Extreme Programming from a CMM Perspective

Mark C. Paulk
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268-5794
mcp@sei.cmu.edu

ABSTRACT
Extreme Programming (XP) has been advocated recently as an appropriate programming method for the high-speed, volatile world of Internet and Web software development. This popular methodology is reviewed from the perspective of the Capability Maturity Model® (CMM®) for Software, a five-level model that prescribes process improvement priorities for software organizations. Overviews of both XP and CMM are provided, and XP is critiqued from a Software CMM perspective. The conclusion is that lightweight methodologies such as XP advocate many good engineering practices, although some practices may be controversial and counter-productive outside a narrow domain. For those interested in process improvement, the ideas in XP should be carefully considered for adoption where appropriate in an organization's business environment since XP can be used to address many of the CMM Level 2 and 3 practices. In turn, organizations using XP should carefully consider the management and infrastructure issues described in the CMM.

Keywords
Software CMM, Capability Maturity Model, CMM, Extreme Programming, XP, agile methodologies, lightweight processes

1 INTRODUCTION

In recent years, Extreme Programming (XP) has been advocated as an appropriate programming method for the high-speed, volatile world of Internet and Web software development. XP can be characterized as a "lightweight" or "agile" methodology. Although XP is a disciplined process, some have used it in arguments against rigorous models for software process improvement, such as the Capability Maturity Model for Software, a five-level model that prescribes process improvement priorities for software organizations developed by the Software Engineering Institute (SEI). Many organizations moving into e-Commerce have existing CMM-based initiatives (and possibly customers demanding mature processes) and desire an understanding of whether and how XP can address CMM practices adequately.

This paper summarizes both XP and CMM and critiques XP from a CMM perspective. Although XP can be characterized as a lightweight methodology that does not emphasize process definition or measurement to the degree that models such as the CMM do, a broad range of processes can be considered valid under the CMM. The conclusion is that agile methodologies such as XP advocate many good engineering practices, although some practices may be controversial and counter-productive outside a narrow domain, and that when thoughtfully implemented in an appropriate environment, XP addresses many CMM Level 2 and 3 practices. For those interested in process improvement, the ideas in XP should be carefully considered for adoption where appropriate in an organization's business environment, just as organizations considering XP should carefully consider the management and infrastructure issues described in the CMM.

2 THE SOFTWARE CMM

The Capability Maturity Model for Software [5, 6] is a model for building organizational capability that has been widely adopted in the software community and beyond. The Software CMM is a five-level model that describes good engineering and management practices and prescribes improvement priorities for software organizations. The five maturity levels are summarized in Figure 1.

The Software CMM is intended to be:

- a *common-sense* application of process management and quality improvement concepts to software development and maintenance -- the CMM practices are not rocket science (even the statistical process control concepts at Levels 4 and 5 have been successfully applied in other industries for decades)
- a *community-developed* guide -- input from hundreds of software professionals was solicited in developing the current release of the CMM

® Capability Maturity Model and CMM are registered with the U.S. Patent and Trademark Office.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

- a model for *organizational* improvement -- which implies a set of priorities that may differ from those of any specific project, but which have been proven effective in organizational transformation
- the underlying structure for *reliable and consistent* CMM-based appraisal methods -- assessments and evaluations based on the Software CMM are widely used by software organizations for improvement and customers for understanding the risks associated with potential suppliers

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 & Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
1 Initial	*Competent people and heroics*	

Figure 1. An overview of the Software CMM.

Although the CMM is described in a book of nearly 500 pages, the requirements to be a Level 5 organization can be concisely stated in 52 sentences: the goals of the 18 key process areas (KPAs) that formally describe the model. The practices, subpractices, and examples that flesh out the model are informative material that guide software professionals in making reasonable, informed decisions about the adequacy of a broad range of process implementations – in environments as diverse as 2-3 person projects in a Web environment and 500 person projects building hard real-time, life-critical systems.

The informative material in the Software CMM is focused on large projects and large organizations, primarily in a custom development or maintenance environment. Even so, the degree of interpretation and tailoring required to use the CMM in radically different environments, such as small start-up companies, small projects, or e-Commerce environments, is relatively minor so long as common sense is applied [7, 4]. The Software CMM's rating components are intended to be abstract enough to capture "universal truths" about high-performance software organizations, at least from a perspective of organizational excellence, and are listed in Table 1.

Table 1. Purpose and Goals of Software CMM Key Process Areas.

