

Are CMM and Agile Methods Really Compatible?

Pietari Laurila

Abstract—A recent stream of literature has suggested that CMM and Extreme Programming, a popular agile method, are compatible. In this paper, we suggest that there are in fact profound differences between CMM and agile methods. Through four perspectives, we show that CMM and agile methods are practice compatible but not idea compatible. CMM endorses scientific management; agile methods reject it. We conclude that CMM and agile methods are compatible if either 1) agile methods are used devoid of their value content, or 2) CMM is used devoid of its value content.

Index Terms—agile methods, CMM, compatibility, values

1. INTRODUCTION

THE software industry has throughout its history battled with overrun budgets and schedules, volatile cost and time estimates, final products crippled with bugs, and increasing complexity (Martinsson, 2002). The battle has been long and brutal. In the 1960s, NATO convened a conference to discuss what had come to be called the *software crisis*. The crisis was not resolved then, and the four ensuing decades have seen little improvement. A recent study showed, for example, that 80 percent of systems are delivered late and over budget, and about 40 percent of development projects fail entirely (Clegg et al., 1996). The problem needs a remedy, and one remedy is the software process. In this paper we examine, compare, and contrast two classes of software processes: those based on the Capability Maturity Model and those based on agile methods.

The Capability Maturity Model (CMM, also known as the SW-CMM) is a set of guidelines for improving and measuring the capability of software organizations (Paulk et al., 1993a). The CMM is not a software process, but a model for process improvement. CMM divides process improvement into five stages. An organization starts at level 1 and over several years progresses to level 5. At level 1, the software process is chaotic. Few processes are defined, and success depends on individual effort and heroics. At level 5, by contrast, the software organization continuously optimizes its processes by piloting innovative ideas and technologies. Section 2.1 contains a more detailed introduction to CMM and its successor, CMMI (CMM Integration).

The Capability Maturity Model is firmly rooted in the engineering methodology tradition (Fowler, 2003). The distinguishing characteristic of engineering methodologies is that they emphasize planning. According to Fowler (2003),

however, planning failed. It did not resolve the software crisis. Worse, it was unpopular and tended towards bureaucracy.

As a reaction to these failures, a new group of methodologies has appeared in the last few years (Fowler, 2003). These are called *agile methodologies*. Agile methodologies try to find the right balance between too much process and too little of it. Section 2.2 discusses agile methods more thoroughly.

At first sight, agile methods seem to directly challenge the engineering focus of CMM. They are adaptive rather than predictive and people-oriented rather than process-oriented (Fowler, 2003). CMM, on the other hand, has traditionally been seen as a rigid, plan-driven method. But this might be a false impression. Indeed, a recent stream of literature (Jeffries, 2000; Paulk, 2001; Martinsson, 2002; Glazer, 2001) has suggested that CMM and Extreme Programming, a popular agile method, are in fact compatible. This stream is supported by another line of inquiry that examines the compatibility of plan-driven and agile methods (Paulk, 2002; Boehm, 2002; Boehm & Turner, 2003). On the other hand, there are also indications that combining CMM and Extreme Programming can be a frustrating experience in practice (Reifer, 2003).

The question this paper will address is therefore this: are CMM and agile methods really compatible? We will examine the question from four perspectives. First, we examine the various meanings of process discipline. Second, we study who is the driver of process improvement. Third, we ask what agile methods and CMM measure and how. Finally, we scrutinize the values behind CMM and agile methods.

The result of this inquiry will be a picture of the true compatibility of CMM and agile methods. This picture can help practicing managers when they evaluate CMM or agile methods for use in their organizations. It can also clarify where the true differences between CMM and agile methodologies lie, so that these differences can be mitigated and eliminated.

The paper is a literature study. On the agile methodologies side, we focus on Extreme Programming (XP) because it seems to be the most popular agile method and also because others have already compared it to CMM. CMM will be the only relevant methodology on what has traditionally been seen as the plan-driven side.

The paper proceeds as follows. Section 2 describes the CMM and agile methods. Section 3 reviews existing research on the compatibility of CMM and agile methods. Section 4 analyzes the compatibility of CMM and agile methods from

the four perspectives mentioned above. The results are discussed in Section 5. Section 6 analyzes the implications of the results for practitioners. Section 7 then concludes.

2. OVERVIEW OF CMM AND AGILE METHODS

2.1 CMM

The Capability Maturity Model is a set of guidelines for improving and measuring the capability of software organizations (Paulk et al., 1993a). The first predecessor of the model was published by the Software Engineering Institute at Carnegie Mellon University in 1987 (Martinsson, 2002). Version 1.0 of the model was released in 1991, followed in 1993 by version 1.1, which incorporated several improvements. Version 1.1 is still the latest version of the model.

The CMM divides process improvement into five stages, or levels (Paulk et al., 1993a). On each level there are a number of Key Process Areas (KPIAs) that should be addressed before an organization can move to the next level. There are in total 18 Key Process Areas. Each Key Process Area contains one or more goals. Altogether there are 52 goals. Table 1 shows the five levels and 18 Key Process Areas of the CMM.

The five levels of the CMM represent different levels of organizational maturity (Paulk et al., 1993a). At level 1, the software process is chaotic. Few processes are defined, and success depends on individual effort and heroics. At level 2, project management processes are established to track cost, schedule, and functionality. Past successes can be repeated. At level 3, the software process is documented and standardized for the entire organization. Projects tailor a suitable process for themselves from shared assets. At level 4, detailed,

quantitative measures of process and product quality are collected. Finally, at level 5, the software organization continuously optimizes its processes by piloting innovative ideas and technologies.

The 18 CMM Key Process Areas share five common features that implement and institutionalize processes (Paulk et al., 1993a). These five common features measure whether the implementation of a Key Process Area is effective, repeatable, and lasting. The common features are Commitment to Perform, Ability to Perform, Activities Performed, Measurement and Analysis, and Verifying Implementation (Paulk et al., 1993a). We next describe each.

