Extreme Programming Practices Used to Facilitate Effective Project Management

Clement James Goebel III, PMP, Partner, Menlo Innovations
jgoebel@menloinnovations.com

Introduction

It is unfortunate that many software development professionals regard project management as formalized paper pushing. It is even more unfortunate when a project manager focuses primarily on the scheduling of meetings, and the creation or maintenance of artifacts instead of fostering high quality communications and coordination between project stakeholders.

Extreme Programming codifies a set of practices that many software developers are willing to adopt in both action and spirit. Many of these practices are grounded in fundamental project management theory. When software development teams embrace the practices of Extreme Programming an opportunity is created for a broad set of project management practices to become meaningful and accessible to the developers, while at the same time making clear, unambiguous information available to the project managers.

This paper describes the practices of Extreme Programming from the viewpoint of project management. However, it is important to acknowledge that Extreme Programming is not a comprehensive project management system, but rather is a set of software development best practices that overlap nicely with best practices from the project management domain.

Risk Management

Software development projects represent an investment of resources by the project's sponsor, an investment that often yields little or no return. The Standish Group's Chaos Report 1994 states that fewer than 10% of software projects in large companies were successful. Medium sized companies do better with 16% of their software projects being successful, and small companies succeed on 28% of their software projects (Standish 1994). Given these statistics it is worthwhile to invest significant effort in Risk Management for software projects. "Research at The Standish Group also indicates that smaller time frames, with delivery of software components early and often, will increase the success rate." (Standish 1994)

Small Releases

Extreme Programming acts on the observation that smaller projects have a higher success rate. It decomposes all software projects into multiple small releases where each release of the software contains only a subset of the required functionality. These Small Releases are incremental production versions of the project's expected final deliverable providing limited subsets of functionality to the system's users. Each release of additional functionality offers stakeholders an opportunity to use the evolving capabilities of the software and provide high quality feedback, thereby improving the quality of progressive elaboration.

The targeted time between releases is two to eight weeks, with a strong prejudice for the smallest possible time period. While it can often be difficult to identify suitable functionality for early releases, the importance of early releases cannot be overstated. Projects that use small incremental releases benefit from user feedback and well understood project status, and they produce a return-on-investment before the project is fully completed.

Early releases that fail to produce business value provide an early warning sign of a project in trouble. Unlike milestones such as Preliminary Design, whose completion will often simply be declared, these early releases require demonstrable software product. Early releases that do not work provide important information, while there is still budget and schedule to react to the detected problems. Another interesting scenario to consider is a project that is cancelled halfway through due to constraints external to the project. In this case, a team that has produced several incremental releases will potentially have delivered something that will return value to the organization despite being cancelled before completion.

Other Extreme Programming practices also address risks common to software development projects. For example, the risk associated with losing a key technical resource is greatly reduced by the practices of Collective Ownership, and Pair Programming. While both of these practices apply to Risk Management, they are also key practices in Human Resources Management and described later under that heading.

Integration Management

Extreme Programming offers nothing to help integrate the efforts of non-software developers. Unfortunately, some advocates of Extreme Programming suggest that the efforts of technical writers, database managers, and quality assurance specialist are not required. In reality, while Extreme Programming does not explicitly describe how to integrate the work of others, the practices do not preclude the ability to integrate with other efforts. Small Releases make Integration Management a more continuous process in contrast to processes that place deployment, documentation, and testing at the end of the schedule.

Continuous Integration

At a more tactical level, the Extreme Programming practice of Continuous Integration requires that the work of software developers be integrated on a daily basis. While this practice can cause additional overhead for individual developers, it allows the team to identify problems daily that would otherwise become undiscovered rework accumulating until all developers integrate their individual work products.

Scope Management & Time Management

Ask most software development teams for a copy of their project plan and you will receive an activity list formatted as a Gantt chart. Many times these activity lists will describe several phases of activities such as Analysis, Design, Construction, and Testing. Areas of functionality will be broken out under these headings in order to assign them to specific programmers, but seldom are the assignments identified in the Gantt chart clearly traceable back to a Requirements or other specification documents. All too often, the missing item that would help a team improve their planning practices is a well-constructed Work Breakdown Structure. Extreme Programming focuses almost all of its planning efforts on building a thoughtful Work Breakdown Structure and its constituent Work Packages. Extreme Programming does not teach Work Breakdown Structures and Work Packages explicitly, however, careful study of the Story Cards used in Extreme Programming reveals that they are almost identical to Work Packages in their key attributes.

