prematurely specified.
Learning	Learning happens in many ways, including: (1) organizational training based on analyses of process and skill needs and priorities and the design of appropriate training vehicles; (2) through the development activities of the project itself (e.g., the evaluation of proposed product requirements and solutions for feasibility and cost, schedule, quality, and risk impact); (3) through the planning and use of the processes on projects, measurements, and lessons learned are gathered and shared across the organization; (4) as part of a causal analysis process; and (5) through an analysis of how well the organization's and project's quality and process performance objectives are being met by the processes in use.	Learning happens at project/iteration levels, typically bottom-up and just-in-time (i.e., comparable to Kaizen).
Perspective	CMMI takes, has, and assumes a longer-term view.	Agile takes, has, and assumes a short to medium-term view.
Appraisals	The SCAMPI method compares the organization's processes against the practices of CMMI to evaluate whether the organization has implemented processes that achieve CMMI goals.	The desire of Agile practitioners is that appraisals are made only by looking at results (i.e., customer satisfaction and other project outcomes).
Human Development	CMMI has a limited people focus at the project level. There is an expanded people focus at the organizational level. CMMI does not advocate heroics, but instead creating an environment in which people can excel without heroics (i.e., an effective process appropriately leverages people and technology).	Agile has a team and individual focus (i.e., people over process). There has been a tendency among many in the Agile community to advocate "just hire good people" with an implicit undercurrent of developing "hero developers" within a work-life balanced culture.

Life-Cycle Emphasis	CMMI has a strong “review-as-you-develop” emphasis. There is also a tendency to read in CMMI the approach “verify often and validate at the end,” which is NOT the correct way to understand what CMMI is saying (e.g., see the introductory notes to the Validation process area).[39] CMMI is consistent with an environment in which there is a high cost of failure (see *Trust* above). The objective is to proactively avoid the high costs associated with product failure (or time-to-market failure). The conclusion is that we cannot rely on testing alone. Therefore, CMMI encourages documentation, analyses, and reviews before product components are integrated into a functional product. CMMI also encourages frequent (early and mid-course) validations to ensure the right product is being built. The project determines the appropriate application and timing of these review and testing steps.	Agile methods employ concurrent development, test iterations, and informal peer reviews of work products as necessary. There is a tendency to see in Agile a “validate often and verify second approach” which may be a more correct reading. “Fail early, fail fast, and learn” are central to Agile methods. The trade-off is seen as favoring the development of a useable and testable product vs. developing and analyzing requirements and product components. A low cost of delay or low cost of failure is assumed. There is an assumption of incremental delivery being viable. Burn out can arise from repeated time-boxed iterations, so special provisions should be made for this eventuality (e.g., by revising the time-box duration based on experience feedback and by providing some slack around iterations).
Predictability	Predictability is achieved for critical subprocesses (those process steps that are leading contributors to or indicators of quality and process performance) through *statistical management* (which involves, among other things, use of control charts). Predictability is achieved at the project, iteration, or level by monitoring the performance of critical subprocesses and periodically using the organization’s established process performance models, appropriately calibrated, to evaluate whether the project is on track to achieving its objectives. Finally, at the organizational level, analysis of subprocess performance across projects provides an increased understanding of the capability of the organization’s processes and thus of which areas to target for innovation.	Using time-boxed scope-limited iterations, evolving designs/solutions, driving out defects as early as possible, and failing quickly leads toward a predictable development velocity. There is also an expectation of predictability of iteration delivery and scope of delivery. The anecdotal evidence suggests that Agile teams often struggle with predictability of iteration scope, often de-scoping iterations (or sprints) in the final few days. This de-scoping is rooted in a lack of statistical convergence of the velocity data combined with the analysis technique underlying the scope breakdown.
Cost of Failure	Historically, CMMI was developed in a domain of *high cost of failure*. If a plane crashes, the cost of failure is extremely high. Examples within this domain include the development of aircraft, weaponry, spacecraft and safety-critical medical devices.	Agile methods have flourished in a domain of low cost of failure or linear incremental cost of failure. Examples within this domain include Internet commerce, social networking, and games development.