Commitment to Perform. The first step in institutionalizing a process is to create a commitment to perform the process. This typically involves establishing organizational policies and senior management sponsorship.

Ability to Perform. The second step is to create an ability to perform the process. This typically involves resources, organizational structures, and training.

Activities Performed. The third step is to specify the activities to be performed. This involves establishing plans and procedures, performing the work, tracking it, and taking corrective actions as necessary.

Measurement and Analysis. The fourth step is to measure the process and analyze the measurements. This typically includes measurements that determine the status and effectiveness of Activities Performed.

Verifying Implementation. The final step is to ensure that the activities are performed in compliance with the process that has been established. This typically involves reviews and audits by management and software quality assurance.

A successfully institutionalized process becomes part of the organizational culture, or “the way we do things around here” (Paulk, 2001). Institutionalization is a very important feature of CMM and one we will discuss in more detail in the coming sections.

The Capability Maturity Model is currently being replaced by Capability Maturity Model Integration (SEI, 2002). The CMMI combines three previous CMM models into one: the software CMM (SW-CMM), the integrated product development CMM (IPD-CMM), and the software acquisition CMM (SA-CMM). In addition it is compatible with EAI Interim Standard 731, System Engineering Capability Model (SECM). The Carnegie Mellon Software Engineering Institute no longer intends to update the software CMM, and the last CMM training classes were held in 2003.

There are a few differences between the software CMM and CMMI (SEI, 2003). The CMMI contains new process areas, new modern best practices, and new generic goals. In addition, there is a new continuous representation to complement the old staged representation.

On the whole, the differences are minor (SEI, 2003). At level 2, there is one new Key Process Area, Measurement and Analysis, which was scattered throughout the CMM. At level 3, the Software Product Engineering Key Process Area is

TABLE I
AN OVERVIEW OF THE SOFTWARE CMM

Level	Focus	Key Process Areas
5: Optimizing	Continual process improvement	Defect prevention Technology change management Process change management
4: Managed	Product and process quality	Quantitative process management Software quality management
3: Defined	Engineering processes and organizational support	Organization process focus Organization process definition Training program Integrated software management Software product engineering Intergroup coordination Peer reviews
2: Repeatable	Project management processes	Requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management
1: Initial	Competent people (and heroics)	

replaced by five more detailed process areas. Risk management has been formalized into its own process area. Finally, there is a new Decision Analysis and Resolution process area. There are no substantial changes, only restructurings, at levels 4 and 5.

The CMMI's new generic goals should have little impact on organizations that currently use the software CMM (SEI, 2003). Further, SEI believes that there is no superior CMMI representation: the choice between staged and continuous models depends on the context of the organization, with the staged model probably being preferred by organizations that previously used the software CMM (SEI, 2002). All this suggests that what applies to the software CMM will apply to CMMI as well.

Despite the fact that the software CMM is being phased out, we chose to analyze it rather than CMMI. The reason is that all existing comparisons of CMM and Extreme Programming are based on the software CMM. But as noted, it should be possible to generalize the results to CMMI.

2.2 Agile methods

Agile methodologies appeared in the late 1990s as a reaction to the failures of plan-driven methods (Fowler, 2003). In 2001, the Agile Alliance distilled the ideas of agility into the Agile Manifesto (Beck et al., 2001). The Manifesto consists of four pairs of values:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation; and
- Responding to change over following a plan.

The Manifesto concludes: "That is, while there is value in the items on the right, we value the items on the left more."

Fowler (2003) identifies two contrasts between agile and plan-driven methods. The first is that agile methods are adaptive rather than predictive. Agile methods adapt to changing requirements and environments. They do not try to plan everything in advance. The second difference identified by Fowler is that agile methods are people-oriented rather than process-oriented. Plan-driven methods try to make the process work for whoever will be using it. Agile methods, on the other hand, contend that no process will ever replace the skill of the development team, so the role of the process is to support the development team in their work.

The number of agile methods has mushroomed in the past few years. The most popular agile method seems to be Extreme Programming (XP), which was first used at Chrysler to develop a payroll application (Beck, 2000). Other agile methods include Scrum, feature-driven development (FDD), adaptive software development (ASD), the Crystal family of methodologies, agile modeling, and DSDM (Abrahamsson et al., 2003). We cover XP in more detail because XP is the focus of existing comparisons of CMM and agile methods.

The underlying assumption behind XP is that the traditional high cost of change can be obviated by modern technologies (Beck, 2000). If so, it becomes possible to "embrace change." This idea is one of the key premises of agile methods. As

TABLE II
RECOMMENDED PRACTICES OF EXTREME PROGRAMMING

Practice	Description
The Planning Game	Quickly determine the scope of the next release by combining business priorities and technical estimates.
Small releases	Put a simple system into production quickly. Release new versions on a very short cycle.
Metaphor	Guide all development with a simple shared story of how the whole system works.
Simple design	The design of the system should be as simple as possible at all times.
Testing	Developers write unit tests for all functionality; customers write acceptance tests.
Refactoring	Programmers restructure the system without changing its behavior.
Pair programming	All production code is written by two programmers at one machine.
Collective ownership	Anyone can improve any system code anywhere at any time.
Continuous integration	Integrate and build the system many times a day.
40-hour week	Work no more than 40 hours a week.
On-site customer	Include a real customer on the team full-time.
Coding standards	Programmers write all code in accordance with the team's coding standards.

Beck (2000: 24) concedes: "If a flattened change curve makes XP possible, a steep change curve makes XP impossible."

XP demonstrates its agility by its values: communication, simplicity, rapid feedback, and courage (Beck, 2000). It proposes 12 practices, which are listed in Table 2. Even though the practices have been around for decades, putting them together may represent a paradigm shift (Paulk, 2001). It will be interesting to watch how universal these practices ultimately become.

