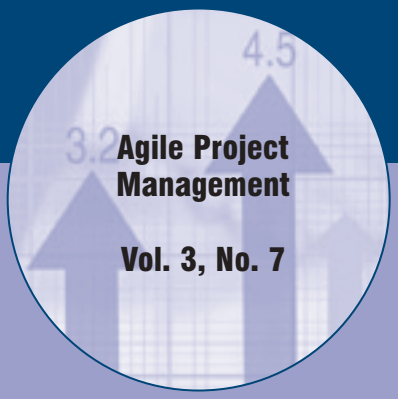

Agile Project Management

Vol. 3, No. 7

Executive

CUTTER CONSORTIUM

CMM Versus Agile Development: Religious Wars and Software Development

by Ken Orr

Fellow, Cutter Business Technology Council

In this *Executive Report*, Cutter Business Technology Council Fellow Ken Orr examines the debate between agile methods and Capability Maturity Model methods. He compares the similarities and differences between the two methodologies and explores the circumstances when one approach might be preferable over the other. In addition to tracking the evolution of best practices for software development, Orr proposes "next practice" as the future method of software development. He presents the next practice quadrants as it "provides an analysis framework ... used to think about alternative and better ways of developing software."

Report

About Cutter Consortium

Cutter Consortium's mission is to foster the debate of, and dialogue on, the business-technology issues challenging enterprises today and to help organizations leverage IT for competitive advantage and business success. Cutter's philosophy is that most of the issues managers face are complex enough to merit examination that goes beyond simple pronouncements. The Consortium takes a unique view of the business-technology landscape, looking beyond the one-dimensional "technology" fix approach so common today. We know there are no "silver bullets" in IT and that successful implementation and deployment of a technology is as crucial as the selection of that technology.

To accomplish our mission, we have assembled the world's preeminent IT consultants — a distinguished group of internationally recognized experts committed to delivering top-level, critical, objective advice. Each of the Consortium's nine practice areas features a team of Senior Consultants whose credentials are unmatched by any other service provider. This group of experts provides all the consulting, performs all the research and writing, develops and presents all the workshops, and fields all the inquiries from Cutter clients.

This is what differentiates Cutter from other analyst and consulting firms and why we say Cutter gives you access to the experts. All of Cutter's products and services are provided by today's top thinkers in business and IT. Cutter's clients tap into this brain trust and are the beneficiaries of the dialogue and debate our experts engage in at the annual Cutter *Summit*, in the pages of the *Cutter IT Journal*, through the collaborative forecasting of the Cutter Business Technology Council, and in our many reports and advisories.

Cutter Consortium's menu of products and services can be customized to fit your organization's budget. Most importantly, Cutter offers objectivity. Unlike so many information providers, the Consortium has no special ties to vendors and can therefore be completely forthright and critical. That's why more than 5,300 global organizations rely on Cutter for the no-holds-barred advice they need to gain and to maintain a competitive edge — and for the peace of mind that comes with knowing they are relying on the best minds in the business for their information, insight, and guidance.

Cutter Business Technology Council

CMM Versus Agile Development: Religious Wars and Software Development

AGILE PROJECT MANAGEMENT ADVISORY SERVICE

Executive Report, Vol. 3, No. 7

by Ken Orr, Fellow, Cutter Business Technology Council

> Today, a new debate rages: agile software development versus rigorous software development.
>
> — Jim Highsmith, Fellow, Cutter Business Technology Council

HOW TO START A RELIGIOUS (OR SOFTWARE) WAR

In 1517, as legend has it, Dr. Martin Luther tacked his famous *95 Theses* on a church door in the small German city of Wittenberg, thus indirectly setting in motion some of the most vicious religious wars in history. In 2001, a number of proponents of various radical software development methodologies roughly aligned with Extreme Programming (XP) met in Salt Lake City, Utah, USA, and issued something they called the *Agile Manifesto*, thereby signaling the beginning of a new software methodology war — one between the old guard, who they saw as being represented by the supporters of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM), and their new agile development movement. So, it would seem, if you wish to start a religious (or software) war, you ought to issue some sort of edict.[1]

[1]When I started writing this introduction, I was quite proud of my analogy of Luther's *95 Theses* and the *Agile Manifesto*. Unbeknownst to me, Cutter Consortium Fellow Robert Charette had not only picked up on the same analogy, but he had done one better by dressing up like Luther at a software conference and tacking a copy of the *Agile Manifesto* to a fake cathedral door!

In December 2001 and January 2002, *Cutter IT Journal* devoted two full issues to "The Great Methodology Debate" (Part I: Vol. 14, No. 12; Part II: Vol. 15, No. 1), which contained the viewpoints of those on both sides of this issue. This *Executive Report* is intended to go more in-depth about the relationship between "agile" and "rigorous" or "light" and "heavy" systems development approaches.

Having been a participant myself in many methodology wars over the years, I don't expect to persuade any of the major combatants. What I hope to do is help clarify what I think this debate is really about and to see whether, at some point, we can come up with new ways of looking at how software development can

be improved even in the most troubled organizations.

SOFTWARE METHODOLOGY WARS

Q: What's the difference between a bank robber and a methodologist?

A: You can negotiate with a bank robber.

Every decade or so, there seems to be yet another software development methodology struggle. During the 1970s, the battle was between various forms of structured and traditional development; in the 1980s, the struggle was between various forms of data modeling and traditional development; during the 1990s, it was between warring factions of object-oriented (OO) design and traditional development. Like most religious wars, the most intense conflict of methodology wars has been between either closely related methods or between those that are furthest apart. Today, in the first decade of the 21st century, the current software development war is between those supporting agile methods and those supporting CMM — a representative of the traditional "waterfall" software development approaches.

In these software wars, hardly anybody gets killed or maimed, but to many of the participants, the war is real enough. Careers are made or lost; organizations thrive or perish. Perhaps the most interested audiences for the software development wars are the CIOs and development managers of large organizations. Just when they are convinced that being a CMM Level 3 software organization represents the current best practice in software, someone (usually one of their younger programmers) informs them that CMM is out and something called Extreme Programming or agile development is in. What is a manager to think?

This report is intended to compare agile software with CMM methods to see how they differ philosophically, how they are the same, and under what circumstances one approach might be preferable to the other.

Our second goal is to shed some light on the hidden agenda precipitating these software wars. Indeed, much of the propaganda produced from both sides is constructed according to similar but differing agendas. One of the most important and common hidden agenda items is power: "Who is in control, the programmer or the manager?" There is a sense in which the agile revolution can also be labeled "the programmer's revenge." During the 1980s and 1990s, there was increasing emphasis on CASE that aimed at eliminating, or at least deemphasizing, programmers and programming. Programmers were forced to do all kinds of design and documentation that they found bothersome and tedious. OO design put development back in the hands of developers; agile development has really turned the control of development over to programmers/developers and users, while somewhat cutting out software development management.

Another agenda item involves process: "Should you have a strict project sequence, or should you be flexible?" Programmers prefer flexibility, but managers prefer predictability. To some degree, the CMM versus agile debate is an argument about how people work best on complex problems.

Another item deals with documentation and design: many programmers see documentation and design as a waste of time, but managers like it. Managers prefer to see tangible things (deliverables)

The ***Agile Project Management Advisory Service Executive Report*** is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Managing Editor: Brooke Spangler, E-mail: bspangler@cutter.com. Production Editor: Allison Make, E-mail: amake@cutter.com. ISSN: 1536-2981.

emerging from development — tangible things that they and users can look at to see whether everyone is on track. Agile developers assert that the only thing people are really interested in is the software itself, which they deliver a little at a time from the beginning of the project.

All of the items on this hidden agenda color the dialogue surrounding the CMM versus agile war. As in any war, the trick is to make the war into a proposition of "us" versus "them" — we're on the side of angels, they're on the other side. CMM proponents are in favor of process improvement and improved control, while agile developers are in favor of increased user involvement and rapid development. But, as we will see, the issues are not so black and white, and as the debate between CMM and agile matures, both sides are changing.

CMM AND AGILE DEVELOPMENT — A LITTLE BACKGROUND

Most successful methods have many predecessors and much history. The individual histories of these methods are often important because they can help explain where certain ideas came from and why they were so important at specific times or in specific problem domains. In this report, we will look at CMM and agile development to get some feel about which domains (or problem areas) each comes from, what kinds of problems each intends to solve, and how each has evolved.

CMM BACKGROUND

The origin of the CMM model is in large-scale military software development. As part of its long-term program to promote improved large-scale software development and software management, the US Department of Defense (DOD), in conjunction with Carnegie Mellon University, set up the Software Engineering Institute in 1984. The charter of the SEI was to promote software development best practices and reuse. Over time, the SEI has become a major force in software research and education, and CMM has become one of the major programs developed and promoted by the SEI. The SEI has taken the role of extending and promoting software engineering in general, and CMM, in particular, outside the defense community.

Watts S. Humphrey

In the 1980s, Watts S. Humphrey, a researcher and manager who came to the SEI from IBM, took the ideas of the software lifecycle and combined them with the idea of quality maturity levels, which lead to CMM. That initial publication then became the focal point for the development of CMM, as we know it today [4].

In a *Crosstalk* article from 1998, Humphrey explained the basic motivation behind the development of CMM:

> The US Air Force asked the Software Engineering Institute to devise an improved method to select software vendors.... A careful examination of failed projects shows that they often fail for nontechnical reasons. The most common problems concern poor scheduling and planning or uncontrolled requirements. Poorly run projects also often lose control of changes or fail to use even rudimentary quality processes [5].

It's pretty clear that if you think most projects fail because of poor project management or change control, then you are likely to come up with a method that, like CMM, focuses on installing improved project control. If, on the other hand, you think that the reason most projects fail is the result of poor user communication, poor development practices, and poor programming, you are likely to come up with a different approach.

The Evolution of CMM

It is easy to see how CMM has been associated closely over its history with the DOD and with military systems. This is not surprising, since the DOD is perhaps the largest long-term purchaser of software development services in the world and its influence has been immense, especially in the earliest days of computers.

The defense folks have been developing large-scale systems longer than anyone and, some would say, more successfully than almost any other organization.

DOD has also been funding research in software development and software methods longer than any other organization, with the possible exception of the telecommunications industry.[2] The reason for this interest in software development is that critical DOD systems tend to be immense and state-of-the-art, relying on huge amounts of custom hardware and software. Moreover, almost all DOD systems are developed using outside contractors. With so much money at stake, DOD has worked mightily to take as much risk out of their critical systems projects as possible.

For decades, DOD has been promoting rigorous systems methodologies. In the 1950s and 1960s, for example, DOD began to promote the first waterfall methodologies adapted from its work on hardware development.[3]

This approach emphasizes the idea of a serial, phased approach to development: planning, requirements, design, development/testing, and deployment. Each phase produces voluminous documentation used by the next phase.

The waterfall approach allows careful consideration of problems and ensures that major elements are not missed. This serial approach also makes it possible to break up the work so that one organization can do the planning and requirements, while another can do design, development/test, and deployment — a standard procurement process with government projects.