Tag	KPA Purpose and Goals
	Maturity Level 2 – Repeatable
Requirements Management	*… to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project.*
RM Goal 1	System requirements allocated to software are controlled to establish a baseline for software engineering and management use.
RM Goal 2	Software plans, products, and activities are kept consistent with the system requirements allocated to software.
Software Project Planning	*… to establish reasonable plans for performing the software engineering and for managing the software project.*
SPP Goal 1	Software estimates are documented for use in planning and tracking the software project.
SPP Goal 2	Software project activities and commitments are planned and documented.
SPP Goal 3	Affected groups and individuals agree to their commitments related to the software project.
Software Project Tracking & Oversight	*… to provide adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.*
SPTO Goal 1	Actual results and performance are tracked against the software plans.
SPTO Goal 2	Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.
SPTO Goal 3	Changes to software commitments are agreed to by the affected groups and individuals.
Software Subcontract Management	*… to select qualified software subcontractors and manage them effectively.*
SSM Goal 1	The prime contractor selects qualified software subcontractors.
SSM Goal 2	The prime contractor and the software subcontractor agree to their commitments to each other.
SSM Goal 3	The prime contractor and the software subcontractor maintain ongoing communications.
SSM Goal 4	The prime contractor tracks the software subcontractor's actual results and performance against its commitments.
Software Quality Assurance	*… to provide management with appropriate visibility into the process being used by the software project and of the products being built.*
SQA Goal 1	Software quality assurance activities are planned.
SQA Goal 2	Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.
SQA Goal 3	Affected groups and individuals are informed of software quality assurance activities and results.
SQA Goal 4	Noncompliance issues that cannot be resolved within the software project are addressed by senior management.

Tag	KPA Purpose and Goals
Software Configuration Management	*… to establish and maintain the integrity of the products of the software project throughout the project's software life cycle.*
SCM Goal 1	Software configuration management activities are planned.
SCM Goal 2	Selected software work products are identified, controlled, and available.
SCM Goal 3	Changes to identified software work products are controlled.
SCM Goal 4	Affected groups and individuals are informed of the status and content of software baselines.
	Maturity Level 3 -- Defined
Organization Process Focus	*… to establish the organizational responsibility for software process activities that improve the organization's overall software process capability.*
OPF Goal 1	Software process development and improvement activities are coordinated across the organization.
OPF Goal 2	The strengths and weaknesses of the software processes used are identified relative to a process standard.
OPF Goal 3	Organization-level process development and improvement activities are planned.
Organization Process Definition	*… to develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.*
OPD Goal 1	A standard software process for the organization is developed and maintained.
OPD Goal 2	Information related to the use of the organization's standard software process by the software projects is collected, reviewed, and made available.
Training Program	*… to develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.*
TP Goal 1	Training activities are planned.
TP Goal 2	Training for developing the skills and knowledge needed to perform software management and technical roles is provided.
TP Goal 3	Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.
Integrated Software Management	*…to integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.*
ISM Goal 1	The project's defined software process is a tailored version of the organization's standard software process.
ISM Goal 2	The project is planned and managed according to the project's defined software process.
Software Product Engineering	*… to consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.*
SPE Goal 1	The software engineering tasks are defined, integrated, and consistently performed to produce the software.
SPE Goal 2	Software work products are kept consistent with each other.

Tag	KPA Purpose and Goals
Intergroup Coordination	*… to establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.*
IC Goal 1	The customer's requirements are agreed to by all affected groups.
IC Goal 2	The commitments between the engineering groups are agreed to by the affected groups.
IC Goal 3	The engineering groups identify, track, and resolve intergroup issues.
Peer Reviews	*… to remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of defects that might be prevented.*
PR Goal 1	Peer review activities are planned.
PR Goal 2	Defects in the software work products are identified and removed.
	Maturity Level 4 – Managed
Quantitative Process Management	*… to control the process performance of the software project quantitatively. Software process performance represents the actual results achieved from following a software process.*
QPM Goal 1	The quantitative process management activities are planned.
QPM Goal 2	The process performance of the project's defined software process is controlled quantitatively.
QPM Goal 3	The process capability of the organization's standard software process is known in quantitative terms.
Software Quality Management	*… to develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.*
SQM Goal 1	The project's software quality management activities are planned.
SQM Goal 2	Measurable goals for software product quality and their priorities are defined.
SQM Goal 3	Actual progress toward achieving the quality goals for the software products is quantified and managed.
	Maturity Level 5 – Optimizing
Defect Prevention	*… to identify the cause of defects and prevent them from recurring.*
DP Goal 1	Defect prevention activities are planned.
DP Goal 2	Common causes of defects are sought out and identified.
DP Goal 3	Common causes of defects are prioritized and systematically eliminated.
Technology Change Management	*… to identify new technologies (i.e., tools, methods, and processes) and transition them into the organization in an orderly manner.*
TCM Goal 1	Incorporation of technology changes is planned.
TCM Goal 2	New technologies are evaluated to determine their effect on quality and productivity.
TCM Goal 3	Appropriate new technologies are transferred into normal practice across the organization.
Process Change Management	*… to continually improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.*
PCM Goal 1	Continuous process improvement is planned.
PCM Goal 2	Participation in the organization's software process improvement activities is organization wide.
PCM Goal 3	The organization's standard software process and the projects' defined software processes are improved continuously.