3. RELATED WORK

In this section we review prior work on the compatibility of CMM and XP and, more generally, on the compatibility of plan-driven and agile methods.

3.1 Comparisons of CMM and XP

The earliest work on the compatibility of CMM and XP was written by Jeffries (2000). Jeffries noted that XP has some characteristics in common with the higher CMM levels, up to and including level 5. XP is not a level 1 process: it is not chaotic and rejects individual heroics. On the contrary, XP requires that a number of best practices are followed. These practices make success repeatable. XP thus fulfills the key level 2 requirement.

XP fulfills many level 3 requirements as well. XP's practices include readiness criteria (completion of user

stories), verification mechanisms (unit and functional tests), and completion criteria (user-acceptable scores on the functional tests). In addition, management can monitor project performance through four XP variables: resources, scope, quality, and time.

Jeffries (2000) next argues that XP is a CMM level 4 compatible quantitatively managed process. Quality goals are set for products via functional tests. Unit tests must run at 100% all the time. Project velocity is calculated and adjusted as the team gains more experience. On the other hand, Jeffries notes that CMM organizations usually measure more things than is required in a single XP project. To be compatible with agility, measurements should be low cost and only used when needed.

Defect prevention, a CMM level 5 practice, is addressed in XP with testing. The tests prevent future occurrences of old errors. The idea is to automate defect prevention rather than set up involved procedures that leave open the possibility of human error.

Although XP addresses many CMM requirements, Jeffries (2000) concludes that an XP team would not lie at CMM level 5. That would require more documentation and proving than is recommended in XP.

Jeffries's work was written from an agile perspective (Jeffries was one of the brains behind XP). Perhaps the first analysis of XP from a CMM perspective was written by Mark Paulk (2001). Paulk argued that CMM and XP are, in fact, complementary. The software CMM focuses on management issues and on systematic process improvement, while XP focuses more on engineering issues. The CMM tells organizations what to do, but does not say how to do it. In contrast, XP is a set of well-defined practices with specific how-to guidelines. XP can thus be seen as a CMM

Level	Key process area	Satisfaction
2	Requirements management	++
2	Software project planning	++
2	Software project tracking and oversight	++
2	Software subcontract management	-
2	Software quality assurance	+
2	Software configuration management	+
3	Organization process focus	+
3	Organization process definition	+
3	Training program	-
3	Integrated software management	-
3	Software product engineering	++
3	Intergroup management	++
3	Peer reviews	++
4	Quantitative process management	-
4	Software quality management	-
5	Defect prevention	+
5	Technology change management	-
5	Process change management	-

+ Partially addressed in XP
 ++ Largely addressed in XP (perhaps by inference)
 - Not addressed in XP

Common feature	Practice	Satisfaction
Commitment to perform	Policy	-
	Leadership and sponsorship	-
Ability to perform	Organizational structures	+
	Resources and funding	+
	Training	+
Measurement and Analysis	Measurement	+
Verifying implementation	Senior management oversight	-
	Project management oversight	++
	Software quality assurance	+

+ Partially addressed in XP
 ++ Largely addressed in XP (perhaps by inference)
 - Not addressed in XP

implementation model.

Paulk (2001) substantiates his argument by analyzing which CMM Key Process Areas are addressed in XP. The results are presented in Table 3. It is interesting to note that unlike Jeffries, Paulk does not think that XP's quantitative metrics make XP a quantitatively managed process.

Although both CMM and XP are concerned with culture, CMM emphasizes institutionalization more than XP (Paulk, 2001). The CMM institutionalization process consists of the five steps described in Section 2.1. Table 4 shows the degree to which XP addresses the CMM institutionalization goals. Paulk does not include the Activities Performed step in his analysis, presumably because it assumed to be fulfilled by the 12 XP practices.

Paulk's analysis shows that XP addresses the CMM institutionalization goals only partially. In particular, XP does not address the Commitment to Perform goal. Paulk (2001) maintains that although many of these institutionalization issues may fall outside XP's scope, they are crucial for its successful adoption. We will have more to say about institutionalization in Section 4.

The discussion of XP and institutionalization is followed by a discussion of XP's applicability (Paulk, 2001). While CMM can scale from a few developers to hundreds of developers, XP probably cannot. XP's lack of design documentation also means that XP should not be used for life-critical or high-reliability systems. On the other hand, Paulk likes XP's values and states that organizations using the software CMM should focus on simplicity and communication just as XP does.

Paulk (2001: 8) concludes: "The argument that CMM's ideal of a rigorous, statistically stable process is antithetical to XP is unconvincing. [...] We can thus consider XP and CMM complementary." As we will see, this is the view most authors have adopted. It will be put under critical scrutiny in Section 4.

Paulk's comparison was extended by Martinsson (2002). Martinsson analyzes how well the 12 core practices of XP satisfy the 52 goals of the CMM. The results are similar to Paulk's. At level 2, for example, all goals of the Requirements

management and Software project planning KPAs are fully satisfied.

Two differences stand out when we compare Martinsson's (2002) study to that of Paulk's (2001). First, unlike Paulk, Martinsson thinks that pair programming can at least partially satisfy the three goals of the Training KPA. Paulk would probably prefer a more formal training program. The second difference is that Martinsson believes that XP partially addresses the goals of the Quantitative process management KPA. The heart of the matter, as Martinsson notes, is what is meant by "quantitative control." Martinsson concludes that user story and engineering task estimates fit the bill. Paulk, on the other hand, reasons that XP addresses few of the level 4 or 5 KPAs in a rigorous statistical sense. We will return to this debate in Section 4.2.