Because DOD was such a big customer, many large military/aerospace firms adopted (or were forced to adopt) waterfall methods very similar to DOD's.[4] As a result, those organizations with the most rigorous methods were often large, military software vendors. This pattern is being repeated today in the private sector. As more and more large organizations are buying huge chunks of their software development from outside vendors, they are looking for better ways to manage this important relationship. And since the DOD has had the most experience in controlling outsourced software projects, these organizations are taking a clue from DOD's efforts and are beginning to force the adoption of CMM practices among their vendors.

One particular segment that has picked up CMM is the outsourcing software vendors, especially those in India. For more than a decade, large Indian software companies have been among the most aggressive in the world in pursuing higher and higher levels of CMM certification.[5] This certification was initially sought to allay fears that customers might have about having software work done by Indian firms, and then it was used to convince people that it was all right to move development to another continent. Cutter Business Technology Council Fellow Ed Yourdon, who sits on the board of directors of an Indian software company says that, by now, many of these companies have become experts at following CMM best practices and have even found ways of applying CMM management techniques to small projects.

CMM and Quality

Many ideas behind CMM are taken directly from statistical product quality work pioneered by experts like W. Edward Deming and Joseph Juran, particularly the principal idea that organizations improve quality by gaining increasingly greater (statistical)

[2]Most students of technology recognize the importance that the DOD Advanced Research Project Agency played in the development of high technology. This organization gave early support of research into computers, software, databases, and, of course, the Internet. Other major technology research funding came from various military and intelligence services.

[3]To be honest, DOD has never been as successful in its software management activities as it has with its hardware, but, then again, no other organization has been uniformly successful when it comes to developing really large, state-of-the-art software systems.

[4]DOD has reinforced this by making systems development and systems management methods part of the standards that vendors must comply with to get DOD projects.

[5]According to the SEI, roughly 40% of all CMM Level 4 and Level 5 certifications are for Indian firms.

control of their processes.[6] This idea is used worldwide within the quality movement. It is not only one of the guiding ideas behind CMM; it is also one of the guiding ideas behind other quality initiatives, such as ISO 9000.[7] The idea of capability levels actually comes from the quality world; indeed, both the terms and definitions for the levels (initial, repeatable, defined, managed, and optimized) are taken directly from other quality work.

The Status of CMM in the World of Software Development

In the complex world of software development outsourcing, CMM has become a kind of high-level certification. In the same way that becoming a certified Oracle database administrator or certified Microsoft software engineer have become valuable on a résumé, becoming a CMM Level 3 organization has become a widely accepted certification in software marketing. Becoming a Level 4 or 5 organization is supposed to make you a top software engineering firm.

An interesting note is that in talking to some of my CIO friends, they are somewhat bemused by the emphasis on becoming a CMM Level 3 organization, but they are more than ready to use it as a selling point with management on outsourcing large tasks. On the other hand, many of these same managers are somewhat leery of organizations that represent themselves as Level 4 or 5. Most of these CIOs are afraid, I assume, that Level 4 or 5 will be too costly to do business with because they are too formal and too controlled.

The CMM Levels

CMM has five levels (see Figure 1):

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimizing

Level 1: Initial

Level 1 is fundamentally a software process that is ad hoc. There is no specific methodology, and each project is more or less a new activity. The initial stage is the default in software management. According to Joseph Raynus, President and CEO of VistaPortal, "Unfortunately, the initial level processes are the most practiced processes in the software business" [6].

Ultimately, the most serious problem with most Level 1 organizations is that management doesn't realize it's at that level. Like an

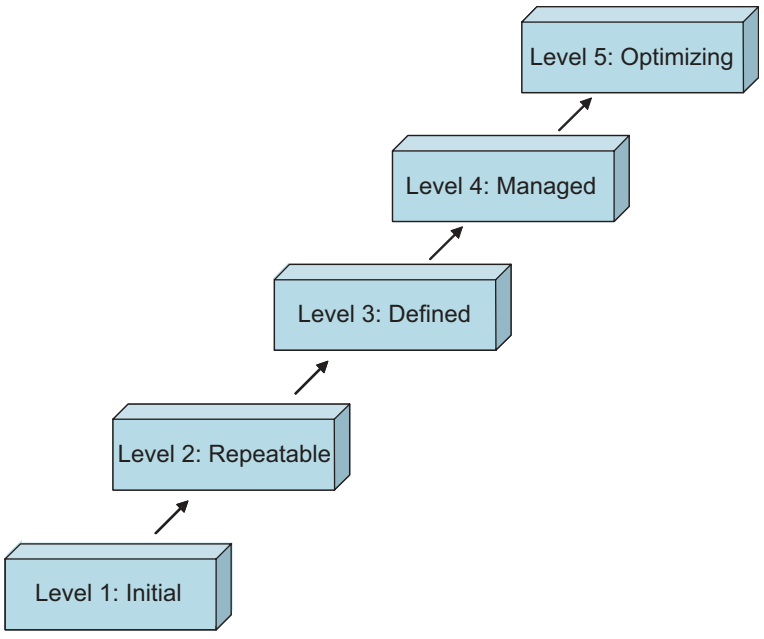

Figure 1 — The Capability Maturity Model (CMM) ladder.

[6]It is important to note that CMM means mostly software management processes such as estimating, scheduling, quality control. The CMM has historically been faulted for its lack of interest in product development process, which we will see is where agile development excels.

[7]One of the most controversial ideas behind CMM is whether the software development process can be managed as closely as a manufacturing one. Indeed, there are a number of major thinkers that consider the idea of quality CMMs as fundamentally flawed.

alcoholic who can't be helped until he or she admits the problem, Level 1 organizations typically can't be improved until management recognizes that it is, in fact, a Level 1 organization. In the real world, this recognition often occurs when the organization wants to market to another organization that insists its vendors have a CMM process in place.

Level 2: Repeatable

Basic management exists within Level 2 organizations. Level 2 organizations have a basic project management structure in place that tracks costs, schedules, and, to some degree, systems functions. As the name implies, the software process in place is repeatable and reinforced by training and management controls.

Repeatable level organizations use the same processes time after time. Because the organization has repeatable processes, it can begin to work on measuring and improving these processes. It can also begin to work on the development process itself.

Level 3: Defined

At Level 3, a software development organization has a well-defined software engineering approach as well as a project management framework. Here, the processes are well documented and communicated across the organization through standards and training. Raynus says, "If the repeatable level defines what to do and who should do it, the defined level specifies when to do it and how to do it" [6].

Though Level 2 organizations are mostly concerned with project management processes, Level 3 organizations are, theoretically at least, also concerned with defining and measuring software development processes. This difference represents a major difference in quality.

Level 4: Managed

At Level 4, software development is increasingly managed using statistical methods. Both process and product quality measures are collected and monitored to quality controls. At Level 4, software products are not only managed to meet time and cost schedules, they are also managed to meet specific customer satisfaction and defect levels. Level 4 organizations understand and practice statistical quality control.

Level 5: Optimizing

At Level 5, organizations reach quality nirvana. Software is produced as in Level 4, but the organization is continually working to improve its processes and standards. Level 5 organizations are world-class innovators in software development.

The idea behind CMM is that through a certification process, organizations can be checked out, their current state can be identified, and management can be provided with a roadmap for improving their performance. By committing to the CMM process, organizations can start wherever they happen to be, and through a conscious statistical quality control program can begin to improve, just as the total quality management (TQM) gurus say. All it takes is commitment, training, measurement, and correction.

In a recent conversation with Council Fellow Ed Yourdon, he discussed the experiences of the Indian software company whose board of directors he sits on. This firm had embarked, as had so many Indian firms, on CMM to help in marketing to the US. Over time, however, they had become such experts in its use that they are now using it on smaller and smaller projects. Indeed, it is now part of the organizational culture.[8] This organization had pushed even further by putting management practices on the Web for all of their clients to see in real time.

The Pluses of CMM

Like many quality-based initiatives, CMM appeals to organizations and managers interested in control and certification since it

[8]Council Fellow Ed Yourdon remarked that there was less cultural resistance to the kind of structure imposed by CMM in India than in the US. How much of a role cultural differences play in implementing rigorous software processes is still an open question.

gives them some objective qualification of their software development process. The process is normally that an outside CMM organization will come in to certify an organization for a specific CMM level. This is important in organizations where CIOs and others are looking for ways to add outside creditability.

CMM certification was first required, as you might expect, on DOD projects, and, over time, it has been extended to a variety of federal, state, and local agencies. Today, many large private organizations lean heavily on CMM certification as a means of screening software vendors. As previously mentioned, certain outsourcing vendors, especially the large "body shops" in India, see high-level CMM certification as both a marketing and management tool. By showing that they are independently certified, they increase their creditability and reduce client concerns about moving development out of house or out of the country.

Also, as we've mentioned, some organizations have gone a step further by exposing their management processes to their clients via the Web, allowing customers to track their CMM processes in, more or less, real time, and this trend is certain to continue.

Furthermore, there is certainly nothing wrong with being well managed. Software is not such a unique business where good management practices don't apply. In many respects, CMM represents applied common sense. On the other hand, it is important to understand where it comes from (i.e., large outsourced projects).

CMM is clearly the best-known, best-documented software management method in the world. There are tens of thousands of managers and programmers who have been trained in its use and who find CMM as easy shorthand for communicating with others. Being recognized as an industry best practice has many advantages to an organization trying to better itself.

The Minuses of CMM

On the negative side, many people have argued that CMM certification appears to be somewhat arbitrary. Organizations that certify in CMM make their money by showing that it has helped organizations move up the ladder to Level 5. Getting certification, at least at Levels 2 and 3, often seems to be more associated with good paperwork than with good management. It is, after all, very difficult for an outside organization to come in, and, in a very short time, determine whether an organization is doing what it says it's doing. Just look at the current accounting mess we're in with Enron, WorldCom, and others.

A second major concern is that CMM certification is largely about an organization's management processes (estimating, scheduling, control) and not nearly so much about the quality of the software products produced.[9] There are even those who say that to become a Level 2 or 3 organization, a company doesn't necessarily need to have a good (or state-of-the-art) software process in place but only must blindly follow whatever process it has.

Other critics say that CMM certification is not so much of a quality certification, but a marketing one. As with ISO 9000, organizations seeking CMM certification bend over backward to produce reams upon reams of documentation that can then be reviewed by CMM inspectors. (In defense, we noted earlier that although many of the Indian software organizations came into CMM for marketing purposes, the process generally becomes ingrained and produces much better overall software management.)

No matter what your viewpoint, though, one thing is clear: CMM Level 3 or 4 ordinarily involves a very strong (heavy-handed) management style. By and large, more attention in these organizations is paid to project management than to actual production. This is a matter of concern for many software developers who would much rather design and program than fill out project management forms.

[9] In response to the criticism about CMM's lack of objectivity, the SEI has created a measurement program to support CMM certification; this program is called the Software Engineering Measurement and Analysis Initiative.

Some Comments on CMM

In the end, CMM depends upon comparing performance with management. Perhaps the best analogy is with sports. CMM proponents suggest that we compare a little league baseball team with a major league one, if you want to see the difference, let's say, between Levels 1 and 4. And, if you want to see the difference between a Level 4 and Level 5 organization, look at the New York Yankees against any other major league franchise. (Note that there is little discussion here about the skill of the individual players.)