With the exception of Software Subcontract Management, which is not applicable if an organization does not do subcontracting, the key process areas and their goals should be applicable to any software organization. Companies that focus on innovation more than operational excellence may downplay the importance of consistency, predictability, and reliability, but performance excellence is important even in highly innovative environments. It is difficult to identify any goals in Table 1 that will not provide value to an organization, if thoughtfully implemented.

3 EXTREME PROGRAMMING

Extreme Programming is a lightweight (or agile) software methodology (or process) that is usually attributed to Kent Beck, Ron Jeffries, and Ward Cunningham [2, 3, 8]. XP is targeted toward small to medium sized teams building software in the face of vague and/or rapidly changing requirements. XP teams are expected to be co-located, typically with less than ten members.

The critical assumption underlying XP is that the high cost of change has been (or can be) addressed by technologies such as objects/patterns, relational databases, and information hiding. As a consequence of this assumption, the resulting XP process is intended to be highly dynamic. Beck's book is subtitled "embrace change," and the XP team deals with requirements changes throughout an iterative life cycle with short loops. The four basic activities in the XP life cycle are coding, testing, listening, and designing. The dynamism is demonstrated via four values: continual *communication* with the customer and within the team, *simplicity* by always focusing on the minimalist solution, rapid *feedback* via unit and functional testing (among other mechanisms), and the *courage* to deal with problems proactively.

Most of the principles espoused in XP, such as minimalism, simplicity, an evolutionary life cycle, short cycle times, user involvement, good coding standards, and so forth, are commonsense and appropriate practices in any disciplined process. The "extreme" in XP comes from taking common sense practices to extreme levels, as summarized in Table 2. Although some may (improperly) interpret practices such as "focusing on a minimalist solution" as meaning hacking, in reality XP is a highly disciplined process. Simplicity means focusing on the highest priority, most valuable parts of the system as currently identified rather than designing solutions to problems that are not yet needed… and may never be needed as the requirements and operating environment change.

Table 2. The "Extreme" in Extreme Programming.

Common Sense Practice	XP Extreme	XP Implementation
Code reviews	Review code all the time	Pair programming
Testing	Test all the time, even by the customers	Unit testing, Functional testing
Design	Make design part of everybody's daily business	Refactoring
Simplicity	Always leave the system with the simplest design that supports its current functionality	The simplest thing that could possibly work
Architecture	Everybody will work to refine the architecture all the time	Metaphor
Integration testing	Integrate and test several times a day	Continuous integration
Short iterations	Make iterations really, really short -- seconds and minutes and hours, not weeks and months and years	Planning game

XP can be summarized by twelve practices. Although many other practices can be considered part of XP, these twelve are the basic set.

1. *Planning game* -- quickly determine the scope of the next release, combining business priorities and technical estimates. Tthe customer decides scope, priority, and dates from a business perspective, while technical people estimate and track progress.
2. *Small releases* -- put a simple system into production quickly. Release new versions on a very short (two-week) cycle.
3. *Metaphor* -- guides all development with a simple, shared story of how they whole system works.
4. *Simple design* -- designed as simply as possible at any given moment.
5. *Testing* -- continually write unit tests which must run flawlessly; customers write tests to demonstrate functions are finished. "Test then code" means a failed test case is an entry criterion for writing code.
6. *Refactoring* -- restructure the system without changing behavior to remove duplication, improve communication, simplify, or add flexibility.
7. *Pair programming* -- all production code written by two programmers at one machine.
8. *Collective ownership* -- anyone can improve any code anywhere in the system at any time.
9. *Continuous integration* -- integrate and build the system many times a day, every time a task is finished. Continual regression testing means no regressions in functionality as a result of changed requirements.