[39] Verification confirms that the product meets its specifications; validation ensures it meets the needs of the user in the end-use environment.

References/Bibliography

URLs are valid as of the publication date of this document.

[Anderson 2004]
David Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference (ISBN: 0131424602).

[Anderson 2005a]
David Anderson. "Stretching Agile to Fit CMMI Level 3: The Story of Creating MSF for CMMI Process Improvement at Microsoft Corporation." *Agile Conference*. Denver, CO, 2005. http://ieeexplore.ieee.org/iel5/10705/33795/01609821.pdf

[Anderson 2005b]
David Anderson, Sanjiv Augustine, Christopher Avery, Alistair Cockburn, Mike Cohn, Doug DeCarlo, Donna Fitzgerald, Jim Highsmith, Ole Jepsen, Lowell Lindstrom, Todd Little, Kent McDonald, Pollyanna Pixton, Preston Smith, & Robert Wysocki. *The Declaration of Interdependence*. 2005. http://www.pmdoi.org/

[Anderson 2008]
David Anderson. *David Anderson's Website and Blog, AgileManagement.net*, 2008. http://www.Agilemanagement.net

[Boehm 1976]
Barry Boehm. "Software Engineering." *IEEE Transactions on Computers,* December 1976.

[Boehm 2002]
Barry Boehm. "Get Ready for Agile Methods, with Care." *IEEE Computer,* January 2002.

[Boehm 2003]
Barry Boehm & Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley (ISBN: 0321186125).

[Canditt 2008]
Sabine Canditt & Winfried Russwurm. The First CMMI-Based Appraisal in an Agile Environment at Siemens AG. SEPG North America, 2008. http://www.sei.cmu.edu/cmmi/adoption/pdf/Canditt08.pdf

[Cao 2008]
Cao, Lan and Ramesh, Balasubramaniam. "Agile Requirements Engineering Practices: An Empirical Study." *IEEE Software*, January/February 2008.

[Carleton 2006]
Anita D. Carleton & Tim Chick. "Extending the Team Software Process for Systems Engineering." *TSP Symposium*. Long Beach, CA, 2006. http://www.sei.cmu.edu/tsp/sym2006-presentations/extendtsp.pdf

[Charette 2004]
Dr. Robert Charette, Laura M. Dwinnell, & John McGarry. "Understanding the Roots of Process Performance Failure." *CrossTalk: The Journal of Defense Software Engineering,* August 2004.

[Chrissis 2007]
Mary Beth Chrissis, Mike Konrad, & Sandy Shrum. *CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition*. Boston, MA: Addison-Wesley (ISBN: 0321279670)

[Cockburn 2004]
Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams. Boston, MA: Addison-Wesley (ISBN:* 0201699478).

[Cusumano 1995]
Michael A. Cusumano & Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. New York, NY: Free Press (ISBN: 0028740483).

[Dalton 2008]
Dalton, Jeff. *Jeff Dalton's Blog: Ask The CMMI Appraiser*, 2008.
http://askthecmmiappraiser.blogspot.com/

[DeMarco 1999]
Tom DeMarco & Timothy Lister. *Peopleware: Productive Projects and Teams, 2nd Edition*. New York, NY: Dorset House Publishing Company, Inc. (ISBN: 0932633439).

[Elm 2007]
Joseph P. Elm, Dennis R. Goldenson, Khaled El Emam, Nicole Donatelli, & Angelica Neisa. *A Survey of Systems Engineering Effectiveness: Initial Results.* (CMU/SEI-2007-SR-014). Software Engineering Institute, Carnegie Mellon University, 2007.
http://www.sei.cmu.edu/publications/documents/07.reports/07sr014.html

[Fukuyama 1995]
Francis Fukuyama. *Trust: The Social Virtues and the Creation of Prosperity*. New York, NY: Free Press (ISBN: 0684825252).

[Glazer 2001]
Hillel Glazer. "Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity." *CrossTalk: The Journal of Defense Software Engineering,* November 2001.

