Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity

Hillel Glazer
Entinex, Inc.©

Many process-oriented software developers (some of whom use the CMM) think of Extreme Programming (XP) as a "seat-of-the-pants" development method. Many high-speed cutting-edge developers (whether they use XP methods or not) see CMM as a cumbersome unnecessary impediment to developing software quickly. This is the result of a myth: Software development speed must be sacrificed when following a process-disciplined approach (such as CMM). This myth is defeated when we look at two realities: The CMM is tailorable, and XP is disciplined. An alternate look at the realities helps open up a new possible approach so that these methods can work together. This article puts forth ideas to bridge the gap between the two sides using the suggested approach, and concludes that process discipline can be achieved without sacrifice to the speed of development.

In the quest for *better, faster, and cheaper,* many companies are wrestling with a methodology problem convinced that their choice is between agile development vs. stable processes. People on both sides of the mat are sure that the two are perpetual adversaries. This article puts an example from each side in the match: Representing agile development will be Extreme Programming (XP); and representing stable processes will be the Capability Maturity Model® (CMM®). It follows a train of thought that seeks to dispel the adversarial myth by looking at its possible origins then building a bridge into reality.

The Myth

The misconception among many commercial software developers is that process discipline in software development (such as the CMM) is incompatible with fast-moving development processes such as XP. A similar misconception among many process-oriented people – CMM or otherwise – is that developing software quickly is tantamount to chaos. If these two views persist, they will keep excellent development teams from realizing the benefits of structured process improvement, and likewise keep larger organizations from looking at alternative development methods. They will be forever locked in a perpetual wrestling match.

Let's face it, whenever someone says CMM, most people think of big, lumbering, cumbersome, bureaucratic paperwork and with good reason. When people think of the places that have typically applied CMM to their organization, or when they look at the place that developed the guide – the Department of Defense (DoD) and their contractors – this could be an apt description, often followed by a sense of dread and loathing.

A quick look at XP may help frame the discussion. From Don Wells' [1] comprehensive Web site on the subject, we can learn that XP has well-defined rules and practices that can be summarized into four main areas: planning, designing, coding, and testing. In the June 2001 issue of CROSSTALK, Leishman [2] discussed the differences between the traditional and the XP development life cycles, while Duncan [3] did an excellent job of expanding on the requirements aspect of software planning. Figure 1, courtesy of Wells, depicts a typical XP project. In Figure 1, readers seeing unfamiliar terms such as *spike* and *system metaphor*, or the unorthodox placement of *acceptance test* or *test scenarios* are encouraged to investigate the referenced materials for more information.

XP is an exercise in iteration. The four XP rules areas are not a sequence for the entire project in one shot through. They contain activities that occur with each iteration. Code is developed by pairs of programmers, tested, and integrated in very small increments. Not only are the requirements gleaned from the user *stories* (much like use cases), but the customer is intimately involved with what and when code is implemented based on the progress of the development and the planning results. Furthermore, XP has rules that govern what *small increment* really means. Planning also includes what many would call project estimating, tracking, and controls, as well as changes to how future XP cycles will apply the experience gained from the previous iterations.

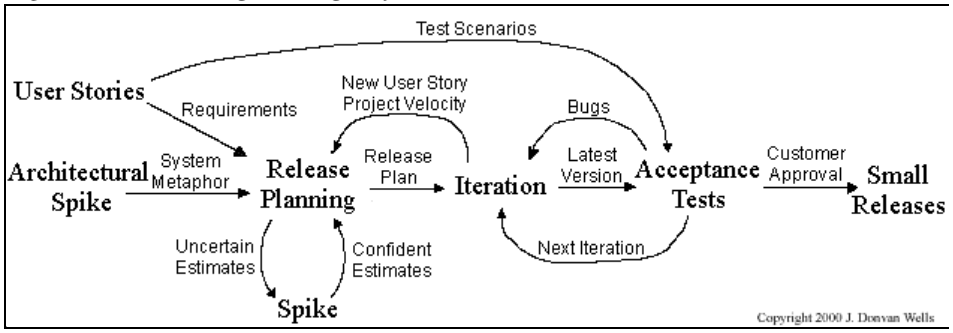
Designing and coding are distinct activities in XP. However, they occur along with testing in very tight yet simple formation. XP does not have a coding standard, except to specify that there must be one. There are several other philosophical and practical aspects to XP based on the XP creator's experience such as when code is to be *reused*, the details of designs, the timing of functionality, the characteristics of the development team, and how to ensure a closed-loop traceability between testing and design.