10. *40-hour week* -- work no more than 40 hours per week as a rule; never work overtime two weeks in a row.
11. *On-site customer* -- real, live user on the team full-time to answer questions.
12. *Coding standards* -- rules emphasizing communication throughout the code.

The planning game and small releases depend on the customer providing a set of "stories," or short descriptions of features, that characterize the work to be performed in each release. Releases are two weeks apart, and the team and customer must come to agreement on which stories (simple use cases) will be implemented within a two-week period. A pool of stories characterizes the full functionality desired by the customer, but only the subset identified as those features most desired by the customer for next two week release are being implemented at any time. New stories can be added to the pool at any time, thus the requirements can be highly volatile, but implementation proceeds in two-week chunks based on the most desired functions currently in the pool, thus the volatility is managed. An on-site customer is needed to support this style of iterative life cycle.

"Metaphor" provides the overarching vision for the project. This could be considered a high-level architecture, but XP emphasizes design while at the same time minimizing design documentation. Some have characterized XP as not allowing documentation outside code [1], but it is probably more accurate to say that, since XP emphasizes continual redesign (via refactoring whenever necessary), there is little value to detailed design documentation… and maintainers rarely trust anything other than the code anyway. Design documentation is typically thrown away after the code is written. The only time design documentation is kept is when the customer can no longer come up with any new stories. Then it is time to put the system in mothballs and write a five to ten page "mothball tour" of the system. A natural corollary of the emphasis on refactoring is to always implement the simplest solution to satisfy the immediate need. Changes in the requirements are likely to supersede "general solutions" anyway.

Pair programming is one of the more controversial practices in XP since it has resource consequences for the managers who decide whether or not the project will use XP. Although it may appear that pair programming will lead to twice the resources, research has shown that pair programming leads to higher quality and decreased cycle time [9]. For a jelled team the effort increase may be as little as 15%, while the reduction in cycle time may be 40-50%. For internet-time environments, the increased speed to market may be well worth the increment in effort. Collaboration improves the problem-solving process, and the increase in quality will also have a significant impact on maintenance costs, which appears likely to more than pay for any added resource costs over the total life cycle.

Collective ownership means that anyone can change any piece of code in the system at any time. The XP emphasis on continuous integration, continual regression testing, and pair programming are intended as protections against problems here.

"Test then code" is the phrase used to express XP's emphasis on testing. It captures the principle that testing should be planned early and test cases developed in parallel with requirements analysis, although the traditional emphasis is on black-box testing. Thinking about testing early in the life cycle is a well-known good software engineering practice, even if too infrequently practiced.

The basic XP management tool is the metric, and the medium of the metric is the "big visible chart." In the XP style, three or four measures are typically all a team can stand at one time, and those should be actively used and visible to the team. "Project velocity," the number of stories of a given size that can be done in an iteration, is one recommended XP metric.

When adopting XP, i.e., the XP attitude toward process improvement, the recommendation is to adopt XP one practice at a time, always addressing the most pressing problem for your team. As one might expect, the XP attitude towards change is that it's "just rules" -- the team can change the rules at any time as long as they agree on how they will assess the effects of the change. The advocates of XP recognize that XP is an intensely social activity, and not everyone can learn it. Having said this, it must also be recognized that XP is a "system" or "methodology" that demonstrates emergent behavior, and to gain the full benefit of XP a reasonably complete set of the basic practices is needed.

4 XP, PROCESS RIGOR, AND THE CMM

The values of XP should be captured in any modern software project, even if the implementation may differ radically in other environments. Communication and simplicity may be stated in other terms (coordination and elegance, for example), but without them non-trivial projects face almost insurmountable odds. The XP principles of communication and simplicity are fundamental process design principles for organizations using the Software CMM also. When defining processes, organizations should capture the minimum essential information needed, use good software design principles (such as information hiding and abstraction) in structuring the definitions, and emphasize usefulness and usability [7]. Rapid feedback is crucial to real-time process control; it has even been captured in previous centuries by aphorisms such as "don't throw good money after bad," and in the quantitative sense can be considered the soul of Level 4. One of the consequences of the Level 1 to 2 culture shift is

demonstrating the courage of our convictions by focusing on realism in our estimates, plans, and commitments.