All teams and all software organizations do the right things (such as plan, define, design, code, test, and install). The difference between the good ones and the bad ones is that the good ones do the right things in the right sequence more often. The great organizations keep looking at what they are doing and try to find ways to do it better. An organization at Level 5 keeps reinventing itself to incorporate new people, new technology, and new management strategies.

Some Keys to Success with CMM

If you want to do business with defense agencies and with an increasing number of large corporations, you need to show CMM certification. Some organizations use this as an opportunity to make major improvements in their software processes. Other organizations see it simply as something else they must do in order to do business with the government. The ultimate danger of CMM is that it can easily harden into mindless paperwork. In a discussion with Cutter Consortium Fellow Robert Charette, he remarked that this freezing in place was the biggest problem with large organizations using CMM.

Just like TQM, CMM must be worked on every day to be of value. New approaches need to be evaluated, and CMM must be modified to keep up with the times. CMM software development managers have to take a "sundown" approach to processes and forms. If a process doesn't make sense any more, or, if some form is out of date, it ought to modified or let go.

In addition, CMM managers have to work to maintain organizational flexibility. In particular, they need to ensure that they promote face-to-face communication whenever possible.

Finally, CMM needs to be automated. If some particular bit of information needs to be kept up to date, or some formal approval needs to be given at a certain point, then these things ought to be automated so that they are as easy to do and are integrated as possible. Of all the approaches out there, CMM must be constantly reengineered.

Because of its tendency toward rigidity and bureaucracy, CMM in the wrong hands can stifle all innovation. Pretty soon, if you're not careful, CMM becomes all the bad things that its opponents say about it.

THE HISTORY OF THE AGILE MOVEMENT

Unlike CMM, agile development is a product of the 1990s, not the 1970s or the 1980s. For the most part, the people most responsible for the agile movement, with some notable exceptions, came of age at the beginning of the "object revolution" in the late 1980s and early 1990s. Cutter Consortium Senior Consultants Kent Beck and Alistair Cockburn along with Martin Fowler, chief scientist at ThoughtWorks, were involved at one time or another in teaching and promoting OO techniques.[10] In its early days, the object revolution promised to sweep the planet and completely change how people did software.

Today, the object revolution has been very successful in many ways; today's most popular development languages and operating environments all have a definite OO flavor. But not everything has gone as planned. For example,

[10]There are some notable exceptions, of course. Council Fellow Jim Highsmith, author of two influential books on agile development [2, 3] is a pioneer in systems methodologies going back to the 1970s. Cutter Consortium Fellow Robert Charette, Steve Mellor, and Ken Schwaber are other greybeards that have joined forces with the agile camp.

components have not been as big or as easy to use as OO gurus predicted. Even more disturbing, OO projects have often been as late and over budget as non-OO projects. As a consequence, some of the most devoted OO advocates came to recognize by the mid-1990s that OO techniques alone would not revolutionize the software world; there needed to be a new way to build software.

But as those promoting OO techniques found, OO design and coding were not enough to take full advantage of the new opportunities that OO offered. To truly create new classes of software in very short periods of time, the software process itself needed to be radically overhauled. A number of different approaches of radical new OO development were experimented with, and agile development was born. Today, agile development represents the coming together of a number of the most successful approaches built around a common set of ideas.

Among the most important early themes of the agile development movement include the following:

- Systems are best developed in short increments.
- Users and developers must work hand in hand.
- Each systems increment should be designed to handle the minimal requirements.
- When changes in requirements occur, the agile development folks believe that they should be designed in rather than added on and that there should be minimal documentation beyond the code.

Most former OO programming gurus involved in the agile movement have a general dislike for paperwork, especially much of the paperwork imposed by rigorous software approaches such as CMM. In the circle of agile gurus, a strong minimalist bias exists. The software is alpha and omega, the beginning and the end; everything else is overhead. The ultimate argument goes like this: "Why worry about documentation or project management forms, etc., if all that people are really interested in is the product itself?" Then, "What if we throw a small number of really good programmers in a small room with the user and let them develop the software a little at a time. That way, you can eliminate all of the waste and miscommunication." And so was born XP, the father of agile development.

Instead of being imposed upon by some large organization like the DOD, agile development was actually a somewhat spontaneous development of a number of independent consultants who came across many of the same ideas.

One of the most influential and widely followed gurus in the agile development movement is Cutter Consortium Senior Consultant Kent Beck. Beck has written numerous books and articles on XP and has lectured around the world. After working for a number of organizations including Apple and Hewlett-Packard, it was his work on a project for Chrysler in the mid-1990s that allowed him to begin to formulate his ideas on a radical new way of developing systems.

Throughout the late 1990s and now into the 21st century, an increasing number of people became associated with XP or other fast-track methodologies. A number of books came out of this work, and more and more organizations became interested.

Unfortunately, there were many ships flying the flag of extreme, incremental development, and software developers and managers were having a tough time understanding which methodology did what. Finally, in early 2001, a group of agile gurus met in Salt Lake City, Utah, USA, to see whether they could come up with a set of common principles. Out of that meeting came the famous *Agile Manifesto*.

The *Agile Manifesto*

The *Agile Manifesto* as put forward at Salt Lake City was a very simple and straightforward set of common principles (see Figure 2).

Remember, if you want to start a religious or software war, issue an edict or manifesto. The issuing

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

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.

Figure 2 — The *Agile Manifesto*.

of the *Agile Manifesto* represented a watershed event for those involved with radical methodologies — the Rubicon had been crossed; the gauntlet had been thrown down. It began the war with the establishment, or at least those supporting CMM.

As we will see, there are some fundamental differences between the agile movement and those who support CMM, but then again, their goals and objectives are also different. First, we will discuss the full set of process and concepts that advocates of agile development propose, and then we'll discuss the audience that agile development addresses.

The Agile Development Approach

If CMM is a set of loosely connected software management best practices, agile development is a set of highly integrated development and management practices.[11] In his book, *Extreme Programming Explained* [1], Beck proposes 12 different ideas that are key to his form of agile development:[12]

1. The planning game
2. Small releases
3. Metaphor
4. Simple design
5. Testing
6. Refactoring
7. Pair programming
8. Collective ownership
9. Continuous integration
10. 40-hour week
11. On-site customer
12. Coding standards

[11] There are a number of other agile methodologies that have signed on to the *Agile Manifesto*. Some of the better known are Scrum, Dynamic Systems Development Method (DSDM), Feature Driven Development (FDD), and Crystal Light. All of these methods share most of the basic tenets with XP, especially the high level of user involvement and the importance of rapid prototyping.

[12] Throughout this section, I use XP and agile development interchangeably since XP is arguably the best known and best documented of all the agile development approaches.

The Planning Game

For Beck, "software development is always an evolving dialogue between the possible and the desirable" [1]. Neither the customers (business users) nor the technical folks should have complete control. The business users decide scope and priorities, as well as the time and scope of the product releases. The technical people, on the other hand, decide estimates, technical consequences, the development process, and detail scheduling. In XP projects, the planning game is a give-and-take process emphasizing true collaboration.

Small Releases

Beck says, "Every release should be as small as possible, containing the most valuable business requirements" [1]. There is a saying in my organization (the Ken Orr Institute) that a doable project is one that is small enough to be done quickly and big enough to be interesting to the business, customer, etc.

Metaphor

Every project should have a single, overarching concept or metaphor. Metaphor is intended to replace the term architecture within a project and provide it coherence. According to Beck, "As development proceeds, and the metaphor matures, the whole team will find new inspiration from examining the metaphor" [1].

Of all the components, the metaphor is perhaps the most difficult one to grasp. At some level, it is supposed to be the essence or the irreducible core of whatever it is your project is building. When done right, it makes it possible for team members to keep focused. We'll discuss more about metaphors later in this report.

Simple Design

From Beck's standpoint, "The right design for software at any given time is one that:

- Runs all the tests.
- Has no duplicated logic. Be wary of hidden duplication like parallel class hierarchies.
- States every intention important to the programmers.
- Has the fewest possible classes and methods." [1]

As you might imagine, "simple" has lots of meanings to lots of people, and I can imagine a number of scenarios where Beck's definition of simple might not turn out to be so simple after all.

Testing

"Top XP teams practice 'test-driven development,' working in very short cycles of adding a test, then making it work. Almost effortlessly, teams produce code with nearly 100% test coverage, which is a great step forward in most shops," writes Beck [1].

Test-driven development means that agile development starts by defining tests before developing code. It is easy to overlook the importance of this idea. In doing so, programmers are forced to work out the inputs and outputs that they are trying to achieve and then to build code that produces the correct results from the predefined tests. Like so many factors in agile development, the basic strategy rather than a set of heavy-handed management controls is responsible for good code and good programming discipline.

Refactoring

Refactoring is one of the most interesting ideas of agile development. In essence, refactoring involves the constant redesign of the entire OO class hierarchy, and methods such as new requirements come to be known. Agile development programs evolve through iteration. At each release, new requirements are introduced. Each new set of requirements mandates that the programming team go back and design these new requirements into the existing code.

Refactoring goes hand in hand with the idea of minimal requirements. One of the basic tenets of XP, and therefore, agile development, is that of implementing only those requirements that are known for certain and that are involved in the current release.

Historically, one of the problems with radical iterative design is that after a number of iterations, the program(s) becomes increasingly difficult to maintain. As change after change is added to the program, it becomes more and more difficult to understand. However, with the idea of refactoring, the design is revisited at each iteration, and changes are integrated, thus making the systems evolution smoother.

In my opinion, refactoring is one of the key ideas of agile development. If agile development's form of iterative development with minimal documentation is to succeed, then there must be an increased emphasis on design. Refactoring really means design/redesign.

Early versions of OO design emphasized spending enormous amounts of time designing class hierarchies and methods before building an application using the classes and methods. But this made early OO very front-end loaded. A great deal of time was usually spent on the initial classes, only to discover that you always had to redo them as the project progressed and you learned more about the problem. In OO, this

became known as "class thrashing."

In agile development, there is minimal design up front, but that minimal design is repeated for each iteration. The belief is that if you do it right, the program will improve, not degrade over time. There are some difficulties with this approach, however. For one, manually refactoring OO code is not an easy task, and, for another, some kinds of refactoring (i.e., data refactoring) are much more difficult than others.

Unfortunately, even though agile development, coming as it does from OO, is a highly technical approach, there is little discussion about the importance of automated tools in such critical areas as class and method refactoring. Because of the property of inheritance, changes in one class often ripple through a system in unpredictable ways. Recently, tools have begun to make this process faster and more reliable.[13]

Class and method refactoring are only one part of the problem. An equally large problem that has been largely ignored by agile development is the importance of data refactoring. Most of the major problems associated with redesign occur whenever it is necessary, for whatever reason, to redesign the underlying database.

[13]My own personal opinion is that automated refactoring tools will become more and more important to the success of agile development projects.