Properties of	
Work Packages	**Story Cards**
Are deliverable-oriented.	Describe something of value to the user.
Usually contains no more than eighty hours of effort to complete.	Need to be of a size that you can build a few of them every two weeks.
Are independent.	Should be independent of each other.
Are testable.	Can be tested.
Can be broken into activity lists.	Are broken down into tasks.

Exhibit 1 (Project Management Institute 1998) (Beck 2001)

Planning Game

Extreme Programming teams capture Work Packages as hand-written descriptions of user functionality often written on an index card. These index cards are called Story Cards to remind project participants that the contents of each card should describe a story that a user might tell about functionality instead of the technical activities that software developers instinctively describe. These Story Cards, or Work Packages, are used to facilitate estimates, build schedules, authorize work, drive testing, and report status.

Wedding.com >> Summary Report

Create a report that lists the number of guests invited, the number of guests that have RSVP’d in the affirmative, the number of guests that have RSVP’d regrets, and the number of invitations that as yet have not RSVP’d.

Show the total budget for the wedding, the funds already spent, the funds required to pay for items already ordered or booked, and the budget amount still remaining.

Also show the date the report was printed and the number of days until the wedding.

Exhibit 2 – Sample Story Card

Work authorizations are made every iteration, typically every two weeks. At the beginning of each iteration the developers estimate each Story Card as a team. Any Story Card that is estimated to exceed the length of the iteration is then decomposed into two or more new Story Cards that as a set represent the same functionality as the original. These new Story Cards, or Work Packages, are then estimated.

Estimated stories are then grouped into releases of useful functionality that could add some measure of value to the user, or at least help reduce the risk of the project. Each release is then broken down into two-week development iterations, and stories are again assigned to the iterations using a business value perspective and the cost estimates instead of a development efficiency strategy.

Once the remaining project schedule has been updated, the next two-week development iteration begins. The developers discuss as a group how to break this iteration’s stories into simpler developer tasks, or activities. The team members distribute the tasks amongst themselves and begin development. During the iteration all stories and tasks that are completed are marked as such in a publicly viewable manner. Software developers are only allowed to work on authorized stories, and are not allowed to go around the work authorization process to do a little extra work now to make something easier later just because it is on the long-term schedule. If they want to change the development strategy then story cards must be rewritten, re-estimated and rescheduled, effectively enforcing a lightweight but process-oriented change control process. Development continues for exactly two weeks, then the next planning cycle begins again even if all the work scheduled has not yet been completed.