3.2 Idea compatibility of CMM and XP

A complementary perspective on the compatibility of CMM and XP is provided by Hillel Glazer (2001). The studies discussed above conducted a practice compatibility analysis. They showed that CMM and XP can be *practice compatible*, in that XP can be an implementation model for CMM-based process improvement. Glazer is more concerned with *idea compatibility* (our characterization). He tries to dispel what he dubs the process myth: that having a process means sacrificing agility or creativity.

According to Glazer (2001), many commercial developers mistakenly believe that process discipline is incompatible with speedy development. This misconception is held by both process-oriented people, who often think that speed equates to chaos, and fans of agile methods, who often think that CMM and similar methods mean nothing but cumbersome paperwork. Glazer believes nothing could be further from the truth. Process discipline does not have to send a chill down the developer's spine, and XP does not resemble chaotic hacking. Process discipline can be achieved without sacrificing the speed of development.

Glazer (2001) gives several reasons for his view. The first is the customizability of CMM. Although CMM has traditionally been associated with extremely heavy processes at government contractors (and with good reason), in no way does it mandate a bureaucratic process. On the other hand, it would be wrong to call agile methods undisciplined or immature. XP, for example, includes many of the hard-taught lessons learned from many years of practical development experience.

Glazer (2001) suggests that CMM is best seen as a management methodology and XP as a development methodology. These two methodologies can and should work together. Glazer's conclusion mirrors that of Paulk's (2001): CMM and XP are complementary.

Glazer (2001) makes one more noteworthy point: that Quality Assurance (QA) is not Quality Control (QC). The difference between QC and QA is that QA is process oriented while QC is product oriented. As defined by Glazer, QC is testing, which is definitely a major part of the XP methodology. In Glazer's words, testing *controls* quality; it

does not *assure* it.

QA, then, is the process of verifying quality. Glazer (2001) suggests that there are three elements to QA that XP does not address: management visibility, independent review and audit, and assurance of the application of standards and the development process. Glazer thus sees quality assurance as formal external review, which is not a typical XP practice (Martinsson, 2002). Instead, XP prefers pair programming, tests, and refactoring. The risk here is that the good practices may be abandoned under pressure. This topic will receive more attention in Section 4.

3.3 Limits of agile methods

Not content to declare that XP can be used with CMM, Paulk (2002) recently launched a counteroffensive, critiquing agile methods through the four values of the Agile Manifesto. Paulk's arguments are described below.

Paulk first argues that the focus of agile methods on people over process will leave many organizations cold. He points out that agile methods require good generalists to execute them. These people, however, are in short supply. Even if they can be found, management often puts unreasonable demands on them. To address the management skills gap, something like CMM will then be needed.

Paulk next draws attention to the unfortunate tendency by agile advocates to denigrate the value of documentation. It is quite different, Paulk notes, to argue against non-essential documentation than to argue against all documentation. Even if we accept that non-essential documentation is to be avoided (which we should), agile methods fail to explain how they should be scaled up to large teams and life-critical and mission-critical software. In these domains, a large amount of documentation must be produced regardless of the process.

Paulk then expresses his reservations with the value "customer collaboration over contract negotiation." He notes that there must a high degree of trust to rely on customer collaboration. Enough trust cannot always be established; then contracts must be used. Indeed, in many customer-supplier relationships, such as those involving government customers, extensive collaboration may be impossible.

Paulk next critiques the agile preference for responding to change over following a plan. Quoting Dwight Eisenhower, he points out that planning is important even if the plans themselves may not be. And agile methodologies themselves do a fair amount of planning. Paulk closes his critique as follows: "Planning for change is quite different from not planning at all."

Paulk concludes his paper in a new tone. He concedes that even though agile methodologies may be in principle compatible with the discipline of models such as the CMM, combining the two may prove difficult, if not impossible, due to differences in the "spirit of the agile philosophy" and that of traditional disciplined models. In other words, agile methods and traditional disciplined models may be practice compatible but not idea compatible, using the terminology introduced earlier. In Paulk's view, the rift between the two is real. For

example, he warns that aligning the two approaches in a government-contracting context may be an insurmountable challenge.

3.4 Combining agile and plan-driven methods

While Paulk's work delineates the boundaries of agile methods, Boehm's (Boehm, 2002; Boehm & Turner, 2003) gives structured advice on when agile methods and plan-driven methods should be applied. Boehm's first paper (2002) advocates using risk management to determine the right mix of plan-driven and agile methods. There are two sources of risks in any software project: inadequate plans and project delays. Inadequate plans cause oversights, delays, and rework, while project delays lead to a loss of value capture. The task is to minimize total risk exposure by investing the right amount of time and effort in plans. Figure 1 illustrates this idea.

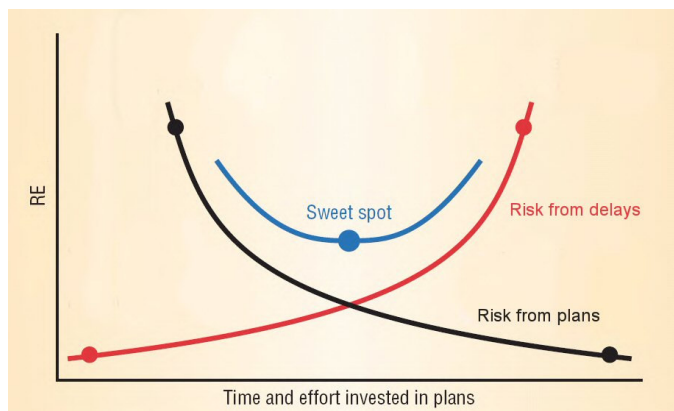


Figure 1 Risk exposure (RE) as a function of time and effort invested in plans (Boehm, 2002).

This risk-based approach was extended in Boehm & Turner (2003). Boehm and Turner propose a five-step process for tailoring a suitable process for an organization. First, risk analysis is applied to map the project's environmental, agile, and plan-driven risks. If the analysis suggests that either purely agile or purely plan-driven methods should be used, the project falls in the home ground of one of the methods. Then the appropriate method is chosen. If the analysis shows that the project does not fall into the home grounds of either method, the project team should develop an architecture that encapsulates agile parts. The agile parts are developed with agile methods and the rest with plan-driven methods. Finally, the development process is constantly monitored and improved.