Historically, OO design has been somewhat light on database design and redesign. In the analysis section of this report beginning on page 14, we will discuss both the importance of data refactoring and automated refactoring tools, of which automated database redesign tools are the most important.

Pair Programming

Pair programming is just what it sounds like — all programming should be done by pairs of programmers working so closely together that one can pick up and/or modify the work of the other at any time.

Agile developers maintain that pair programmers working together are far more productive over the long haul than two programmers working independently. They maintain that by working in tandem, the design of each part of the code is made more clear and easier to comprehend and that, by working in pairs, individuals learn better habits and produce more than they would on their own.

The pair programming approach institutionalizes the practice of code reviews. Each day, each pair programmer must understand all of the code that he or she is responsible for no matter which partner writes it. Agile development maintains that the result is better, producing tighter code that is easier to understand. Used in conjunction with test-first development, pair programming speeds the development process by forcing both programmers to use the same (or at least a common) design strategy.

Pair programming in an agile development or XP environment creates an atmosphere where people know more because they are involved more. According to Beck, "If two people pair in the morning, in the afternoon they might easily be paired with other folks. If you have responsibility for a task that is unfamiliar to you, you might ask someone with recent experience to pair with you. More often, anyone on the team will do as a partner" [1].

Pair programming is knowledge management par excellence. By making sure that for every function there are two people who completely understand what is going on, there is far less reliance on one key individual who might leave, get sick, or otherwise become unavailable.

Collective Ownership

"In XP, everybody takes responsibility for the whole system," Beck says [1]. In agile development, the central product of the organization — the software — is owned by the group. Instead of individual programs being more or less exclusive property of one individual from concept through turnover to maintenance, the entire set of code is the property

of everyone in the group. Anyone can look at any piece of code and can, within the boundaries of pair programming, change anything as well.

Continuous Integration

Because of its emphasis on speed, agile development focuses on constant integration. Rather than have a set of periodic program (system) builds, the code base is in a state of continuous integration. New programs can be added at any time. The pair programming teams integrates the code base. If something fails, typically the team that has made the most recent changes is normally the group that tracks down the problems. From work in TQM and lean manufacturing, it is well known that the shorter the cycle between manufacturing and testing, the better the ultimate end product.

The 40-Hour Week

Agile development gurus preach that software development is more like a marathon than a sprint. In knowledge-based work, it is important that the individuals that make up the team be as fresh as possible. Although there is not much that software development managers can do to help team members avoid stress and burnout at home, they can certainly do so on the job.

The rule, then, on agile development projects, is that people in the normal development cycle only work 40 hours a week. Though there may be occasions for overtime, there shouldn't be overtime worked more than one week at a time: in other words — no death march projects.

The On-Site Customer

States Beck: "A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. By 'real customer,' I mean someone who will really use the system when it is in production" [1].

There is an old joke about involvement versus commitment: if you're fixing bacon and eggs for breakfast, the chicken is involved, but the pig is committed. This is absolutely true of the customer/user on an agile development project. People who are going to be using the system are responsible for defining how that system is going to work, and their commitment is for the period of the entire project.

The total commitment of the user is one of the major hallmarks of an agile development project. Users are as responsible for the definition, user interface, review, test case presentation, and prototyping of the end product as anyone on the team. Because of this, there is very little finger-pointing when the product is delivered for broad installation and use.

Coding Standards

Finally, there is the issue of coding standards. Because ideas of pair programming, collective ownership, and (let me not forget to mention) minimal documentation, it is critical that all of the code follow very strict guidelines. In this environment, "you simply can't afford to have different coding practices. With a little practice, it should be impossible to say who on the team wrote that code" [1].

Comments on Committing to Agile Development

Even the most casual reader will note the kinds of differences between CMM strategy and that of agile development. CMM is an organizational religion. It is intended to define relationships between large organizations, not unlike the medieval church. Agile development, on the other hand, if not a personal religion, is surely the religion of a small group (the team). Where CMM defines large external measures of obedience, agile development focuses on individual/small group responsibilities.

Where CMM is often referred to as a heavy or rigorous method, it is clear that individual contributions in an agile development "church" are much more closely scrutinized by the fellow members of the local church. Similar to many of the early Protestant sects, agile development insists on very proscribed behavior, which, in turn,

assumes a high skill level among the team members.

A second factor that one observes when looking at agile development is how closely a series of psychological group dynamics, beliefs, and research are woven into the approach. People work better in groups, so we program in teams; code review is good, so we always have pair programming; output-oriented design is good, so we incorporate test-first development; people work better with enough rest, so we institute the 40-hour work rule.

In a typical CMM environment, organizations can organize any way they want or use any method they want, as long as they commit to doing the same things the same ways, measure what they do, and try to improve it. Agile development is a much more tightly integrated package. You can't decide to implement items 1, 4, 6, and 9 (out of the 12 ideas Beck proposes) and leave out the rest. Indeed, if you listen to many of the agile development gurus, you need to commit to the entire program if you're going to be truly successful — each of the 12 items above is there for a reason, and taking any one item out can jeopardize the entire program.

It's interesting to note here the similarity between Beck's 12 practices and Deming's 14 points of management. Like Beck, Deming stresses that you need to do all 14 if you are going to be successful in instituting a successful quality program. Organizations almost always pick and choose which ones to include in their specific program. For example, organizations routinely ignore a number of Deming's rules, especially the ones about no slogans, driving out fear, and breaking down barriers between departments.

Agile development aims at forever changing the way people develop software. In my mind, it is a much more rigorous management approach than CMM. Agile development, like Deming's quality, requires a much stronger commitment up and down the line. Moreover, agile development, like Deming's practice, requires a commitment to a very different style of management. In CMM, we see most of the responsibilities being placed upon the systems and project managers. In agile development, most of the responsibilities are placed upon the team. Teams are responsible for setting priorities and schedules, designing, testing, and delivering. The product rolls out in small increments, but in an almost Japanese fashion, where individuals are not accountable except as part of a team.[14]

Comments on Minimalist Design and Refactoring

> Deliver quickly. Change quickly. Change often.
>
> These three driving forces compel us to rethink traditional software engineering practices.
>
> — Jim Highsmith

One of the more controversial concepts underlying all of agile development is the idea that the product (prototype) should be developed by taking into account only those things the user currently wants and are specified for the current release. This is a controversial idea because it seems to many experienced managers and developers as simply an extreme form of hacking. Rather than see this as an essential ingredient within an overall project philosophy, many managers see this as simply a way to avoid real, long-term requirements and design consideration.

The defense that the application developers give for working only on the things that they know right now is that if one begins to consider future (unknown) requirements rather than current (known) requirements, there is no easy place to stop. When you're thinking about all the different ways this product may be used in the future, do you stop at three, five, or 10 years in the future? And since you are, after all, guessing about what those future requirements might be, there is a good chance that you will guess wrong. Agile developers maintain that it is better to focus on what you know, and be prepared to change.

[14] In true team environments, the team succeeds — individuals only succeed as part of the team.

Personally, I have always favored a minimalist approach. As it turns out, the methodology that my colleagues and I evolved back in the distant past worked in large part because, like agile development, it was both minimalist as well as highly incremental. So an approach that involves designing each increment just for what you know at that particular point in time has special appeal. Over the years, this design strategy has helped my colleagues and me create systems that are much easier to understand, design, and implement.

Our approach, like that of agile development, is that when you develop the next increment of an incremental project, you take the new, modified requirements and do a complete redesign, integrating the new requirements with the old. In agile development, this is not called redesign, it is called refactoring, but it amounts to the same thing. Clearly, then, whether this approach is successful over the duration of a large project is a function of how good you are at refactoring.

The trick to having a clean, elegant design in an agile development environment is to build things in, not add things on. In practice, this grows more and more difficult as the size of the program gets bigger and bigger and pressures to finish the product/system grow.

Comments on the Agile Manifesto

If you talk to the people who signed the *Agile Manifesto*, you will find that they are by and large proud it. They might not be as committed as say the framers of the US Declaration of Independence, but the statement reflects what they think is important.

On a second level, the *Agile Manifesto* achieved another, perhaps unstated goal, which was to draw a line in the sand. Here, the authors said, we're in favor of people, working software, customer collaboration, and responding to change — take that all you folks using heavy, rigorous management methods (e.g., users of CMM)! It was clearly a way of getting people's attention — and it worked. Like everything else involved with agile development, I think that it was very carefully thought out, but as a propaganda piece, it may go down with "loose lips sink ships."

My quarrel with the *Agile Manifesto* is that, in the long run, it creates, or seems to create, artificial barriers to future improvements in the software development process. I believe, for example, that for something like agile development to be adopted widely and really affect software productivity and quality, it will be necessary to place as much emphasis on rigor, documentation, and tools as we do on people, collaboration, and working software.

Some forms of refactoring, namely data refactoring, are simply too complex to be done by hand, no matter how skilled the practitioners or how good the team. Saying that we value people over tools deemphasizes the development of tools that make agile development work better, faster, and cheaper.

Tactically, I think that issuing the *Agile Manifesto* was a good idea. Strategically, I think that having drawn a line in the sand, the agile development folks may have to stand on the wrong side. Over the long haul, individuals and interactions are not opposed to process and tools. Neither is working software necessarily at odds with comprehensive documentation — documentation using the right tools should be a byproduct of producing working software. Neither is customer collaboration necessarily at odds with contract negotiation, though here I'm inclined to give the agile developers some slack on this issue — contract negotiation is usually a lose-lose activity. Finally, responding to change is not fundamentally opposed to following a plan.

Now the agile development folks will say that they never said they were against tools and processes, just that individuals and interactions are more important. Granted! But there are few generals in the world who wouldn't rather have a group of relatively untrained recruits with rifles against the

most skilled folks with bows and arrows. Civilization and software operate at many levels; major advances in civilization often result from seemingly simple advances in technology.

The Pluses of Agile Development

To begin with, agile development is all about the end product. In an agile development project, the product is always the clear focus — not the requirements, nor the design, but the final, delivered product. Having the product as its main focus makes up for a lot in the real world of project management.

A second major advance that agile development brings to software is institutionalizing the process of incremental product releases, while, at the same time, challenging the time involved to produce them. Most nontechnical people respond best to something they can see and touch.

Another plus is the focus on integrating many programming best practices into basic principles that all work together. The agile development folks have done an admirable job of taking a lot of things that have been known for a long time — how programmers work best and how to produce quality software — and blended them together into a single coherent whole. After studying it for a while, it strikes me that agile development is quite a lot more than the sum of its parts.

The Minuses of Agile Development

Right up front, agile development presents a number of hurdles for many organizations. One such hurdle is that unless you're already committed to OO design, you may have to start there. Most of the agile development gurus start out assuming that OO development is the basic platform.

Next, there are a number of organizational issues. Many organizations have trouble with activities where the team, not the individual, is the key. Personally, I find this refreshing, but for many hard-line managers, it may be difficult. And then there is the issue of the full-time user. In many organizations, it is simply impossible to get the full time of many of the key people. I know that some in agile development shrug and say, well that's tough, no full-time user, no product or system; however, the issue of user involvement can be a deal breaker.

