

Enabling Software Quality with Extreme Programming

Presenters

Dave Rooney

Principal Consultant
Mayford Technologies Inc.
dave.rooney@mayford.ca
<http://www.mayford.ca>

Dave Rooney has been using Extreme Programming since 2000 with great success. Dave has been developing business applications for the past 15 years with Java/J2EE, C/C++, and PowerBuilder, and was initially drawn to XP by its focus on delivering business value to the customer. He quickly realized its potential for reducing or eliminating many of the obstructions in place in many software projects and appreciated XP's emphasis on creating well-tested, well-written code.

Nicole Martin

National Manager IRRS and IMRS, Business Information Solutions
Real Property Services
Public Works and Government Services Canada
nicole.martin@pwgsc.gc.ca

Nicole Martin has been an employee of Public Works and Government Services Canada for 16 years, the last 11 in project management for national systems. Dave Rooney initially introduced her to Extreme Programming in October 2001. She found that the discipline and the practices brought to the project by XP increased the service and the satisfaction to the client as well as improving the quality of the end product. In addition, XP practices consolidated the spirit of the teamwork thus facilitated the implementation of the project.

Enabling Software Quality with Extreme Programming

Abstract	4
1 Introduction	5
2 Background	6
2.1 Four Variables of Project Management	6
2.2 Extreme Programming Values	6
2.2.1 Communication	6
2.2.2 Simplicity	6
2.2.3 Feedback	7
2.2.4 Courage	7
2.3 Key Practices	7
2.3.1 The Planning Process	7
2.3.2 Frequent, Small Releases	8
2.3.3 Metaphor	8
2.3.4 Simple Design	8
2.3.5 Testing	8
2.3.6 Refactoring	8
2.3.7 Pair Programming	8
2.3.8 Collective Ownership	8
2.3.9 Continuous Integration	9
2.3.10 Sustainable Pace	9
2.3.11 On-site Customer	9
2.3.12 Coding Standard	9
3 Extreme Programming and Software Quality	10
3.1 The On-site Customer	10
3.2 Focus on Testing	10
3.2.1 Acceptance Testing	10
3.2.2 Unit Testing	10
3.2.3 Test-Driven Development	11
3.2.4 Pair Programming	11
3.3 Iterative Development	11

Enabling Software Quality with Extreme Programming

4	Case Study	13
4.1	IRRS – The Information Repository and Reporting System	13
4.1.1	Previous projects	13
4.1.2	Methodology	13
4.2	The Client.....	13
4.2.1	Situation	13
4.2.2	Project Team	13
4.2.3	Methodology	14
4.2.4	Why use the practices?.....	14
4.3	Benefits	15
4.3.1	The Team	15
4.3.2	Individual	15
4.3.3	Other teams	15
4.3.4	Managers.....	16
5	Conclusion	17
6	References.....	18

Abstract

This paper provides a brief overview of the practices of Extreme Programming, and details their effect on software quality at all stages of planning, development and testing. Extreme Programming (XP), despite the connotations of its name, is a disciplined approach to developing and delivering high quality software. It consists of a set of best practices observed over many years of software development, and assembled into a process that is greater than the sum of its parts.

The quality of the software produced is considered to be the highest priority of an XP development team. The quality of the code is maximized through test driven development and unit testing. The quality of the overall system is maximized through continuous integration. The quality of the system from the business perspective is maximized through iterative development and frequent small releases, which allows for functional testing at much smaller intervals. All of these aspects of XP provide feedback in very short cycles to the development team and business clients, providing a clear picture of the progress of the system's development and allowing for adaptation to changes in the business environment.

The paper will include a case study of the use of Extreme Programming for the development of a system at a large federal government department, and how the XP practices improved the overall quality of the application and allowed for changes in the business.