When does a method fall in the home ground of either agile or plan-driven methods? Boehm and Turner (2003) present a framework for answering this question. The framework consists of five axes. Agile methods will be preferred when team size is small, software is not critical, environment is dynamic, personnel is experienced, and culture thrives on chaos. Conversely, plan-driven methods will be preferred when team size is large, software is critical, environment is stable, personnel are inexperienced, and culture thrives on order. Figure 2 illustrates the framework.

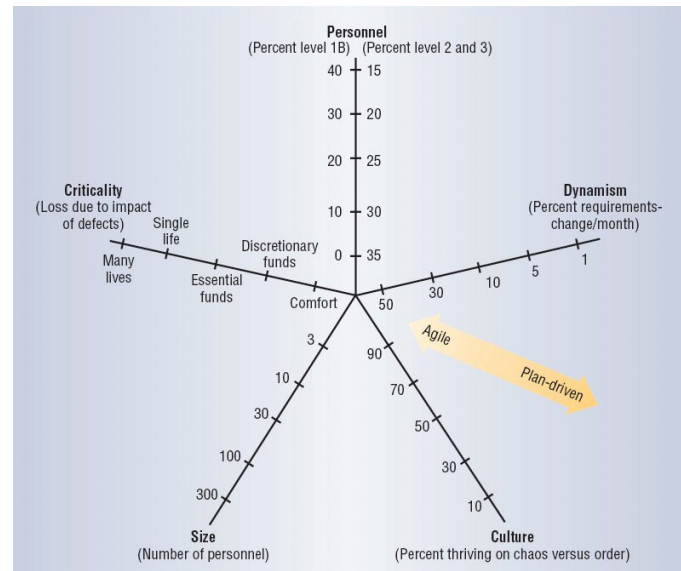


Figure 2 A framework for choosing between agile and plan-driven methods (Boehm & Turner, 2003).

With this background on the compatibility of CMM and XP and agile methods and plan-driven methods, we are ready to address the question: are CMM and agile methods really compatible?

4. COMPATIBILITY OF CMM AND AGILE METHODS: FOUR PERSPECTIVES

In this section we examine the compatibility of CMM and agile methods from four perspectives. The perspectives are complementary. The primary focus of the four perspectives is on idea compatibility, although section 4.2 briefly touches on practice compatibility. Together these perspectives shed light on the true compatibility of CMM and agile methods and limits thereof.

4.1 What does discipline mean?

Most authorities agree that agile methods are “disciplined” (Paulk, 2001; Glazer, 2001; Boehm, 2002). This leaves open the question of what discipline actually means.

Kent Beck (Beck & Boehm, 2003: 44) gives five dictionary definitions for discipline. Discipline can be “a field of study”; “training that corrects, molds, or perfect the mental faculties and moral character”; “control gained by obedience or order”; “orderly or prescribed conduct or pattern or behavior”; and “a rule or system of rules governing conduct or activity.”

Beck infers that XP can be said to be disciplined in all these senses except one: control gained by obedience or order. Beck (Beck & Boehm, 2003: 44) in fact explicitly rejects that definition: “If this narrow definition of discipline is used, I say hooray for undisciplined processes. Efforts to force developers and customers to work according to a procedure developed by those not immediately responsible for results have uniformly failed.”

Beck's inference, that externally-imposed processes are useless, is contested later in the same paper by Barry Boehm (Beck & Boehm, 2003). He thinks that CMM level 3 efforts

have brought about substantial improvements in process quality. Here there is a large disagreement that merits further discussion.

The roots of CMM can be traced back to scientific management and its ideal of managed, repeatable, and quantifiable organizational processes (Lycett et al., 2003). The rationalism of scientific management was first applied in manufacturing. It was expected that it would apply to software programming as well.

In traditional engineering, the task of management is to plan, execute, and control (Lycett et al., 2003). This characterization of managerial work underlies CMM's worldview, as a casual reading of the CMM specification (Paulk et al., 1993a) reveals. The five CMM common features almost exactly mirror the plan-execute-control cycle. So do the 52 goals of the 18 Key Process Areas.

Agile methods grew out of dissatisfaction with the plan-execute-control cycle (Fowler, 2003). It had long been understood that the construction metaphor applies only imperfectly to software (McConnell, 1993). The rationalism of scientific management was supplemented by a focus on people and communication. Programmers would no longer be replaceable parts, as scientific management expected them to be (Fowler, 2003).

It is true that CMM adapts scientific management to the realities of software production. Indeed, the Software Engineering Institute created the People CMM (P-CMM) (Curtis et al., 2001) to address many of the critical people issues facing organizations. But the specification leaves no doubt that the ideological orientation of the P-CMM lies firmly in the "external knowledge capture" camp rather than the "peer-based knowledge capture" camp advocated by agile methods (terms defined in Lycett et al. (2003)). Managers develop their subordinates, institute training programs, define required competences, and try to create the right incentives for people to learn the competences. The P-CMM, in other words, is also a product of scientific rationalism.

We can by now see that "discipline" is not synonymous in CMM and agile methods. In CMM managers control the show. They define what must be accomplished, plan, execute, and control. In CMM, therefore, the definition of discipline includes the final sense Beck explicitly rejects: control gained by obedience or order.

The difference can be characterized as follows. CMM defines discipline as "adherence to standards," whereas agile methods define discipline as "adherence to principles." The difference is one of degree. Agile methods see discipline as following good practices even under pressure. But they are usually not dogmatic about the practices. XP, for example, endorses a "just rules" approach, whereby rules can be modified as necessary. CMM, in contrast, functions under the "say what you do, then do what you say you do" mentality (Lycett et al., 2003). To deviate from plans and standards is seen as undisciplined and immature – a failure.