To be sure, given the compact time frames and immediacy of customer access, one can guess that XP is not intended for large and/or extremely complex projects spread across several locations with no single customer voice. Some say 20 developers would be a big team.

The Reality

The truth is that the CMM – and process

Figure 1: *Extreme Programming Project*



® Capability Maturity Model and CMM *is registered in the U.S. Patent and Trademark Office.*

discipline in general – do not have to send a chill down the developer's spine. One of the most overlooked aspects built right into the CMM is the fact that it is meant to be tailored to the organization. Another significant but often-overlooked facet is the definition of *maturity* as it applies to software development. The definitive text on the CMM, Paulk, et al [4], refers to immature organizations as those whose "processes are generally improvised." And that among immature organizations "even if a software process has been specified, it is not rigorously followed or enforced."

One thing we can see about XP, even from this brief explanation, is that it is not improvised. As any XP developer will tell you, it will not work unless the XP methodology is followed. In fact, looking at what makes a process mature is simply that it is (again, thanks to Paulk) "explicitly defined, managed, measured, controlled, and effective."

Why then is there such an impasse between the goals of agile, creative, and nimble development and the goals of process improvement? Could it all be related back to the same basic sticking points found in most situations? Could it be a simple matter of defining terms and expectations? What if the problem were as simple as that of confusing the *how* of development with the *what* of development? The *pre*scriptive vs. the *de*scriptive.

The Recipe vs. the Menu

A few words may be appropriate here to further explain the previous paragraph. Many standards, frameworks, and methodologies developed by government and industry are very rigorously defined. Like a recipe, they prescribe what to do and how to do it: measure three cups flour, beat in two eggs, grease a 9-inch x 11-inch x 1.5-inch pan, etc. They are full of verbs and adverbs.

On the other hand, a menu describes items that are listed: appetizer: salad or paté … ; soup: cream of mushroom or tomato … ; entrée: baked salmon or roasted chicken … ; dessert: chocolate brownies or vanilla ice cream … Menus are mostly nouns and adjectives.

From a menu you can tell a lot about an establishment. You can tell what their strengths are; you can tell what ingredients they like to use; in most cases you can tell whether you might find something to fit your appetite. Among better establishments, you are likely to find similar characteristics from menu to menu.

The CMM is more like a menu. It does not tell you how to develop software, or how to manage your software development. It simply lists those items found on the menu where good software products are served.

Given the history of most standards, in an industry with more than enough recipes, a menu is a challenging mental switch for some people to make. As a result, many think they are being told to cook, when all they are being told is to dine.

The Suggestion

How does this apply to XP vs. CMM? Let us say that the friction shows up due to a misinterpretation of terms. For example CMM'ers looking at XP see an undisciplined software management and improvement methodology, and XP developers see CMM as a too rigid development methodology. Well, there is the possible source of the answer!

> "If XP is viewed as a development methodology, what is to keep a software management methodology such as the CMM from being there at the same time?"

What if we chose to distinguish XP as a software *development* methodology and CMM as a software *management* methodology? What if XP and CMM were not in any way working at cross purposes. What would we find if we looked closely at the difference between development and management? Could the two be viewed as complementary – even mutually supportive of one another? Would that get us closer to solving the problem? I think so.