The expectation of all Business Managers is to deliver a final product on time, on budget, at minimal risks to meet all of the business community requirements using the internal quality standards that satisfied all stakeholders. The flexibility of Extreme Programming allows reviewing the plan during the different iterations, rapidly change existing requirements, integrate new requirements, answer to priority changes and ultimately remove requirements if necessary. The major output of the practices is the highest quality of the software product delivered to the client that satisfy the developers, the business analysts and the end users.

All the team members adopted Extreme Programming quickly. As soon as we started to include all the team members to the status meeting, the Leaders realized that this method would help increasing the motivation of the members. Being involved at all the stages of the development cycle gave them confidence and secure them about their role as well as made the process less stressful. In fact, the XP practices increased the productivity of the team, allowed testing at all time during the cycle, increased communication between the team members, increased the spirit teamwork, permitted the rollout of a full package without bug fixes at a later time and certainly contributed to the great success of the project.

1 Introduction

Quality in a software product can have differing meanings depending on the perspective of the individual. The Institute of Electrical and Electronic Engineers (IEEE) defines quality as "the degree to which a system, component or process meets customer or user needs or expectations" (IEEE90).

For the purpose of this paper the IEEE definition will be used, though specifically in the context of a software system.

Quality is an attribute of a system that walks hand in hand with risk. The more risks that are identified and mitigated, the higher the quality will be. Similarly, the sooner that the risks are identified and mitigated, the higher the quality will be.

These ideas are not new, and indeed have been part of project management best practices for many years. What is new is the level to which Extreme Programming elevates the practices, and thus the quality of the products delivered. The practices themselves are relatively simple in nature, but are mutually beneficial to create a methodology that is greater than the sum of its parts.

2 Background

Extreme Programming (XP) is a methodology for building software based on longstanding industry best practices. These practices include evolutionary prototyping, short release cycles, and active end-user involvement in requirements definition and acceptance testing. XP is based on the work of Kent Beck, Ward Cunningham, Ron Jeffries and others, and was used initially in 1996 at the Chrysler Corp. (now DaimlerChrysler) for a project to re-engineer their compensation system. The specific practices were chosen based on the belief that they are mutually supportive.

2.1 Four Variables of Project Management

Variable	Traditional Projects	XP Projects
Scope	Fixed	Variable
Time	Fixed	Fixed
Cost	Fixed	Fixed
Quality	Variable	Fixed

On a project using a traditional methodology, Scope, Time and Cost are fixed. As a result, the Quality of the system is variable. If the Time estimates weren't accurate, or external factors such as staff turnover weren't taken into account, the Quality suffers as the staff struggles to meet the release deadline. Additionally, there is little leeway to accommodate changing business requirements, or requirements that were not clearly defined in the first place.

An XP project makes the Scope variable, and Time, Quality and Cost fixed. This approach allows adaptation to any changes to the requirements. The Customer, as owner of the requirements and their priority, is able to make the decisions on how to adjust the Scope in order to keep the Time and Cost fixed. As a result, the team is not working madly to meet a deadline, and the Quality is improved.

2.2 Extreme Programming Values

XP is based on 4 core values:

2.2.1 Communication

- A Customer Representative is co-located with developers
- Conversations are preferred to formal documentation, although documentation can be used to record the conversations
- Collective ownership of code - anyone has the right to change anything
- Pair programming

2.2.2 Simplicity

- "Build the simplest thing that could possibly work."
- Only code what is needed now; don't code for tomorrow's requirements, since they might change

Enabling Software Quality with Extreme Programming

- Write code that clearly shows its “intent”,

2.2.3 Feedback

- Minutes to hours: Programmers write unit tests - even before writing the code.
- Hours to days: Programmers estimate development time whenever the Customer writes a new requirement "story"
- Weeks to months: Customers write acceptance tests for each iteration and release.

2.2.4 Courage

- Programmers are encouraged to rewrite code and refactor designs.
- Unit tests allow Programmers to quickly verify that changes have not affected the system
- Communication and simplicity support courage: dissatisfaction is quickly communicated through the team, and simple designs are easier to modify.