It is interesting to note that CMM and agile methods both

value discipline and defined practices. Viewed from this perspective, agile methods could even be called conventional. *Anarchist programmers*, by contrast, would reject discipline and say "we need no process." We will return to this point in Section 6.

4.2 What do we measure?

There have been different interpretations of whether XP satisfies CMM level 4 (Managed). Paulk (2001) does not think so, while Martinsson (2002) partially does. The heart of the matter, as Martinsson notes, is what is meant by "quantitative control." We next analyze whether XP really addresses the KPAs at CMM level 4. More broadly, we compare and contrast what CMM and agile methods typically measure.

CMM level 4 contains two KPAs (Paulk et al., 1993b): Quantitative Process Management and Software Quality Management. The purpose of Quantitative Process Management is to control process performance quantitatively. The purpose of Software Quality Management, on the other hand, is to develop a quantitative understanding of the quality of the end product.

The Quantitative Process Management KPA contains three goals. They are: 1) The quantitative process management activities are planned; 2) The process performance of the project's defined software process is controlled quantitatively; and 3) The process capability of the organization's standard software process is known in quantitative terms.

In Martinsson's (2002) view, XP satisfies the first goal fully and the second partially, but fails to satisfy the third. The first goal is satisfied because the project tracker calculates how many units of work the team and each individual programmer have been able to deliver during each iteration. The second goal is partially satisfied by a practice often referred to as "yesterday's weather," which implies that an XP team should schedule exactly as much work as the team's average velocity indicates. The third goal is not satisfied in XP and will have to be implemented. This implementation could mean collecting project velocity and other statistics from prior projects and using them in current projects to forecast productivity.

We disagree with Martinsson's analysis and concur with Paulk's (2001). To illustrate, we use the following quote from the CMM 1.1 specification (Paulk et al., 1993b: L4-1): "Quantitative Process Management involves establishing goals for the performance of the project's defined software process, [...] taking measurements of the process performance, analyzing these measurements, and making adjustments to maintain process performance within acceptable limits. When the process performance is stabilized within acceptable limits, the project's defined software process, the associated measurements, and the acceptable limits for the measurements are established as a baseline and used to control process performance quantitatively."

Quantitative process management, as defined by CMM, involves establishing goals for the performance of the process. The XP planning game does not set goals for the *performance* of the process; it only sets goals for what should be

accomplished, in light of observed performance. The XP planning game, in other words, is descriptive, whereas quantitative process management is prescriptive.

Quantitative process management also involves maintaining process performance within acceptable limits. This stabilization process resembles statistical quality control. The ideal CMM process produces precise, repeatable measurements. In XP, project velocity assumes some degree of stability in iteration-to-iteration performance, but stability is not seen as a goal, as it is in the CMM.

Some agile methodologists have regarded quantitative measurements with scepticism. In scientific management there was a strong push to develop objective approaches to measuring the output of people (Fowler, 2003). According to Fowler (2003), however, those attempts failed. The simplest things about software, such as productivity, cannot still be measured. And without proper measures, any kind of external control is doomed.

Fowler (2003) approvingly quotes Robert Austin as saying that when measuring performance you have to get all the important factors under measurement. Otherwise the doers of a process will alter their behavior to produce the best measures, even if that reduces the true effectiveness of the process. This implies that measurement-based management is best suited for repetitive, simple work – exactly the opposite of software development.

Fowler apparently believes that the lack of good measures undermines the case for quantitative process management. But there is another line of argumentation that leads to the same conclusion. Agile methods in general give more weight to individuals than CMM does. If the abilities of people differ widely, quantitative measures are in the long run unstable. Then it will not be possible to “maintain process performance within acceptable limits.”

Those agile advocates who agree that measurements either cannot be defined or are unstable will likely reject CMM level 4. On the other hand, we cannot rule out the possibility that there are agile advocates who believe in the stability of measurements (see, for example, Jeffries, 2000). These people would likely find CMM level 4 compatible with their worldview.

The picture that emerges from this comparison is one of compatibility and incompatibility. CMM measurements can be compatible with agile methods, but many agile developers will probably disagree with the assumptions behind CMM level 4. The controversy stems from a battle between scientific management, which believes that programmer work can be measured, and agile values, which uphold the dignity of people. It will be interesting to watch how this controversy will be resolved.

4.3 Who drives process improvement?

It is easy to accept that CMM is a management methodology and XP an engineering methodology (Glazer, 2001). Although this means that CMM and XP can be complementary, it also creates an inconsistency.

The roots of CMM, as noted, can be traced back to scientific management and its ideal of managed, repeatable, and quantifiable organizational processes (Lycett et al., 2003). CMM makes no secret of who it is that drives process improvement. Managers first get their own house in order at level 2 and then impose processes on others at level 3 (Paulk et al., 1993a). The process is further stabilized and then continuously enhanced at levels 4 and 5.

An important CMM concept is visibility into the software process (Paulk et al., 1993a). Senior managers rely on visibility to track a project’s progress. One of the benefits of CMM, the CMM specification notes, is that each succeeding maturity level incrementally provides better visibility into the software process. At level 1, the software process is a single black box. At level 2, the structure of the process becomes visible as a series of black boxes. At level 3, managers have visibility to the internals of the boxes. Finally, at levels 4 and 5, managers are able to measure progress and problems quantitatively.

Another key CMM concept is the prediction of performance. As maturity increases, expected and realized outcomes become more and more similar. The variability of realized results decreases. Expected results also improve, so that a level 5 organization produces software quickly and predictably.

These features produce a need for managerial control. Indeed, the “say what you do, then do what you say you do” mentality requires frequent audits (Lycett et al., 2003). Audits, in turn, require tangible outputs, which are later verified by managers. We can see that in CMM, the manager is king.