It would seem to me that working on an agile development team would not be everyone's cup of tea.[15] In most North American organizations, programmers are brought up to be responsible for their individual activities. Many programmers went into programming because they like to work alone — just them and the machine. Agile development calls for a very socialized individual. It may be a lot better; it's just not the way that many people have been brought up.

A more significant issue for agile development is the concept of the user. All large systems have many users, not just one. Getting to the requirements with many users is not a simple task. When I see people talk about the user or the customer, I'm always somewhat amused because it shows that there is a fundamental problem understanding the nature of the problem. If you have a customer, then there is some chance you will have a unified set of requirements. If you have customers, you will always have conflicts and ambiguity because multiple people will always have different viewpoints. And having a single person stand in for the customer doesn't necessarily solve your problems; you're still one level removed from the real customers.

Some Keys to Success with Agile Development

The keys to being successful with agile development are almost the mirror image of those you need to be successful with CMM. Where CMM tends toward rigidity over management, agile development tends to be perceived as out and out hacking with an overall lack

[15] Recently, a friend of mine related a story of why "team teaching" had failed in a specific university setting. It was concluded that team teaching failed because many, if not most, university professors go into teaching because they prefer working alone. Even if team teaching might be better for students, it didn't fit the personality profiles of the teachers.

of discipline.[16] Agile development projects also have a tendency to become inwardly focused.

Agile development differs most from traditional systems approaches in organizational matters. As a result, if your organization is going to be successful using agile development, then it must face the possible organizational fallout issues early on.

When agile development gurus say that you need an on-site customer, they mean it. They don't mean half-time commitment or quarter-time; they mean full-time. This is very hard for many organizations to meet. Indeed, agile development not only calls for the full-time commitment of users, it requires the key most knowledgeable users — the most difficult people in the organization to get your hands on. The only thing that can be said in the defense of agile development's insistence on the full-time, key user involvement is that it's the only thing that works in the long run if you want good systems. As we have said for years, good systems require good users.

Secondly, when agile development calls for pair programming, it means two programmers working together all the time. This is also a requirement that many organizations have trouble with. Many software managers and many (if not most) programmers see themselves as hardy individualists. Indeed, many of these people don't like anyone even suggesting that they look at their code (even after its done, in some cases).

This has to be overcome. Programmers must be briefed and must understand the nature of pair programming, coding standards, and the fact that they are to do test-first design and deliver working versions every couple of weeks. A second issue involved with pair programming is the measurement of individual performance — you can't do it. If you (or your management) insist on measuring individual performance, don't start down the road to agile development because you'll only end up frustrated. Agile development is based on team performance; you need to understand that.

Agile development is much more of a stretch than CMM is for most organizations. Although CMM may be seen as overkill in many organizations, at least most people in the band will recognize the music and the instruments. With agile development, we're talking about something quite different, something more along the lines of new music, new instruments, and no conductor — a jazz ensemble as opposed to a concert band.

As a result, training is a key element to the success of agile development. Successful organizations need to bring in people with significant experience doing agile development and use them to train their people bottom up.

One of the other keys to success with agile development is a hands-off project management style. Project and software managers must learn that agile development is managed mostly by the project team (including the users). Agile development is consciously a minimalist management strategy. If you want high-quality software delivered quickly, application developers argue, then you should manage the key pressure points (i.e., the product releases).

Finally, organizations need to study agile development closely before they start. In fact, the key managers, users, and technical leaders should go to visit reference sites and talk seriously with their counterparts to find out what kind of problems they encountered.

Agile development is normally promoted bottom up in most organizations. It is usually brought in by some of the best and brightest young programmers, often the same people who have been the most serious supporters of OO development and new languages. For this reason, serious attention must be paid to getting some of the older, less up-to-date people educated. It is probably a good

[16] As you can see in our earlier discussion, agile development is clearly not an undisciplined programming approach; however, it's how many people from a more traditional standpoint see the movement.

idea to have a few of these people introduced to the first or second agile development project both for support and as a way to gauge what kinds of problems you're liable to have when you try to roll out the approach to other projects.

THE CURRENT BATTLEGROUND

As with any war, there is a lot of propaganda. And anywhere there is propaganda, there are slogans. For the agile development folks, the slogan is probably, "We're agile, and you're slow, bloated, and old-fashioned." If you believe the application developers, agile development is a minimal, fast-track approach, and everything else, especially CMM, is heavy-handed, bureaucratic, and doomed to failure.

On the other hand, CMM proponents point to the fact that CMM is used by some of the largest software organizations in the world on some of the largest projects. Their slogan would probably be, "Agile development is for small (inconsequential) projects; if you want to do some important (big, mission-critical) ones, you need CMM!"

As is always the case, there is some truth to both these points of view. CMM is bureaucratic and somewhat prone to producing huge amounts of paper. It is very expensive, but then again, so is failure. Small boats are, as a rule, faster and more maneuverable than big boats; on the other hand, you can't transport millions of gallons of oil or thousands of passengers on a small boat. The approaches that you use on small projects often don't scale up.

The agile development folks respond that their management movement is just as much a development strategy. If you're careful, they maintain, with how you plan your increments, you can do big things with a lot fewer people in a lot less time. They argue that you can create much larger projects (and organizations) than you could have in the past. The key is collaboration and commitment.

So who are you to believe? Well, as with most things, the answer lies somewhere in the middle. The war between CMM and agile development provides the opportunity to make both approaches better.

"We're Agile, Too!" or "Teaching the Elephant (CMM) to Dance"

How do you take an approach like CMM that is heavily about management control and measurement and make it light on its feet? This is the question that CMM organizations everywhere are confronting as they face more and more advocates of agile development.

Reengineering CMM

CMM has progressed along the statistical control path perhaps as far as it can go. Now it must evolve, and to evolve, it may need to go through a process of totally rethinking its mission.

For one thing, CMM must become less document-centered. Most of the systems that we are building today for our customers are built using electronic workflow approaches. Our customers are after systems that minimize data entry, forms, meaningless approvals, and paper. They are after us to build systems that are rapidly adaptable to changes in business practices and technology. CMM must be looked at in the same vein. One of the lessons that CMM should be learning from agile development is minimalism. What's the minimum we can do to produce a quality product?

Another thing that CMM must do is to emphasize collaboration. Software development is all about knowledge management, and knowledge ultimately rests in the heads of people. CMM needs to become more people-centric as opposed to organization-centric or management-centric. Increasingly, people in other disciplines have been coming up with better, more exciting environments in which to work. The hardware people, for example, have been working on concurrent engineering approaches for a long time now, and computer-aided design and manufacturing are state-of-the-art. CMM should be looking to provide a more automated

management umbrella that makes agile software development the norm.

Finally, CMM must stress speed as well as quality. The world is simply moving too fast to wait for quality software to finally spill over the last waterfall.

"We Do Scale Up!" or "Getting Agile Development to Work on Large, Outsourced Projects"

Now we're looking at the problem from the other side. How do you take a methodology, like agile development, that basically evolved in small groups and then scale it up so that it works on projects with hundreds of developers and thousands of users? How do you take a methodology that was developed for highly skilled OO programmers and use it in low-skill organizations? These are some of the questions that proponents of agile development must face today.

Reengineering Agile Development

What changes need to be made to agile development today? Well, for one thing, more attention must be paid to some basic management issues: architecture, design, documentation, integration, and tools.

Agile development depends on breaking projects into time-boxed iterations. Application developers have to come up with methods for breaking up larger and larger projects into small agile development ones that can be somehow brought together. The only way that I know to do this is by spending some time at the beginning building a systems architecture that can then be used for planning a series of small incremental projects that can be brought back together to form a larger whole. With something as dynamic as agile development, this process can be more complicated than in a CMM environment. In Figure 3, from Highsmith's most recent book, there is an iterative agile development systems lifecycle that has a place in the adaptive cycle planning phase for this kind of systems architecture activity [3].

Agile development simply must produce more design artifacts. The idea that the code ought to be the only documentation you'll ever need doesn't cut it. That's a little like saying that the chip contains all of the design concepts that you'll ever need to understand, let's say, a Pentium X computer. In its rush to minimize, agile development has thrown too much out with the bath water.

Finally, agile development has to get over its aversion to relying on tools. The software business has made itself one of the key drivers in modern business by providing electronic tools. If one could bring forward the best businesses from the 1960s into the present, they simply couldn't compete. Today's business world is an electronic one that operates in real time. Software development has to keep up. No matter how good our management strategy for software development, we can't get

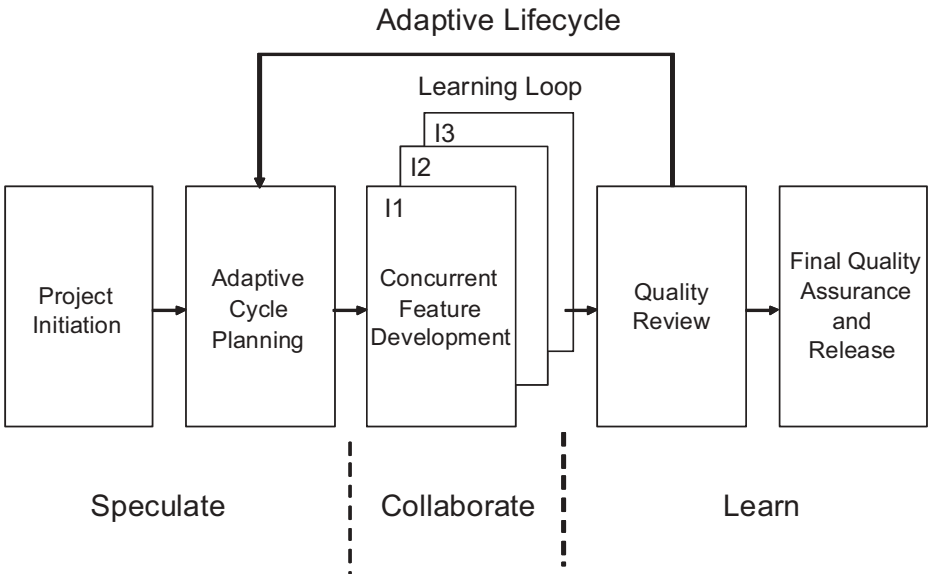

Figure 3 — The agile development systems lifecycle [3].

to where we have to without better and better tools.

"Horses for Courses" or "What Works Where and What Doesn't?"

Even though everyone gives lip service to the idea that some approaches work better in some environments or for some domains than others, no one wants to admit that their approach wouldn't work for everything, if people would just do it "right!"

One of the lessons that I've learned over a long career in software is that a five-man project is a lot different than a 50-man project, which is a lot different from a 500-man project. If you're charged with developing a 1,000 man–year project, you ought to look at CMM. If you're involved in developing a high-risk, high-change product, you ought to look at agile development.

If you look at where CMM and agile development have their biggest following (foothold), you can pretty much tell what they are best suited for. CMM is used mainly for large, outsourced software projects, in military software, and in areas where software reliability and safety are paramount (e.g., drug systems, medical devices, nuclear devices). Agile development is used mostly for product software, especially in areas using new technologies.