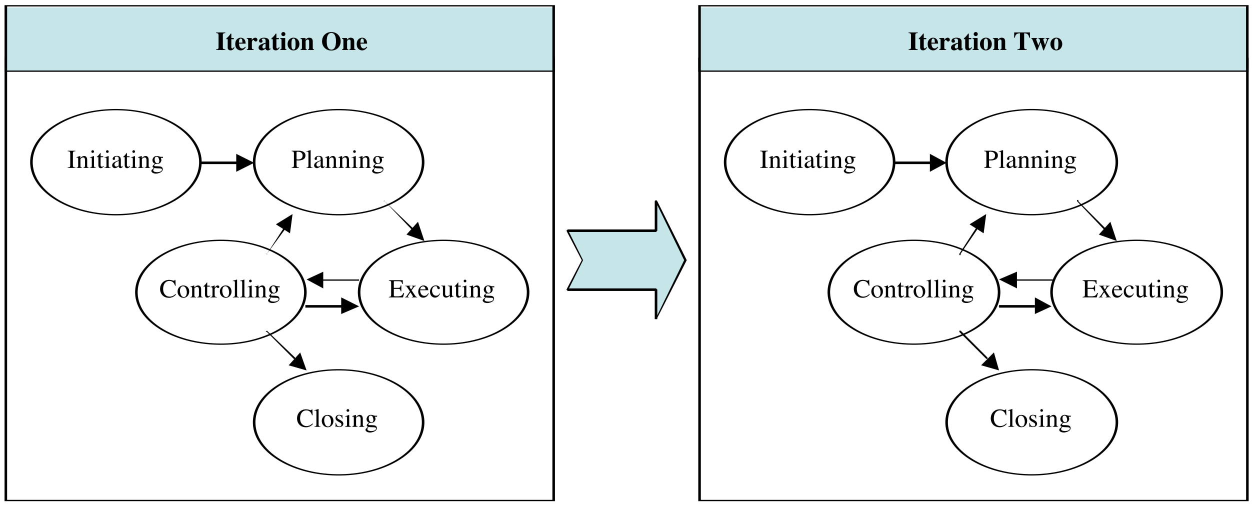

Exhibit 3 (Project Management Institute 2000)

Metrics are computed for each iteration. The metrics taught by Extreme Programming practitioners include Velocity and Yesterday’s Weather. Both of these metrics are simplified versions of Schedule Performance Index, and could easily be augmented with this standard measurement. Next, new story cards are written for any newly discovered functionality, and existing cards might be rewritten to reflect an improved understanding of the desired/required scope. The continual updating of story cards as the team's knowledge improves is a process acknowledgement of the progressive elaboration that affects all nontrivial projects. All new stories are estimated, and any stories that have not been completed (including those not started) have their estimates updated. Release plans and iteration plans are adjusted if necessary. However, instead of slipping delivery dates, Extreme Programming favors floating scope in order to release partial functionality.

The activities for each iteration follow the expected pattern of process flows as described within the Project Management Body Of Knowledge, as seen in Exhibit 3. In between the execution phases of the two iterations there is no code being written and all private code branches have been merged. Each iteration is run like its own small project in order to reduce any risk associated with unintegrated work product.

Human Resources Management

Often one of the most challenging aspects of project management is managing human resources. For software development projects in particular this includes the complex juggling of technical tasks between individual software developers who have different individual skills, effectively treating each developer's assigned tasks as an independent subproject. This type of project plan often suffers from key resource bottlenecks and status meetings reduced to determining which individuals are falling furthest behind. Extreme Programming addresses this head-on by eliminating the dependency on individual developers. Work Packages are scheduled and authorized based on the needs of the business and the users not the needs of the software developers. All developers are cross-trained to work in all areas of the code base. Developers broaden their skills, and project managers stop worrying about keeping individual software developers for the entire duration of the project. The process maintains knowledge of the full code base in the team, not in individuals.

Collective Ownership

When *nobody* owns the code being written, anarchy results as multiple individuals modify any and all code to suit their own needs. The "obvious" correction to this problem is to assign distinct sections of code to individual developers. Unfortunately this creates new problems. First of all the team quickly begins to require that all scheduling be done on a developer by developer basis, effectively working to keep developers busy instead of assembling useful subsets of functionality. The second challenge in individual ownership is that everyone knows what work they will be assigned. This causes odd dynamics in design discussions as developers subconsciously, or consciously, move complexity from one subsystem to another based on where they will be assigned to work.

Extreme Programming uses a model of Collective Ownership; the code is still owned but collectively. Anybody on the team who sees something that should be fixed in the code is required to attend to the task immediately. Collective ownership is very important to keeping the scheduling process focused on value delivered instead of resource availability. It also provides individual developers with the opportunity to grow in areas where their skills are weak. Collective ownership can be a difficult culture to build, and that is why Extreme Programming has a strong set of practices to reinforce good communications and teamwork. However, collective ownership of the code clearly wins out over other models when a developer leaves the team unexpectedly.

Sustainable Effort

Within teams that use Extreme Programming it is recognized that from time to time extra effort will be required to complete releases and other important milestones. However, the practice of Sustainable Effort is an important acknowledgement that teams regularly burning the midnight oil time and time again simply reduce their overall effectiveness beyond any gains made by working longer hours. Not only does productivity fall, but extended periods of extra effort can negatively affect team morale. Therefore, Extreme Programming teams avoid extraordinary efforts in any two consecutive iterations. Instead the scope should be reduced to ensure that quality does not suffer.