Much of the formalism that characterizes most CMM-based process improvement is an artifact of large projects and/or severe reliability requirements, especially for life-critical systems. The hierarchical structure of the Software CMM is intended to support a broad range of implementations within the context of the 18 key process areas and 52 goals that comprise the requirements for a fully mature software process.

As a system becomes larger, some XP practices become more difficult to implement. As projects becoming larger, emphasizing a good architectural "philosophy" becomes increasingly critical to project success. Architecture-based design, designing for change, refactoring, and similar design philosophies emphasize the need for dealing with change in a systematic fashion. Variants of these concepts, including architecture-based design and integrated product teams, may be more appropriate in large-project contexts, perhaps in conjunction with XP within teams. In a sense, architectural design that emphasizes flexibility is the goal of any good object-oriented methodology, so XP (with refactoring) and object orientation are well-suited to one another. Multi-discipline teams are also problematic since XP is aimed at software-only projects.

The main objection to using XP for process improvement is that it barely touches the management and organizational issues that the Software CMM emphasizes. Putting in place the kind of highly collaborative environment that XP assumes requires enlightened management and appropriate organizational infrastructure. The argument that process discipline in the CMM sense -- even to the point of a rigorous, statistically stable process -- is antithetical to XP is unconvincing. XP has disciplined processes, and it is apparent that the XP process is a "well-defined" process. CMM and XP can be considered complementary. The Software CMM tells *what* to do in general terms, but does not say *how* to do it, while XP is a set of best practices that contains fairly specific how-to information -- an implementation model -- for a particular kind of environment. XP practices may be compatible with the intent of a practice (or goal or key process area), even if they do not completely address it.

At Level 2, Requirements Management is addressed by stories, on-site customer, and continuous integration. Software Project Planning is addressed by the planning game and small releases. The XP planning strategy embodies Humphrey's proverb, "If you can't plan well, plan often."

Software Project Tracking & Oversight is addressed by the "big visual chart," project velocity, and commitments (stories) for small releases. The commitment process for XP sets clear expectations for both the customer and the XP team at the tactical level and maximizes flexibility at the project's strategic level. The emphasis in XP on 40-hour weeks is a general management concern that is not addressed in the CMM but is considered a best practice. XP also emphasizes open workspaces, a similar "people issue" that is outside the scope of the CMM. Software Subcontract Management is not addressed by XP (and is likely to be not applicable in the target environment).

While an independent SQA group is unlikely to be a part of an XP culture, Software Quality Assurance could be addressed by the culture of pair programming; assuring conformance to coding standards is a typical SQA responsibility that is handled by peer pressure in an XP environment. However implemented, a CMM-based process has mechanisms for objectively verifying adherence to requirements, standards, and procedures. The XP reliance on peer pressure, while effective in most environments, may be vulnerable to external pressures, and this vulnerability should be considered at the organizational level.

Software Configuration Management is partially addressed via collective ownership, small releases, and continuous integration. While not completely and explicitly addressed, configuration management is implicit in these XP practices. Collective ownership may be problematic for large systems, where communication channels need to be more formalized to be effective, and could lead to SCM failures.

At Level 3, Organization Process Focus is addressed at the team level rather than the organizational level, but the philosophy behind adopting XP one practice at a time and "just rules" implies a focus on process issues. Since XP focuses on the software engineering process rather than the organizational infrastructure issues, this and other organization-level processes are areas that needs to be addressed by organizations adopting XP, whether in a CMM-based context or not. Similarly, Organization Process Definition and Training Program are partially addressed by the various books, articles, courses, and Web sites on XP, but organizational assets are outside the scope of XP. As a consequence, Integrated Software Management cannot be addressed since there may not be any organizational assets to tailor.

Software Product Engineering is well-addressed in many ways by the XP methodology with metaphor, simple design, refactoring, the "mothball" tour, coding standards, unit testing, and functional testing. The lack of design documentation would be a concern in many environments, such as hard real-time systems, large systems, or virtual teams. In such environments, good designs are crucial to success, and the refactoring strategy would be high risk. For example, refactoring after a system has been proven to satisfy hard real-time requirements by a technique such as rate monotonic analysis would mean that the analysis would need to be re-done -- the assumption that change

does not have a high cost would be invalid in such an environment.

Intergroup Coordination is addressed by the on-site customer and pair programming. XP's emphasis on communication appears to result in as comprehensive a solution to intergroup coordination as integrated product and process development (and could be judged an effective IPPD approach), although the software-only context ignores multi-discipline environments.