Highsmith makes the point that agile development evolved to help people with what he calls "exploratory projects." Exploratory projects are those that have the following characteristics:

- **Frontier technology** — pushing the state-of-the-art
- **Mission-critical** — critical to the business' overall success
- **Time-critical** — have a required time window to become operational
- **Constantly changing** — have very volatile requirements

There is no reason that a large organization can't have more than one standard method for software development; one approach for smaller, critical, technology projects; and another for very large, non-time-critical projects. Over time, CMM and agile development will begin to cross-fertilize, but that is a long-term activity.[17]

WHY CAN'T WE ALL JUST GET ALONG?

I learned a long time ago that despite all of the publicity in the press, software wars are side issues. If you are the ones behind CMM, the real competition is not agile development, but apathy. Many, if not most, organizations are stuck at Level 1. While those involved in the 21st century software wars are at war with each other, they are ignoring the great mass of people for whom almost any software development approach would be preferable to what they have.

The CMM and agile development folks need to talk more, and they need to take each other more seriously. The CMM side needs to see agile development as a new paradigm, not just another form of hacking. Unfortunately, many of the principal developers of CMM haven't been actively involved in development for a long time. They simply don't know what is really going on or how the software development and tools have changed.

On the other hand, the agile development side needs to see CMM not just as a failed attempt to overcontrol the creative process but as a way to help organizations do a better, more predictable job on very large, outsourced projects.

At the core, CMM and agile development both want to do the same things, which is to help people do a better job. They simply come at the problem with very different cultural bias.

In the process of developing this report, I took the opportunity to read the various articles in "The Great Methodology Debate" from *Cutter IT Journal*. What struck me the most was that almost all of the authors in that debate misrepresented the other side. Of course, in a debate, one is

[17] The CMM folks maintain that CMM and agile development are compatible. Barry Boehm in a recent dialogue with Council Fellow Tom DeMarco maintained that some CMM Level 5 organizations are working to integrate the two approaches.

expected to make the strongest case possible, but debates normally don't make good science or good technology.

It seems to me that we can either use the current software wars as an excuse to keep going in the direction we've already decided on, or we can see it as an opportunity to advance the state of the art. Our industry needs to clarify what it means and the terms it uses. In fact, we need to be taking to heart some of the ideas that business researchers like Clayton Christensen have been saying about disruptive technologies and plot a course that aims not at getting our organizations to the current best practice but to some higher level, something I've labeled "next practice."

NEXT PRACTICE: EFFORT VERSUS KNOWLEDGE

In today's management jargon, best practice is everything. Best practice represents a safe harbor in a stormy sea of too much hype and too many problems. Best practice means that someone big and successful has used something that has been blessed by someone else. Best practice means that I don't have to defend my decision against outside consultants or inside critics.

Best practice is, therefore, the Good Housekeeping Seal of Approval. Unfortunately, because it is such a great term, best practice is overused and, in most contexts, has lost much of its original meaning. Moreover, best practice is hard to quantify. Which is really software best practice, CMM or agile development? Both sides have data and proof to support their definition of software development as a best practice.

DISRUPTIVE TECHNOLOGIES: WHEN BEST PRACTICE IS NOT GOOD ENOUGH

What's worse is that in the 21st century, best practice may not be good enough or fast enough. I am a big fan of Clayton Christensen and his idea of disruptive technologies. Christensen became interested in this issue when he was doing research for his Ph.D. at Harvard Business School. He was intrigued by the fact that he saw numerous business examples of established, very well-run companies lose out to startup companies in areas where older companies had once dominated.

In particular, Christensen was fascinated by the fact that none of the companies that were big in the 1980s manufacturing 5.25-inch disk drives were able to segue their dominance into the even bigger market for 3.5-inch drives. And as he studied other marketplaces, Christensen found similar cases in other areas such as discount retail (Wal-Mart versus Sears), software (Microsoft versus IBM) and steel (Nucor versus US Steel).

What Christensen concluded was that these companies were not poorly managed; for the most part, they were very well managed — for a given environment. In each of these markets, the dominant firms did what was expected of them: they worked to keep growing and to maintain the profit margins, and they listened closely to their best (largest) customers. In a word, they represented the best practice for their time and for their industry. And, ironically, being the best practice led to their downfall.

As I interpret Christensen, these dominant, best practice companies became trapped, naturally enough, by their own success. As they grew in size, they had to take over more and more of their market to maintain the growth rate. And they had to do this while maintaining their profitability. This made them natural targets for disruptive technology startups that, although not ready for prime time, found niche (low-cost, specialty) markets where they could grow rapidly, learn how to make money with lower margins, and then expand into the major market. Christensen found this process to be repeated over and over again in market after market.

Startup organizations that embraced disruptive technologies learned to prosper with less overhead and less cost. In time, when they overtook the bigger, slower rivals, they became the best

practice (and ultimately subject to some other disruptive technology).

So what are the conclusions that can be drawn from all of this with respect to best practice? Well, if an organization — no matter how well-managed that organization — is not careful, it can be overtaken and replaced by another practice that it can't understand or integrate into its operation. I call those practices based on disruptive technologies "next practice."

Increasingly, organizations in the 21st century will need to be focusing on what the next practices in their industry, market, or specialty will be. In the software world, I believe that next practice has to do with knowledge versus effort. Let me illustrate this by pointing to Figure 4, which has two dimensions. The horizontal dimension is effort or, if effort is too hard to track, amount of documentation. The vertical dimension is value or knowledge.

It seems to me that this figure represents the future of software better than the CMM levels. Here we have four quadrants instead of five levels.

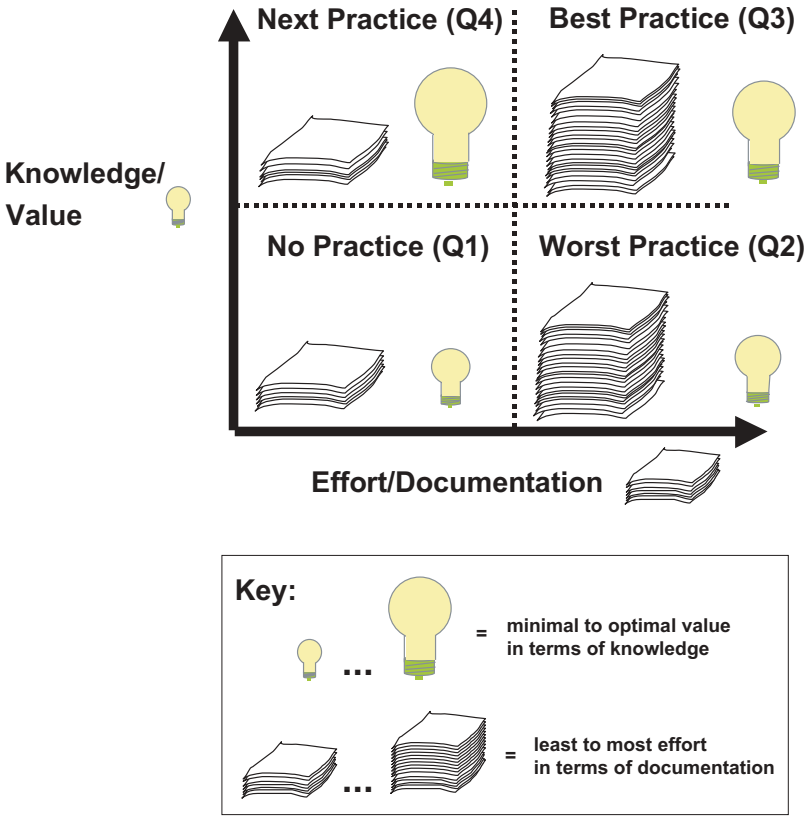

Figure 4 — Next practice quadrants (knowledge versus effort).

Quadrant One: No Practice

Quadrant One (Q1), in our worldview, contains those organizations just getting by. By and large, this quadrant corresponds roughly to CMM Level 1 organizations. They don't expend very much effort in their software activities, and they don't get much return in terms of knowledge.

Quadrant Two: Worst Practice

Organizations that fall into Quadrant Two (Q2) represent those that have the current worst practice — they expend a lot of effort, as is represented by the amount of effort or documentation that they produce, but they seem to be just going through the motions. Q2 organizations are not getting much from their effort.

Quadrant Three: Best Practice

Quadrant Three (Q3) organizations put in a lot of work, but they are more adept (efficient) than the organizations in Q1, and especially Q2, in getting knowledge out of this activity. Organizations in Q3 are often labeled current best practice.

Quadrant Four: Next Practice

Quadrant Four (Q4) organizations have the best of both worlds — they get a lot of knowledge (value) with a lot less effort than Q3 organizations. This quadrant is labeled next practice because organizations in Q4 are often at the leading edge. They are doing the best work, but they haven't

been doing it long enough to have universal recognition. Indeed, Q4 organizations are often practicing what Christensen calls disruptive technology in a marketplace (domain), which is too small or too specialized to be coveted or understood by the broader (dominant) market players.

APPLYING THE NEXT PRACTICE QUADRANTS TO UNDERSTANDING CMM AND AGILE DEVELOPMENT

The next practice quadrants provide a way to compare the discussion of CMM with the agile development debate. Using diagrams, I have tried to place the CMM levels along with agile development to see what each is trying to say and do. This is by no means a scientific analysis, but it does point out some interesting issues, as we will explore in this report.

If you apply the next practice quadrants to CMM, you get some very interesting results (see Figure 5). CMM Level 1 organizations fit naturally in Q1 (no practice). Next, is the most interesting placement. I would put most CMM Level 2 organizations in Q2 (worst practice). Although this may confuse some people, I think that it is, in fact, a fair description of what I think really happens; moving from CMM Level 1 to CMM Level 2 doesn't necessarily improve short-term productivity.

Consider the following scenario: The first step in moving up the CMM levels actually makes things appear worse. Designers and programmers must do a lot more documentation and fill out a lot more forms, but, initially at least, they don't get a lot more out of it in terms of knowledge (output).

This may not fit the CMM promotional literature, but it actually fits my own experiences working with organizations trying to install new processes. After the first blush of excitement, things often get difficult as people try to do new things that they are not good at; there is a lot of additional work that doesn't seem to produce much output. As a result, the people in the trenches often lose interest and are tempted to go back to their old non-process.

This represents one of the real difficulties in following a program like CMM in the real world; before things get better, they tend to get worse.[18] This is not an unknown phenomenon in other areas. The same thing often happens, for example, with individuals who embark on a program of diet, exercise, or any number of other things that are truly good for you. To go from where you are to where you want to be, you often must endure some considerable pain and frustration; that's why it's so hard to change.

So it is with CMM. To get from Level 1 to Level 3, you have to

[18]To quote Council Fellow Tom DeMarco, "The truth will make you free, but first it will make you miserable!"