Quality Management

As programmers move from work authorization to work authorization, and often from one area of the code to another, it is easy to see that maintaining quality in the work product could be challenging. Extreme Programming requires a very disciplined design approach to allow freedom in assigning resources while maintaining high quality.

Simple Design & Refactoring

The first practice that supports quality workmanship is a focus on simple design. Instead of trying to make all code modules very flexible in anticipation of future requirements, Extreme Programming focuses on building reliable and simple code to satisfy the requirements of the work packages that have already been authorized. As new features are added, or stories are authorized, the design is reworked and modified to accommodate the most recently scheduled features. This reworking of the design is done one step at a time, or what Extreme Programmers call Refactoring. After each incremental step of refactoring is completed, the entire application is run through quality control. For example, one incremental refactoring would be renaming a single method or function. After renaming the function the developers validate that the quality control tests for the entire application pass successfully before moving forward with the next incremental refactoring. There will typically be hundreds or thousands of these quality control tests to run, which leads to the need to automate those quality control tests.

Testing

In order to satisfy the requirement for 100% of all quality control tests to pass after each incremental coding step, the need for those tests to run in an automated test harness becomes a necessity. Each and every public method or function must have a functional specification captured in the form of an automated test suite. These automated unit tests are considered to be such an important part of the design process, they are required to be written before the code they are intended to test is written. The power of these low level specifications expressed in executable code is that the computer can be used to verify conformance. It would not be unusual for a member of an Extreme Programming team to run the entire suite of all unit tests dozens of times per hour. The safety net of this executable specification further enhances the team's ability to shift resources from one part of the application to another and enhances their ability to perform collective code ownership.

Pair Programming

The rigor of simple design and 100% test coverage can be difficult to adopt and to maintain. There is also a significant amount of learning that must occur when all programmers are expected to be familiar with all of the code. Extreme Programming offers an unparalleled support network for all of the developers on the team. Whenever production code is being written there must be two developers sitting at a single keyboard and monitor. Having two programmers working together when coding offers several advantages. It reinforces collective ownership in a manner that code reviews by your peers will likely never achieve. Both partners typically learn from each other, and this reinforces team spirit. Likewise each partner feels a need to not let their partner down, so extra effort is often focused on following all of the practices as adopted by the team (DeGraff 2002, 120). Compare this dynamic to typical situations where testing and quality practices are often sacrificed in the later stages of a project in order to maintain schedule.

Pair programming also facilitates communications within the team. Partners change throughout the day; this can provide a fresh perspective and keeps information traveling throughout the team. Developers working an Extreme Programming team typically report a newfound understanding of the overall project as well as a deeper understanding of their teammates' points of view. The effect of this improved alignment in understanding cannot be understated in regards to the impact on quality.

Communications Management

When a project manager mentions the need for improved communications on a project, software developers often immediately envision an increased number of meetings and documents. While formal meetings and written documents have their place in a communication plan there are many other tools for facilitation of communication between project participants. The Extreme Programming practices include several simple practices intended to enhance communications.

Standup Meeting

Ironically, Extreme Programming does away with the weekly status reports and weekly status meetings in part by having a status meeting every day. This meeting uses techniques that everyone has heard of and joked about, but seldom have actually tried to implement. For this daily meeting everyone stands up in a circle, hence the name Standup Meeting. Each person speaks in turn and is seldom interrupted. Each party announces which Story Card they are currently working on and then describes any problem that they are having for which they might need a fresh perspective. If someone else has an answer they simply state that they can help and the required parties discuss the problem after the standup meeting has been completed. Some teams reinforce the process of discouraging speaking out of turn by passing around a speaking token, such as a book or other more creative objects such as a ten pound weight. Most of these meetings are completed in less than fifteen minutes.

Common Workspace

The amount of time that teams expend assembling for meetings can drain untold hours by the end of a significant project. Extreme Programming's Standup Meeting could exacerbate this problem, however, the practice of all project participants working in a Common Workspace makes assembling for the Standup Meeting as simple as standing up and forming a circle. Because being in the same room makes it easy to meet with others on the project many problems that would normally initiate an email exchange or the scheduling of a meeting are solved by developers simply pushing their chairs together for a few minutes or asking questions of each other without even moving their chairs.