Agile methods do not ignore the role of a process, but they reject the notion that a good process must be standardized or repeatable (Lycett et al., 2003). In the agile worldview, emergent situations never give rise to repeatability, because the development environment is in a constant state of flux.

An agile perspective also recommends that the process is adopted voluntarily. According to Fowler (2003), only the developers themselves can choose to follow a process. Managers cannot impose it on them. If they try, they will be resisted, particularly if they have had a significant time from active development. This flies in the face of the CMM.

Fowler (2003) characterizes the ideal relationship between managers and developers as one of equal partnership. Developers make all technical decisions and provide input to the planning process by creating task estimates. Business leadership, on the other hand, guides the team on business needs.

It is illustrative to compare Kent Beck’s (2000) management strategy to the management strategy of the CMM. Management in XP is divided into two roles: the coach and the tracker. The job of the coach is to get everybody else making good decisions. The coach is always available as a development partner, helps programmers with individual technical skills, and explains the process to upper-level managers. The tracker, by contrast, gathers whatever metrics

are tracked and posts them for all to see. The tracker also runs the XP planning game.

The XP management strategy paints a picture of a benevolent manager who is there mainly to help others succeed. The manager intervenes actively only when it cannot be avoided. When CMM speaks of statistical quality control, process visibility, and predictable performance, the images that spring to mind are quite different. They are those of a Taylorist manager, keenly watching over the programmer's shoulder to see if he is slacking off.

This section has indeed uncovered a difference between CMM and agile methods. In CMM, the manager is king; in agile methods, managers and programmers are supposed to be equal partners. Against this background, it is interesting to note that the aim of the two approaches is the same – namely, to maximize business value. Agile methods only trust the programmer more than the CMM does. We can once again see a battle being waged between the ideas of scientific management and those of agile methods.

4.4 What do we value?

The battle between scientific management and agile methods can be recast at a deeper level in terms of values. Agile methods value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan (Beck et al., 2001). CMM does not make its values explicit, but some of them can be deduced.

The CMM clearly values process standardization. The predictability of process performance is very important (Paulk et al., 1993a). The CMM was developed for government contractors, whose bidding would have been greatly facilitated by standard performance.

The antithesis of standard performance is individual heroics. At CMM level 1, “performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations.” (Paulk et al., 1993a) In CMM there is no place for such variation. The first CMM value is therefore process standardization over individual heroics.

The CMM clearly values managerial visibility into the process. In Section 4.3 we argued that in CMM, the manager is king. Scientific management would like workers to be replaceable. The second CMM value, therefore, is management control over programmer initiative.

It must be immediately noted that this value is shared by few plan-driven experts (Turner & Boehm, 2003). Nevertheless, the value is implicit in the CMM specification. In addition, it is probably shared by more than a few managers (Lycett et al., 2003; Cohn & Ford, 2003).

It is noteworthy that in CMM continuous improvement only starts at level 5. By contrast, agile methods stress continuous improvement from the beginning. The CMM first tries to stabilize performance through defined and later quantitatively managed processes. This gives rise to the third CMM value: organizational stability over discontinuous progress.

There are clearly differences between agile values and the values of the CMM. Perhaps the greatest are that 1) CMM tends to advocate more imposing management control than agile methods do and 2) CMM values stability more. These differences will guide us when we in Section 6 unveil the managerial implications of our analysis.

It is interesting to note how many people from plan-driven backgrounds (Paulk, 2001; Glazer, 2001; Boehm & Turner, 2003) have viewed agile methods as CMM-compliant, but not necessarily the other way around (Fowler, 2003; Beck & Boehm, 2003). Could this be a consequence of the greater focus that agile methods place on values and, by extension, idea compatibility? Alternatively, the different views on compatibility could be explained by different interpretations of the applicability of agile values. If agile values are universal, then plan-driven methods are probably ruled out. Plan-driven proponents dispute any claims to universality (Boehm, 2002) and therefore tend to see agile methods as applying only in limited circumstances.

It is not possible to ignore value issues in any process improvement effort. If there is no agreement on values, misunderstandings will arise and people will split into factions (Reifer, 2003). The partial incompatibility of agile and CMM values does not bode well for efforts to combine the two.

5. DISCUSSION

The preceding analysis paints a mixed picture of compatibility and incompatibility: compatibility in the realm of practices and incompatibility in the realm of ideas. Although CMM and agile methods are both disciplined, CMM defines discipline as “adherence to standards,” whereas agile methods define discipline as “adherence to principles.” Although CMM measurements can be compatible with agile methods, many agile developers will probably disagree with the assumptions behind CMM level 4. Although CMM and agile methods both try to maximize business value, in CMM the manager dominates, whereas in agile methods managers and programmers are equal partners. Although CMM and agile methods are practice compatible, their different values can make them idea incompatible.

What causes the idea incompatibility of CMM and agile methods? We think the idea incompatibility is best explained by a disconnect between the ideas of scientific management and those of agile methods. Agile methods reject many of the assumptions behind scientific management. They do not believe programmers are replaceable human resources. They do not believe in the infallibility of planning. They violently reject externally imposed control.

This leads to our conclusion. CMM and agile methods are compatible if either 1) agile methods are used devoid of their value content, or 2) CMM is used devoid of its value content. As an example of 1), managers could impose agile methods on an organization. As an example of 2), a team could agree to use CMM but ignore its planning focus. These examples illustrate customization: using the methods in a slightly

different manner than their creators originally envisaged.

Our conclusion suggests a future line of development for CMM-like process improvement tools. There is still a need for process improvement frameworks. The problem with CMM, viewed from an agile perspective, lies in its ideological orientation. The way the CMM specification is written will certainly irritate many, if not most, agile proponents. But it should be possible to create a CMM-like framework that is more compatible with agile ideas. The new framework would drop the rigid plan-driven focus and embrace change. It would drop the Taylorist ideology and value the uniqueness of people. And it would drop the emphasis on contracting and extol the virtues of communication. This framework could be called the *Agile CMM*.