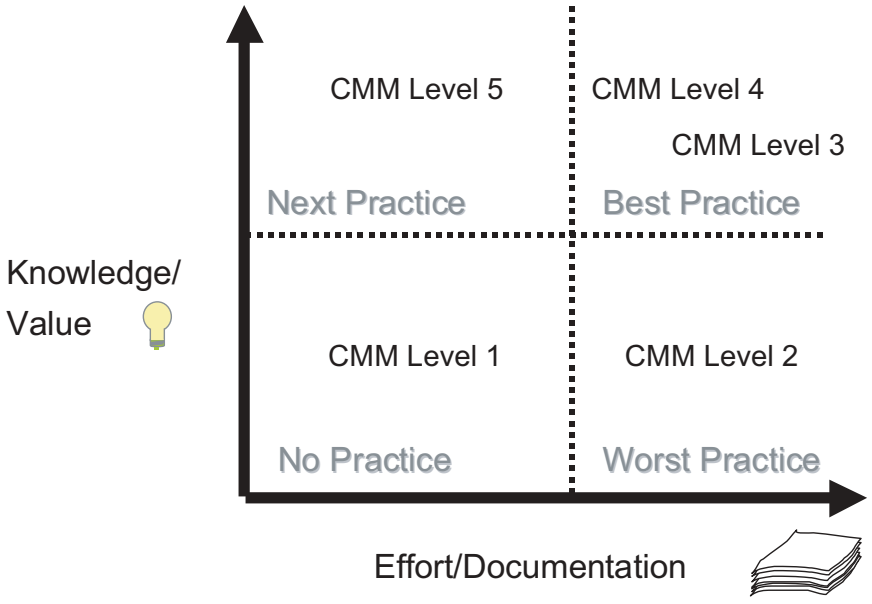

Figure 5 — CMM levels and next practice quadrants.

go through Level 2, which many people will find more difficult and less rewarding than what they are currently doing. It takes a lot of determination on the part of management to stick with the CMM program long enough to make real, recognizable improvements.

Once you break through to Level 3, things get better. Levels 3 and 4, then, represent what most people would label as best practice in the industry today. Again, what is most interesting is that Level 3 organizations work harder and get less out of their efforts than Level 4 organizations because Level 4 organizations are more skilled and experienced — they know what to expect, and they know how to work around problems. Finally, the best of the best make it to Level 5 and break into Q4. These organizations are constantly revisiting their processes and making them more and more efficient and effective. They work less and attain more knowledge.

Unfortunately, there are not many Level 4 and 5 organizations in the world, and most of them are outside the US. As you can see by Figure 6, the vast majority of CMM Level 4 and 5 organizations are classified as offshore. The number is not particularly large by any standards, but US Level 4 and 5 organizations represent 7% of all the organizations seeking CMM certification, which is only a small percentage of the firms nationwide.

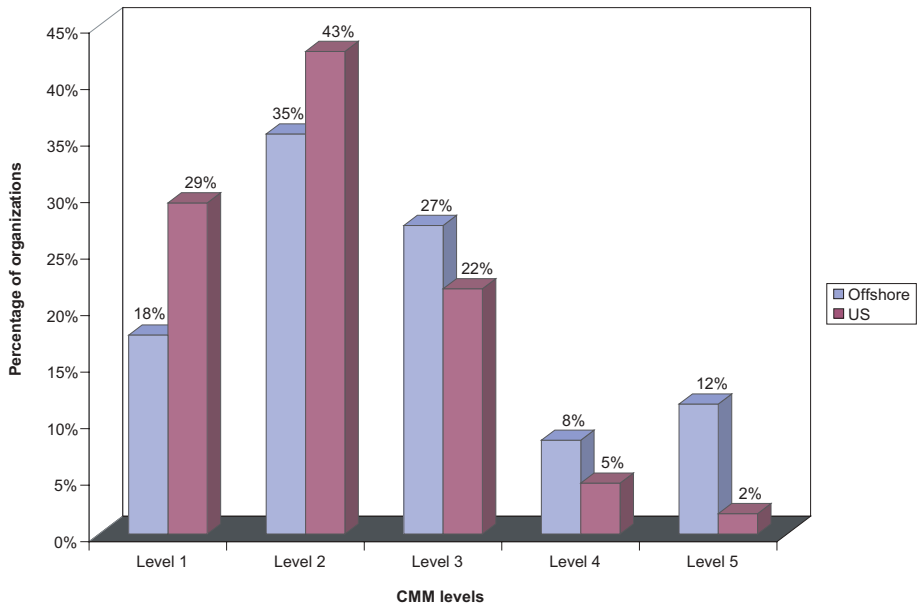

Figure 6 — CMM-certified organizations by level and location. (Source: Software Engineering Institute, 2001.)

APPLYING NEXT PRACTICE QUADRANTS TO AGILE DEVELOPMENT

Where does agile development fit on this next practice diagram? Well, since it is relatively new, there are not a lot of statistics for agile development projects. Based on the reading that I have done, I would place agile development in Q4 (next practice). From everything that I understand about agile development, it attempts, and to a high degree succeeds, in gaining a lot of knowledge — especially about the functionality and behavior of a new program or system — with less effort than you would ordinarily expend. My guess is that a typical agile development project would not provide as much knowledge as might be gained by a CMM Level 5 organization, but it represents a different way to get the information.

It is useful to look at the next practice quadrants as a way to compare CMM and agile development as it points out some interesting things to think about. By focusing on knowledge versus effort, it provides an analysis framework that can be used to think about alternative ways of doing things (see Figure 7). Various questions can then be posed (e.g., "What do we learn from this?"; "Is the knowledge worth the effort?").

Though agile development may not be mature yet, it has come on strong in recent years, and

any serious software development organization ought to look at it. The agile development people have taken a very Alexandrian approach to solving the software version of the Gordian knot. Instead of trying to unravel the traditional software ball of twine, the agile development gurus have simply sliced it in two. Everything that they feel doesn't contribute to direct user communication or the end product is out.

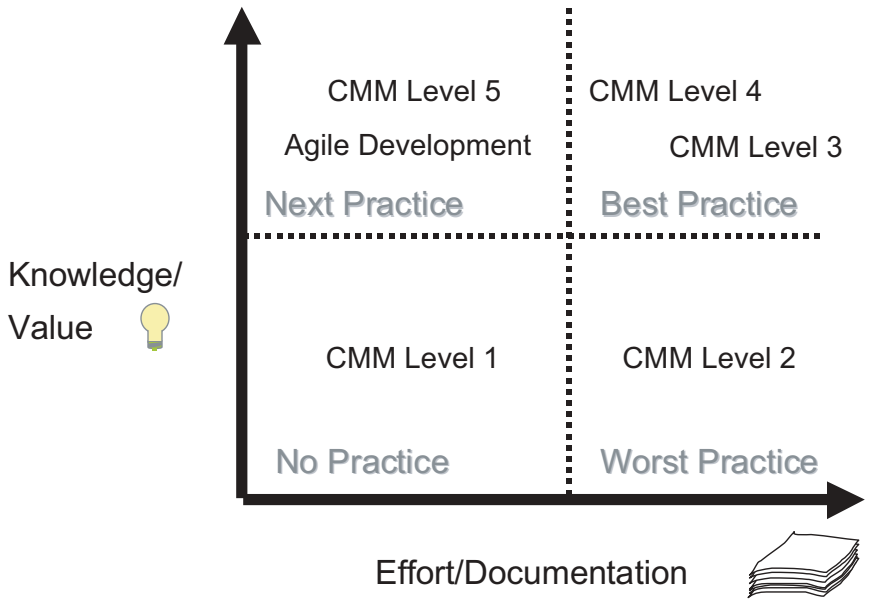

Figure 7 — Agile development and the next practice quadrants.

OTHER VOICES

> At 40, I resolved not to work on a project that didn't leave behind a tool. Social change simply takes too long.
>
> — Buckminster Fuller

From the first time I read the *Agile Manifesto*, I felt that the agile development folks had taken a bad turn. Their very first proclamation is that they value "individuals and interactions over processes and tools." Right there, I felt that the agile developers had boxed themselves in every bit as much as the CMM folks did when they focused so much on "the what" over "the how" of software development.

Software development, like so many things, is an integrated activity; you can't take just one piece of it in isolation. Those of us who spend our lives trying to get others to take advantage of technology are at the same time some of the most backward users of technology. Coding is not an art form any more than circuit board layout is; it is simply a means to an end.

Whether it is Java or COBOL, programming is still programming. As long as we do it principally by hand, we won't be able to make anywhere near the strides that our counterparts in hardware design have made over the years. Imagine if there were a Moore's Law for software. We would go generations beyond Windows XP, generations beyond Oracle 9i, and generations beyond SAP and PeopleSoft.

Software continues to be dragged into ancient discussions about programming style and programming collaboration. If we spent half the effort building and using better tools to support better processes, we wouldn't have so many Dilbert-esque discussions at technical meetings around the world.

Lots of people involved in the current software wars have failed to notice that while debating the right way to do OO design or refactoring, a large part of the world stopped doing development at all. Today, more and more of my clients are in the business of finding, modifying, and installing clumsy, bloated software packages because they got tired of cost and schedule overruns and functional underruns for major software projects.

I strongly believe that agile development will only have a really major effect on the software industry when it stops drawing the line in the sand that says, "Look — we're people people, and we're not into tools or processes."

There is no other area of science or business that takes this point of view. Indeed, all of the advanced sciences and businesses invest more and more of their resources in leveraging technology to help them do more in less time.

I have been developing a requirements and prototyping methodology called agile requirements for some time now. It is built around the integration and use of state-of-the-art tools to help analysts, designers, users, and developers understand their problems in business terms and then, as automatically as possible, design databases, generate programs, and prototype those business requirements and processes. I consider this approach to be an advanced form of agile development, and I see it fitting on the next practice quadrant diagram high in Q4 (see Figure 8).

All advanced approaches today are a synergy among people, technology, and communication. Choosing one of these elements over the other tends, like only pulling on one oar, to drive the software boat in circles. We need everyone pulling as fast as we can if we are to take advantage of what people are good at and what technology and communication are good at.

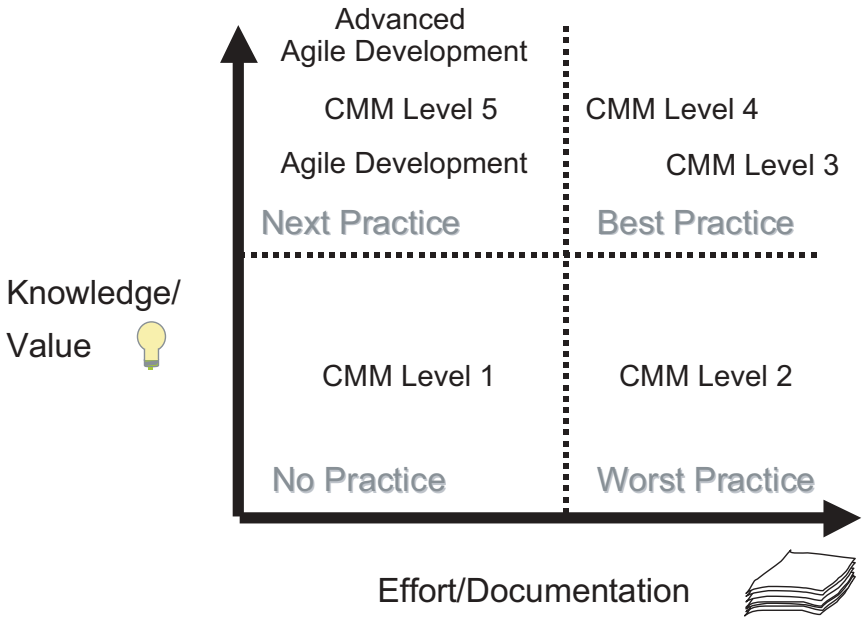

Figure 8 — Advanced agile development.