To help understand the difference between development and management methodologies, we will look to the hardware world for an example. Hardware can be designed and manufactured in any one of several ways. We will call these the development methodologies. The design can be made on paper or by using computer-aided design (CAD) systems. The manufacturing can also be by hand or can employ any number of automation systems at various steps in the production process. Other aspects of hardware production are the tools and tool control, inspection, inventory control, materials ordering, environmental controls, organizational needs, and so on. These latter aspects can be called management methodologies.

The development and management methodologies, therefore, are distinct disciplines. While the two are not completely decoupled, one does not dictate the other. Obviously the management methodologies must complement and support the development methodologies. They must work together to achieve business goals. A desired state is that they are each optimized to work in the same business and operations strategy models. However, fundamentally, whether you draw design blueprints by hand or by CAD is not dictated by how you control the flow of material through the plant.

The Bridge

In the software world, the CMM does not care what development methodology you use. It does not say that the Waterfall [5] model is better than the Spiral [6] model. Beyond that, it does not even say which life cycle or development models to choose. If XP is viewed as a development methodology, what is to keep a software management methodology such as the CMM from being there at the same time? Taken a step further, if the CMM is viewed as a management methodology for software process improvement, we can completely erase any *forced* divorce between CMM and XP.

In fact, as a development methodology, XP goes a very long way toward having a development team behave as quite a mature software process. Contrary to the perception among many organizations, as a development process, XP can be described as follows:

- Disciplined.
- Not an automatic solution to getting projects done better, faster, cheaper.
- Dependent upon constant communication within the development team and with the customer.
- Packaged to include many of the hard-taught lessons learned from many years of practical development experience.

As a result, the XP development methodology Rules and Practices almost explicitly mirror all but the Subcontract Management, and Quality Assurance CMM Level 2 Key Process Areas (KPAs).

Of these last two, if subcontracting exists on XP projects it would have to be addressed, but if not, then it can be tailored out. The last remaining item is soft-

ware quality assurance (QA). I am sure many readers familiar with XP think I am insane – seeing that with all the testing and iterations in XP, QA was not included among those KPAs satisfied by XP. This is due to another of the misunderstandings in engineering (not just software) that I will briefly address. QA is not quality control (QC).

On the Side

To put it simply, QC is testing. QC is definitely a major player in the XP methodology. QC is a step in any of the development methodologies. In XP, QC shows up all over the place in the coding and testing areas. In fact, QC is built into the planning and designing phases of XP before coding begins. The resulting code (when programming in the XP method) comes from having coded the unit tests first after understanding the component requirements. There is also a lot of QC (we hope) in every development shop. But what QC *is not*, is QA.

What is different between QA and QC? In a nutshell, QA is *process* oriented and QC is *product* oriented. *Testing*, therefore is product oriented and thus is in the QC domain. Testing for quality is not *assuring* quality, it is *controlling* it.

QA makes sure you are doing the right things, the right way. QC makes sure the results of what you've done are what you expected. Some people would prefer we redefine these as *product QA* and *process QA*. An approach that I could endorse, but that is not at issue here.

Nonetheless, while there is a lot of obvious QC in XP – testing – QA is not that far out of the XP Rules and Practices either. There is a lot of wisdom built into the XP Rules and Practices that when followed, are the fixing's for predictably high quality output. In fact, some could argue that preplanning the unit tests and then coding the software is a unique approach to QA. The key ingredients that would add the right QA flavor to XP are management visibility, independent review and audit, and assurance of the application of standards and the development process.

In other words, most XP projects that truly follow the XP Rules and Practices could easily and quickly be assessed at CMM Level 2 if they could demonstrate having a process for the following:

- Ensuring that the XP Rules and Practices are taught to new developers on the project.
- Ensuring that the XP Rules and Practices are followed by everyone.
- Escalating to decision makers when the XP Rules and Practices are not followed and not resolved within the project.
- Measuring the effectiveness of the XP Rules and Practices.
- Providing visibility to management via appropriate metrics from prior project QA experience.
- Knowing when the XP Rules and Practices need to be adjusted.
- Having an independent person doing the above.