Peer Reviews is addressed by pair programming. Pair programming may be more powerful than peer reviews, in the sense of code-reading and literate programming, although the lack of structure may lessen its effectiveness. The empirical data on pair programming is currently sparse [9] but promising. Contrasting and comparing pair programming and peer review techniques remains an area needing empirical research as a basis for making informed trade-off decisions.

Few of the Level 4 and 5 key process areas are addressed by XP in a rigorous statistical sense, although Defect Prevention may be partially addressed by feedback during rapid cycles. Potential satisfaction of CMM key process areas by XP is summarized in Table 3, at least within the appropriate domain for XP.

Table 3. Satisfaction of Software CMM Key Process Areas by XP.

Level 2 KPAs	Satis-faction	Level 3 KPAs	Satis-faction	High Maturity KPAs	Satis-faction
RM	√ √	OPF	√	QPM	--
SPP	√ √	OPD	√	SQM	--
SPTO	√ √	TP	--		
SSM	--	ISM	--	DP	√
SQA	√	SPE	√ √	TCM	--
SCM	√	IC	√ √	PCM	--
		PR	√ √		

√ partially addressed in XP
√ √ largely addressed in XP (perhaps by inference) (in the appropriate environment)

Many of the key process areas partially covered or not addressed in XP are undoubtedly addressed in real projects. XP cannot survive without management and infrastructure support, even if it is not explicitly called out. It seems fair to say that XP focuses on the technical work, where the CMM focuses on the management work, but a concern with "cultural issues" is evident in both.

5 CONCLUSION

Most of XP consists of good practices that should be thoughtfully considered for any environment. While the merits of any of these practices can be debated in comparison with other ways of dealing with the same issues, none of them should be arbitrarily rejected.

Putting these practices together as a methodology may be a paradigm shift in the same sense that concurrent engineering is. The concepts in concurrent engineering have been around for decades; integrating those concepts as a system results in a paradigm shift in how to build products. In a similar manner, XP provides a systems perspective on programming (if not the only one), just as the Software CMM provides a systems perspective on organizational process improvement. Organizations that want to improve their capability should take advantage of the good ideas in both and exercise common sense in selecting and implementing those ideas.

The Software CMM focuses on the management issues associated with putting effective and efficient processes in place, along with systematic process improvement. XP is a specific set of practices -- a "methodology" -- that is effective within its context of small, co-located teams. Both have good ideas that can be synergistic, particularly in conjunction with other good engineering and management practices. It is questionable whether XP, as published, should be used for life critical or high reliability systems. The lack of design documentation and the de-emphasis on architecture would be judged risky decisions by most knowledgeable professionals, but one of the virtues of XP is that it can be changed / improved for different environments…

The risk in changing XP is that the emergent properties providing value in its proper context may not emerge. Still, the emphasis in choosing and improving software processes should be to let common sense prevail -- and to use data whenever possible to provide insight when answering challenging questions.

ACKNOWLEDGEMENTS

I would like to thank Kent Beck and Laurie Williams for their comments on a draft of this paper. Their contribution is gratefully acknowledged.

REFERENCES

1. Allen, P. XP Explained. *The Cutter Edge* (June 5, 2001).
2. Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, 1999.
3. Beck, K. Embracing Change with Extreme Programming. *IEEE Computer*, 32, 10 (October 1999) 70-77.
4. Johnson, D.L. and Brodman, J.G. Applying CMM Project Planning Practices to Diverse Environments. *IEEE Software*, 17, 4 (July/August 2000) 40-47.
5. Paulk, M.C., B. Curtis, M.B. Chrissis, and C.V. Weber. Capability Maturity Model, Version 1.1. *IEEE Software*, 10, 4 (July 1993) 18-27.

6. Paulk, M.C., B. Curtis, M.B. Chrissis, and C.V. Weber. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley, Reading, MA, 1995.

7. Paulk, M.C. Using the Software CMM With Good Judgment. *ASQ Software Quality Professional*. 1, 3 (June 1999) 19-29.

8. Siddiqi, J. (ed). *eXtreme Programming Pros and Cons: What Questions Remain?* IEEE Computer Society Dynabook (November 2000). Web Site On-line at http://computer.org/seweb/dynabook/Index.htm.

9. Williams, L., R.R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the Case for Pair Programming. *IEEE Software*, 17, 4 (July/August 2000) 19-25.