[Glazer 2008]
Hillel Glazer. *Hillel Glazer's Blog, Agile CMMI Blog*. 2008. http://www.agilecmmi.com/

[Goldenson 2003]
Dennis R. Goldenson & Diane L. Gibson. *Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results.* (CMU/SEI-2003-SR-009). Software Engineering Insti-

tute, Carnegie Mellon University, 2003.
http://www.sei.cmu.edu/publications/documents/03.reports/03sr009.html

[Highsmith 2002]
James A. Highsmith. *Agile Software Development Ecosystems*. Boston, MA: Addison-Wesley (ISBN: 0201760436).

[Immelman 2003]
Ray Immelman. *Great Boss, Dead Boss*. Gurnee, IL: Steward Philip International (ISBN: 0974036919).

[Konrad 2005]
Mike Konrad & James W. Over. "Agile CMMI No Oxymoron." *Dr. Dobb's Portal: The World of Software Development,* March 2005.

[Larman 2003]
Craig Larman & Victor R. Basili. "Iterative and Incremental Development: A Brief History." *Computer,* June 2003.

[Levine 2005]
Linda Levine. "Reflections on Software Agility and Agile Methods: Challenges, Dilemmas, and the Way Ahead." 2005. http://www.sei.cmu.edu/programs/acquisition-support/presentations/reflections.pdf

[Mayer 1995]
R. C. Mayer, J. H. Davis, & F. D. Schoorman. "An Integrative Model of Organizational Trust." *Academy of Management Review*. July, 1995.

[Paulk 2001]
Mark Paulk. "Extreme Programming from a CMM Perspective." *IEEE Software,* November/December 2001.

[Santos 2007]
Pablo Santos. "SCRUM Meets CMMi: Agility and Discipline Combined." *Dr. Dobb's Portal: The World of Software Development,* August 2007.

[Schwaber 2001]
Ken Schwaber & Mike Beedle. *Agile Software Development with SCRUM*. Upper Saddle River, NJ: Prentice Hall (ISBN: 0130676349).

[Schwaber 2006]
Ken Schwaber. *The Enterprise and SCRUM.* Redmond, WA: Microsoft Press (ISBN: 0735623376).

[SEI 2008a]
Software Engineering Institute. *SEI Website: CMMI*, 2008. http://www.sei.cmu.edu/cmmi

[SEI 2008b]
Software Engineering Institute. *SEI Website: CMMI Performance Results*, 2008.
http://www.sei.cmu.edu/cmmi/results.html

[SEI 2008c]
Software Engineering Institute. *SEI Website: Improving Processes in Small Settings*, 2008.
http://www.sei.cmu.edu/iprc/ipss.html

[SEI 2008d]
Software Engineering Institute. *SEI Website: Team Software Process (TSP)*, 2008.
http://www.sei.cmu.edu/tsp

[SEI 2008e]
Software Engineering Institute. *CMMI Website*, 2008, http://www.sei.cmu.edu

[Sutherland 2007]
Jeff Sutherland, Carsten Ruseng, Jacobsen, & Kent Johnson. “Scrum and CMMI Level 5: A Magic Potion for Code Warriors.” *Agile Conference*. Denver, CO, July, 2005.
http://jeffsutherland.com/2007/09/scrum-and-cmmi-level-5-magic-potion-for.html

[Swoyer 2005]
Stephen Swoyer. “Agile Programming and the CMMI: Irreconcilable Differences?” *Application Development Trends,* February 2005.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** November 2008	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** CMMI® or Agile: Why Not Embrace Both!		5. **FUNDING NUMBERS** FA8721-05-C-0003
6. **AUTHOR(S)** Hillel Glazer, Jeff Dalton, David Anderson, Mike Konrad, & Sandy Shrum		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-2008-TN-003
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. **SUPPLEMENTARY NOTES**		
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12B **DISTRIBUTION CODE**
13. **ABSTRACT (MAXIMUM 200 WORDS)** Agile development methods and CMMI (Capability Maturity Model® Integration) best practices are often perceived to be at odds with each other. This report clarifies why the discord need not exist and proposes that CMMI and Agile champions work toward deriving benefit from using both and exploit synergies that have the potential to dramatically improve business performance.		
14. **SUBJECT TERMS** Agile, Agile methods, CMMI		15. **NUMBER OF PAGES** 49
16. **PRICE CODE**		

17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102