The Resolution

If there is one thing developers have likely understood by now, it is that there is no *silver bullet* – either in development or management methodologies. *A* road to better, faster, and cheaper is not *the* road to better, faster, and cheaper. This article in no way suggests that XP is appropriate for all projects, or for any organization, or is without developmental disciplinary shortcomings. I am suggesting that there is a workable solution to organizations pursuing the use of dynamic, highly agile development methodologies (such as XP) within the context of process discipline (such as CMM).

> "If an XP team decided to add quantitative management (perhaps even statistical techniques) to provide more efficient real-time feedback, it could probably achieve Level 4 process capability."

With the understanding that XP is a software *development* methodology and that CMM is a software *management* methodology, and with people who can tell the difference, these two methodologies cannot only co-exist, but they can generate a mutually supportive environment, profitable company, and reliable product.

In this example, by documenting a project's approach to XP, or even closely following one of the many existing documented approaches, then by introducing the QA KPA, projects are already very close to CMM Level 2. With a little more work at the organizational level, CMM Level 3 is not far off. In an article on the Institute of Electrical and Electronics Engineers' computer.org web site Paulk [7] considers "… XP to be another example of a good software process (or philosophy), at least within the proper context, that would satisfy many Software CMM Level 2 and 3 goals." He also adds, "if an XP team decided to add quantitative management (perhaps even statistical techniques) to provide more efficient real-time feedback, it could probably achieve Level 4 process capability."

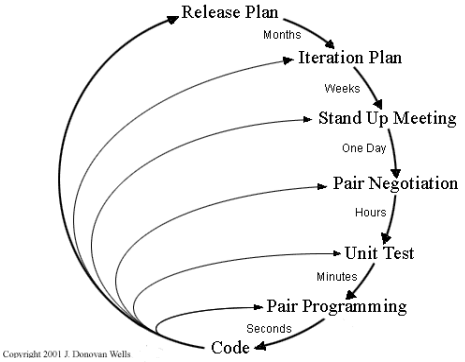


Figure 2*: Release Iteration in XP*

Another Wells' [1] diagram in Figure 2 provides an idea of the detailed development release process. The macro-cycle depicted here offers insight into opportunities for several CMM KPAs such as project planning, project tracking and oversight, organization process definition, intergroup coordination, and others.

Ron Jeffries [8] wrote a quick article outlining the activities performed on a project that used XP and how they related to the CMM's upper four levels. While a very cursory sample, it does convey that there is some interest in seeing the two methodologies work together.

While the list of projects experimenting with or transitioning to XP is as dynamic as the methodology itself, DoD projects are likely to be few among them. As mentioned, XP is intended for small teams of programmers. If DoD projects can be broken down into smaller projects and integration of these components can be tightly managed, then perhaps even these projects can try XP.

At publication, this author personally only knew of one organization giving serious thought to developing software using XP within the CMM process framework. However, although specifics of the effort were proprietary, the prognosis of successfully pinning the myth to the mat is very positive.

The Conclusion

There are many corporate and technical leaders looking to find effective paths

toward better, faster, and cheaper. There has long been the perception that while CMM managed organizations may achieve the *better*, the jury is still out on *cheaper*, and *faster* is clearly not readily evident. Especially when going from Level 1 to Level 2.

Projects thinking of using XP in organizations already assessed against the CMM are encouraged to shed the myth that they could *lose* their CMM rating. Organizations that use XP on their projects wanting to fulfill the intent of the CMM's KPAs are encouraged to shed the myth that they will be *bogged down* with the burden of dead trees. I posit that a symbiotic relationship exists to be found between the speed of agile development methodologies such as XP, and the direction of process improvement management methodologies such as CMM.

The first step is to understand why your processes do or do not fulfill the intent of CMM. Then plot the path of how to make your processes what you need them to be. The intent of the CMM is what you need to demonstrate. If you are effectively using a development methodology like XP, you are already nearly there. All you need to do is prove it.