Onsite Customer

Not only do the software developers work in the Common Workspace but the Extreme Programming practices recognize the need for someone who can answer the software developer's questions. In the ideal world described by the initial texts on Extreme Programming, this would be a representative of the user community also known as the Onsite Customer. Updated interpretations of this practice might replace the sample user of the system with someone who is charged with representing the needs of the users in selecting the priority of the work packages and answering developer questions. However, the key requirement to this practice is that the developers need easy access to someone who is capable and willing to make decisions when questions arise. Answers from the Onsite Customer that change scope significantly must be written as new Story Cards, estimated and then scheduled, thereby maintaining the scope management process.

Metaphor

One final communication practice that helps to facilitate communications between the developers and the Onsite Customer is the Metaphor. The Metaphor practice is probably one of the most difficult practices to master in Extreme Programming. It requires the high-level system design to be done using a metaphorical model that will be reflected in the code and be understood by the Onsite Customer. By modeling the overall system architecture on another commonly understood process, or objects that can be explored in the real world, discussions between software developers are more likely to be somewhat accessible to the Onsite Customer.

Why is Extreme Programming Different

Extreme Programming accepts that humans are fallible and builds a process that not only accepts progressive elaboration, but makes this reality a central theme to all of its other practices. There is also the recognition that following the proscribed practices in the real world everyday can be very challenging. To overcome this difficulty the practices interlock and compliment each other.

Limitations to Extreme Programming

Not all projects would be well served to adopt Extreme Programming. But while the practices build on each other and are more powerful when assembled in the whole, the set of practices were assembled based upon their historical use within successful projects. Therefore, there is value in understanding the individual practices and how they might have a successful impact on projects that do not fully adopt Extreme Programming.

In some teams the partial adoption of the Extreme Programming practices have been used as an excuse to stop doing project management. Software developers self-selecting the practices of interest does not eliminate the need for project management. Nor does the full adoption of the practices and the culture eliminate the need for project management, instead the full adoption of the practices puts in place a core set of project management pieces that will inform a project manager and help them to steer a project to success.

My Experiences

As a certified Project Management Professional my view on these practices are somewhat different than those of the typical software developer. However, the thoughts expressed here are more than an academic opinion about practices that I have studied. I have personally worked with teams from over a dozen organizations that have adopted some portion of these practices. These organizations range from small product firms, to governmental, to large global companies. In each case the improved ability to estimate projects, track progress, and to build software that provides value to the end users have been surprising given the low tech tools used to facilitate high quality project management. (Baer 2003, 125) Successful process change always requires some cultural change as well, and failure to recognize this fact is where many change efforts fail. The practices of Common Workspace and Pair Programming reinforce new social norms within the team in a way that dramatically increases the likelihood of successful change adoption.

Conclusions

Within the community of software development professionals there is a growing movement of experienced practitioners leading a revolt against weekly status meetings, comprehensive specification documents, code reviews, and Gantt charts. When first introduced to these ideas it can be easy to dismiss such advocacy as the promotion of chaos that would leave software developers free to pursue their own goals without accountability to the business. Indeed, such chaos is a very likely outcome of removing management constraints from a software development team. But what would happen if you could implement key project management practices in such a way that software developers were left feeling freed from the tyranny of paper pushing and meetings, while at the same time you improved scope control, task estimation, project status reporting, quality control, and team morale?

Imagine an environment where software developers openly collaborate with other members of the team to solve problems quickly and effectively, produce results in a predictable manner, and embrace project management. Extreme Programming is socially acceptable to many software development teams, while providing practices grounded in fundamental project management theory.

References

Baer, M. (2003) The New X-Men. *Wired,* (November), 125-129.

Beck K. (2001) *Planning Extreme Programming.* Addison-Wesley:Upper Saddle River.

DeGraff, J. & Lawrence, K. A. (2002) *Creativity at Work.* Jossey-Bass:San Francisco.

Project Management Institute. (2000) *A guide to the project management body of knowledge (PMBOK®)* (2000 ed.). Newtown Square, PA: Project Management Institute.

Project Management Institute. (1998) *PMBOK Q&A*. Newtown Square, PA: Project Management Institute.

Standish Group (1994) *The CHAOS Report 1994.* http://www.standishgroup.com/sample_research/chaos_1994_1.php.