CONCLUSION — ON WORDS AND PROGRESS

In my years in software research, I have found that most knockdown, drag-out fights are almost always about homonyms. The really heated discussions occur when two people (or groups) mean very different things by some word or phrase. For example, "objects" (or "object-oriented") means one thing to people trained in and committed to OO and often something very different to those brought up in another form of programming.

The same thing is true about the term data warehousing or requirements engineering. In practice, the same term may mean a number of things to a number of different people all engaged in the same conversation, which creates, as you might imagine, huge opportunities for both confusion and misunderstanding. If you're lucky and no one has become rich by exploiting this confusion, you can finally sit down at the table and discover that, in most areas, you actually agree with the people you've been fighting with.

But misunderstanding what the other guy really means makes communication difficult, to say the least. As I learned from long years teaching in this field, it doesn't matter what you say; what really matters is what the other guy hears. I remember listening to a public radio interview one

day where the interviewee was one of the most renowned language teachers in Hollywood. The interviewer asked him who made him so successful. I think the answer should be posted on every training room in the world: "Well, I always figured if someone did get what I was trying to teach them, it was my fault." We all need to take that attitude when trying to communicate with others, especially others with whom we are in strong disagreement.

As a rule, technology folks are not good communicators. And, over a period of time, especially when what you're communicating is complex (and it usually is) or people reject what you're telling them (which they often do), your critical words and phrases begin to take on emotive characteristics.

At a certain point in a long-term debate, for example, the word object comes to mean good. So when certain people say OO programming, they are really saying good programming. This makes it difficult, if not impossible to talk about alternatives, since, of course, non-OO programming means bad programming, and who in their right mind would want to talk about bad programming?

In a similar fashion, the word agile currently has lots of meanings; some of them are in common use in our latest software war. But the emotive use of agile as in agile development is of recent vintage and may or may not mean what agile means in common discourses. To me, agile conjures up visions of acrobats and gymnasts or organizations that can quickly change. It is not clear that this is what those who talk about agile development really mean.

Or what about quality? Quality is another one of those key phrases. The word quality occurs frequently in CMM material. Who wouldn't want to develop quality software? Clearly, quality means good — nobody would want to develop non-quality software. But does quality software mean that we have to have tons of paper documentation?

Everyone, or nearly everyone, in the software business wants to do better; it's just that different people have different visions of what better means. Some people think that better means more control and less variability. Others think better means quicker, with less paperwork.

What should we be doing? Most CIOs and lots of systems development managers would like to do better but they are deathly afraid of fragging.[19] They are afraid that if they institute a change that either their programmers or users don't like, they're toast. Programmers are a tough group, and in many organizations, they have keys to the company jewels.

[19]Fragging is when an officer is shot, usually in the back, by his own troops.

Personally, I am skeptical about convincing many CMM Level 3 organizations to become agile. I am also skeptical about trying to sell the value of business modeling and design documentation to a shop that has bought into agile development.

But everyone can learn. If we are a large CMM Level 3 or 5 shop, we should be constantly questioning whether we need every bit of paper that we currently require. What if we substitute prototyping for user interface screens on paper? Wouldn't that be better?

What about agile organizations? Maybe they could begin to look at some of the metrics that CMM shops keep and see whether they wouldn't be useful even in an agile environment.

My personal feeling is that the most important thing that we can do is to keep an open mind. Software is not religion. We don't need priests; we need artists, architects, and engineers. We need people who can build quality (here's that word again) software again and again. We need to be able to promise something to someone and mean it.

I think that agile development is a truly disruptive technology. I think that most organizations can learn something about doing a better job at software development by studying agile development and that most organizations can also learn

something from studying CMM. And, I think that much better ways of developing software are on the way.

SOME COMMENTS ON MY OWN FORM OF AGILE DEVELOPMENT

In writing this article, I have come to the somewhat disturbing realization that, in my own organization, we have evolved a form of agile development. For one thing, we don't have much in the way of formal documentation. As the sole user/designer, I basically explain what I want on a white board and my partner Randy develops a prototype of what I have asked for. After I play with the prototype for a while, we revise the solution and come up with another version.

In general, this works well for us. We're small, we know what we want to accomplish (we've been at it for a very long time), and we're working in a very complex systems environment — trying to interface between the internal metadata of two very sophisticated development tools.

A good deal of the success of this current venture is due to the experience and the style of the user and programmer here. Randy is a marvelous programmer who knows (or can pick up) just about any programming language or database management system. He is also fearless and self-taught. Once he gets his teeth into a problem, I'm confident that he will figure it out.

Then there's the chief user: me. I've been in the software development business since the 1960s, and I've been working to develop more powerful tools for modeling business problems and automatic design for the past 25 years. Although I sometimes don't know exactly what I want the final product to look like, I'm sure I'll recognize it when I see it. Finally, the problem is not huge. The key to the tools that we're trying to develop is integration and ease of use. That makes us an ideal candidate for a form of agile-like development.

But there are a number of pieces of agile development that we don't employ, such as test-first development and pair programming. Moreover, we're still going to run into the problem of documentation, which we haven't done as we've been going along.

Now I also feel the need to comment on a previous attempt to solve the same problem. Back in the 1980s, a larger organization (which I would characterize now as CMM Level 3 or 4) and I attempted to do what Randy and I are doing now with just two people. We used a strict methodology, imposed on us both by our customer and our own marketing, and we failed to produce very much (though we did learn a lot about how not to do what we were trying to do).

Even then, I was pretty sure that the approach would not yield success. Part of our difficulties had to do with the problem domain. No one had ever done what we were attempting to do (they still haven't), and the base technology wasn't there. In the mid-1980s, there was very little in the GUI tools or standards, no one knew which way the industry was going, and many of the database tools and code generators had yet to be invented.

What I'm saying — from my personal experience — is that heavy methods don't work as well when there is lots of uncertainty about the technology or the problem. Agile methods work much better when the user and the developer make it up as they go along. On the other hand, there are large classes of problems that are simply too big to be done by such small teams. And coordinating lots of tiny teams of two or three is just as big a problem as coordinating teams of 10 or 20. Scaling agile development is likely to make it look more like CMM than many agile developers want to admit.

In the end, there is no fundamental conflict between CMM and agile development. The objective is producing better software. There is no one way to do this. CMM and agile development both have a lot to offer. CMM plus agile development plus better tools is likely to produce a hybrid that will make for better software for everybody.

ABOUT THE AUTHOR

Ken Orr is a Fellow of the Cutter Business Technology Council and a Cutter Consortium Senior Consultant and contributor to Cutter's Business-IT Strategies Practice. He is a regular speaker at Cutter *Summits* and symposia. Mr. Orr is a Principal Researcher with the Ken Orr Institute, a business technology research organization. Previously, he was an Affiliate Professor and Director of the Center for the Innovative Application of Technology with the School of Technology and Information Management at Washington University. He is an internationally recognized expert on technology transfer, software engineering, information architecture, and data warehousing. Mr. Orr has more than 30 years' experience in analysis, design, project management, technology planning, and management consulting. He is the author of *Structured Systems Development*, *Structured Requirements Definition*, and *The One Minute Methodology*. He can be reached at korr@cutter.com.

REFERENCES

1. Beck, Kent. *Extreme Programming Explained*. Addison-Wesley, 2000.

2. Highsmith, Jim, and Ken Orr. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House, 2000.

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

4. Humphrey, Watts S. *Managing the Software Process*. Addison-Wesley, 1989.

5. Humphrey, Watts S. "Three Dimensions of Process Improvement (Part I: Process Maturity)." *Crosstalk*, February 1998.

6. Raynus, Joseph. *Software Process Improvement with CMM*. Artech House, 1998.

ADDITIONAL RESOURCES

Humphrey, Watts S., and W. Sweet. *A Method for Assessing the Software Engineering Capability of Contractors*. SEI Research Report, 1987.

Jeffries, Ron. *What Is Extreme Programming?* 2001 (www.xprogramming.com).

Newkirk, James W., and Robert C. Martin. *Extreme Programming in Practice*. Addison-Wesley, 2001.

About the Practice

Agile Project Management Practice

Cutter Consortium's Agile Project Management Practice helps companies succeed under the pressures of this highly turbulent economy. The practice is unique in that its Senior Consultants — who write the reports and analyses for the information service component of this practice and do the consulting and mentoring — are the people who've developed the groundbreaking practices of the Agile Methodology movement. The Agile Project Management Practice also considers the more traditional processes and methodologies to help companies decide what will work best for specific projects or teams.

Through the subscription-based publications and the consulting, mentoring, and training the Agile Project Management Practice offers, clients get insight into agile methodologies, including Adaptive Software Development, Extreme Programming, Dynamic Systems Development Method, and Lean Development; the peopleware issues of managing high-profile projects; advice on how to elicit adequate requirements and managing changing requirements; productivity benchmarking; the conflict that inevitably arises within high-visibility initiatives; issues associated with globally-disbursed software teams; and more.

Products and Services Available from the Agile Project Management Practice

- The Agile Project Management Advisory Service
- Consulting
- Inhouse Workshops
- Mentoring
- Research Reports

Other Cutter Consortium Practices

Cutter Consortium aligns its products and services into the nine practice areas below. Each of these practices includes a subscription-based periodical service, plus consulting and training services.

- Agile Project Management
- Business Intelligence
- Business-IT Strategies
- Business Technology Trends and Impacts
- Distributed Enterprise Architecture
- IT Management
- Measurement and Benchmarking Strategies
- Risk Management and Security
- Sourcing

Senior Consultant Team

The Cutter Consortium Agile Project Management Senior Consultant Team includes many of the trailblazers in the project management/peopleware field, from those who've written the textbooks that continue to crystallize the issues of hiring, retaining, and motivating software professionals, to those who've developed today's hottest agile methodologies. You'll get sound advice and cutting-edge tips, as well as case studies and data analysis from best-in-class experts. This brain trust includes:

- Jim Highsmith, Director
- Scott W. Ambler
- Sam Bayer
- Kent Beck
- E.M. Bennatan
- Tom Bragg
- Robert N. Charette
- Alistair Cockburn
- Doug DeCarlo
- Tom DeMarco
- Khaled El Emam
- Ian Hayes
- Ron Jeffries
- Joshua Kerievsky
- Brian Lawrence
- Tim Lister
- Michael C. Mah
- Lynne Nix
- Ken Orr
- Chris Pickering
- Roger Pressman
- Ram Reddy
- James Robertson
- Suzanne Robertson
- Alexandre Rodrigues
- Johanna Rothman
- Lou Russell
- Rob Thomsett
- Colin Tully
- Richard Zultner