One path to better, faster, and cheaper can be found outside the development myth in the peaceful coexistence of agile programming and structured processes and process improvement.◆

References

1. Wells, J. Donovan. "Extreme Programming: A Gentle Introduction." <www.ExtremeProgramming.org>.
2. Leishman, Theron. "Extreme Methodologies for an Extreme World." CROSSTALK, June 2001: 15-18.
3. Duncan, Richard. "The Quality of Requirements in Extreme Programming." CROSSTALK, June 2001: 19-22, 31.
4. Paulk, Mark C., Charles V. Weber, Bill Curtis, and Mary Beth Chrissis, eds. The Capability Maturity Model: Guidelines for Improving the Software Process. Software Engineering Institute. Addison-Wesley Longman, 1994.
5. Royce, W. W. "Managing the Development of Large Software Systems." Proceedings of IEEE WESCON. Aug. 1970.
6. Boehm, Barry. "A Spiral Model of Software Development and Enhancement." ACM SIGSOFT Software Engineering Notes. Aug. 1986.
7. Paulk, Mark. "XP from a CMM Perspective." IEEE Computer Society. Dynabook, 2001. <www.computer.org/seweb/Dynabook/PaulkCom.htm>.
8. Jeffries, Ron. "Extreme Programming and the Capability Maturity Model." 1 Jan. 2000. <www.xprogramming.com /xpm ag/xp_and_cmm.htm>.

About the Author

Hillel Glazer is the principal consultant of Entinex, Inc. He brings a broad spectrum of experience in process engineering and management. He is a student of the evolution of process-centered design, development and production and has followed the progress of Total Quality Management, Integrated Product and Process Development, ISO 9000, and the Capability Maturity Model from their emergence and introduction at the Department of Defense to their subsequent migration to the private sector. The focus of his career is on the issues of product integrity and technology management. He specializes in the management-driven engineering principles of quality, operations, risk, requirements, productibility, configuration, and project management. In merging these disciplines with business and operations strategies he emphasizes the importance of thoroughly planned and integrated process management. He has successfully adapted and evolved these disciplines across the Internet, software, and manufacturing industries.

Entinex, Inc.
1516 Castle Cliff Place
Silver Spring, MD 20904
Phone: (301) 384-4203
Fax: (240) 465-0062
E-mail: hillel@entinex.com

WEB SITES

Object Management Group

www.omg.org

The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Its membership roster, about 800 strong, includes virtually every large company in the computer industry, and hundreds of smaller ones. OMG's best-known specifications include CORBA, OMG IDL, IIOP, the OMA, and Domain Facilities in industries such as healthcare, manufacturing, telecommunications, and many others, UML, the MOF, and CWM. All of OMG's specifications may be downloaded without charge.

Distributed Objects & Components

www.cetus-links.org/oo_distributed_objects.html

This site of general information on distributed objects and components is part of the Cetus Links network. Cetus Links offers quick access and a comprehensive overview of tens of thousands of interesting pages about object-orientation and component-orientation that exist on the Internet. The Cetus Links can be regarded as an index to Internet addresses (http, ftp, and mailto) about object-orientation and component-orientation.

Distributed Object Computing with CORBA Middleware

www.cs.wustl.edu/~schmidt/corba.html

This site features mini-tutorials, including an overview of CORBA, research, on-line specification, related papers, tools, the ACE ORB (TAO), and CUJ and C++ report columns. It also describes the contents of the series of C++ Network Programming books written by Douglas C. Schmidt and Steve Huston.

IEEE Computer Society

http://computer.org

With more than 100,000 members, the Institute of Electrical and Electronics Engineers (IEEE) Computer Society claims to be the world's leading organization of computer professionals. Founded in 1946, it is the largest of the 36 societies of the IEEE. The society is dedicated to advancing the theory, practice, and application of computer and information processing technology. The site features listings of conferences, journals, technical committees, standards working groups, and more, to promote an active exchange of information, ideas and technological innovation among its members.