The agile CMM could become a framework for discussing agile methods. Different agile methods would address the model's key process areas differently. The model would facilitate not only discussion but also the development of new agile methods. The model could be used to gain an understanding of an agile method's scope and applicability. In addition, it could be used by the team to drive process improvement in accordance with business priorities.

6. MANAGERIAL IMPLICATIONS

It is not possible to ignore value issues in any process improvement effort. Managers would do well to assess their own and their organization's values and reflect them against the ideas of scientific management and those of agile methods. An organization valuing innovation, for example, will move towards the controlled change of agile methods, whereas one valuing stability may want to start a formal CMM improvement process.

CMM tends to advocate more imposing management control than agile methods. Managers must therefore assess if their programmers want guidance or control. Programmers accustomed to agile methods may resist the CMM as an attempt at micromanagement. Conversely, programmers accustomed to plan-driven methods may view agile methods as an attempt at micromanagement (Cohn & Ford, 2003).

Anarchist programmers, whom Barry Boehm (2002) dubs "erstwhile cowboys," will reject the imposition of *any* process. They will violently reject all CMM concepts and all that it represents. They may also purport to follow agile methods when they in fact do not. We suspect many open source programmers fall into this camp. If an organization is packed with anarchist programmers, process improvement efforts are doomed to irrelevance.

CMM values stability more than agile methods do. The CMM ideal of organization-wide standardized performance may create disgust among agile proponents, but standardized performance can be extremely valuable for some organizations. Software contractors, in particular, would find standardized performance very useful. We may thus expect that CMM will be used more frequently in subcontracting organizations. This indeed appears to be the case, as a casual

look at Indian software companies reveals.

Our previous analysis showed that CMM and agile methods are compatible if their value contents are ignored. On the other hand, this does not prevent an organization from using CMM on some projects and agile methods on others. The choice can be made for example with Boehm and Turner's (2003) framework.

7. CONCLUSION

Are CMM and agile methods really compatible? Yes and no. They are practice compatible but not idea compatible. Combining the two is possible, but will run into difficulties if value issues are ignored.

The incompatibility mainly stems from a battle between the ideas of scientific management and those of agility. A CMM-like process improvement tool, written with the agile values in mind, could reconcile the differences and become a framework for discussing agile methods.

REFERENCES

- Abrahamsson, P. et al. 2003, "New Directions on Agile Methods: A Comparative Analysis," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 244–255.
- Beck, K. 2000, *Extreme Programming Explained*, Addison-Wesley, Boston, MA.
- Beck, K. et al. 2001, "Manifesto for Agile Software Development," <http://www.agilemanifesto.org>. Referenced 28.3.2003.
- Beck, K. & Boehm, B. 2003, "Agility through Discipline: A Debate," *IEEE Computer*, vol. 36, no. 6, pp. 44–46.
- Boehm, B. 2002, "Get Ready for Agile Methods, with Care," *IEEE Computer*, vol. 35, no. 1, pp. 64–69.
- Boehm, B. & Turner, R. 2003, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Computer*, vol. 36, no. 6, pp. 57–66.
- Clegg, C., et al. 1996, *The Performance of Information Technology and the Role of Human and Organizational Factors*, technical report, Economic and Social Research Council, UK.
- Cohn, M. & Ford, M. 2003, "Introducing an Agile Process to an Organization," *IEEE Computer*, vol. 36, no. 6, pp. 74–78.
- Curtis, B., Hefley, B., & Miller, S. 2001, "People Capability Maturity Model (P-CMM) Version 2.0," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU/SEI-2001-MM-001.
- Fowler, M. 2003, "The New Methodology," <http://www.martinfowler.com/articles/newMethodology.html>. Modified April 2003. Referenced 28.3.2004.
- Glazer, H. 2001, "Dispelling the Process Myth," *CrossTalk*, November 2001, pp. 27–30.
- Jeffries, R. 2000, "Extreme Programming and the Capability Maturity Model," http://www.xprogramming.com/xpmag/xp_and_cmm.htm. Modified 1.1.2000. Referenced 28.3.2004.
- Lycett, M., Macredie, R., Patel, C., & Paul, R. 2003, "Migrating Agile Methods to Standardized Development Practice," *IEEE Computer*, vol. 36, no. 6, pp. 79–85.
- Martinsson, J. 2002, "Maturing Extreme Programming Through the CMM," Master's thesis, Dept. Computer Science, Lund Univ., Lund, Sweden.
- McConnell, S. 1993, *Code Complete*, Microsoft Press, Redmond, WA.

Paulk, M., Curtis, B., Chrissis, M. & Weber, C. 1993a, "Capability Maturity Model for Software (Version 1.1)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU/SEI-93-TR-024.

Paulk, M., Weber, C., Garcia, S., Chrissis, M. & Bush, M. 1993b, "Key Practices of the Capability Maturity Model Version 1.1," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU/SEI-93-TR-025.

Paulk, M. 2001, "XP from a CMM Perspective," *IEEE Software*, vol. 18, no. 6, pp. 19–26.

Paulk, M. 2002, "Agile Methodologies and Process Discipline," *CrossTalk*, October 2002, pp. 15–18.

Reifer, D. 2003, "XP and the CMM," *IEEE Software*, vol. 20, no. 3, pp. 14–15.

Software Engineering Institute (SEI) 2002. "Capability Maturity Model Integration (CMMI), Version 1.1," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU/SEI-2002-TR-012.

Software Engineering Institute (SEI) 2003, "Upgrading from SW-CMM to CMMI," white paper, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Turner, R. & Boehm, B. 2003, "People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods," *CrossTalk*, December 2003, pp. 4–8.