2.3 Key Practices

XP is based on 12 key practices. None of these practices by themselves are new, indeed most have been used by software developers for decades. The difference with XP is that it combines all of the practices into a single methodology in which they complement and support one another.

Before proceeding, bear in mind that any team's software development methodology needs to be customized to the team and their circumstances. No methodology is just a collection of rules to be performed robotically, and XP is no exception.

2.3.1 The Planning Process

The planning process allows the customer to define the business value of desired features. Cost estimates provided by the programmers are used to choose what needs to be done and what needs to be deferred. The Customer determines the features that are the highest priority, while the Programmers make the estimates for how long it will take to implement them.

The features, known as Stories, are then selected for a Release which is on the scale of 2-3 months. Within the Release, the Stories are further separated into Iterations of 1-2 weeks in length. The Stories that deliver the highest business value are implemented first.

The Customer has the right to request changes to the Stories, add or remove Stories or to rearrange their order of implementation. The Programmers, meanwhile, estimate the time that it will take to accommodate the changes.

The effect of this process is that the project is “steered” by the Customer, similar to the act of steering a car. Rather than making abrupt, large changes, the Customer and Programmers are constantly making small changes. This allows them to “stay on the road”, to extend the metaphor.

Enabling Software Quality with Extreme Programming

2.3.2 Frequent, Small Releases

The Programmers get a simple system working early, and update it frequently on a very short cycle. This allows incremental testing by the Customer, identifying any problems from the business perspective as early as possible.

2.3.3 Metaphor

The team uses a common "system of names" and a common system description that guides development and communication. For example, a team developing a payroll application may use the metaphor of an assembly line; an employee's gross pay arrives at the beginning of the line, and at each point an action such as income tax withholdings, pension deductions, etc. takes place. The final product at the end of the assembly line is the employee's pay cheque.

The Metaphor can also be considered analogous to the architecture for the system,

2.3.4 Simple Design

A program built with XP should be the simplest program that meets the current requirements. There is not much building for the future. Instead, the focus is on providing business value. Of course it is necessary to ensure that a good design is used, and in XP this is brought about through Refactoring.

"Treat every problem as if it can be solved with ridiculous simplicity. The time you save on the 98% of problems for which this is true, will give you ridiculous resources to apply to the other 2%." Paul B. MacCready, builder of the Gossamer Condor

2.3.5 Testing

The team focuses on validation of the software at all times. Programmers develop software by writing tests first, then building the software that fulfills the requirements reflected in the tests. Customers provide acceptance tests that enable them to be certain that the features they need are provided.

2.3.6 Refactoring

The team improves the design of the system throughout the entire development. Keeping the software clean does this: without duplication, with high communication, simple, yet complete. The use of Design Patterns is a fundamental part of Refactoring.

2.3.7 Pair Programming

XP Programmers write all production code in pairs, with two programmers working together at one machine. Pair programming has been shown by many experiments to produce better software at similar or lower cost than programmers working alone.

2.3.8 Collective Ownership

All the code belongs to all the Programmers. This lets the team go at full speed, because when something needs changing, it can be changed without delay.

2.3.9 Continuous Integration

The team integrates (checks in) and builds the software system at least once per day. This keeps all the programmers on the same page, and enables very rapid progress. Perhaps surprisingly, integrating more frequently tends to eliminate integration problems that plague teams who integrate less often.

2.3.10 Sustainable Pace

Tired programmers make more mistakes. XP teams do not work excessive overtime, keeping themselves fresh, healthy, and effective. The rule of thumb is that no one should work more than two weeks in a row with overtime.

2.3.11 On-site Customer

An XP project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as the programmers have them. The effect of being there is that communication improves, with less hard-copy documentation - often one of the most expensive parts of a software project.

2.3.12 Coding Standard

For a team to work effectively in pairs, and to share ownership of all the code, all the programmers need to write the code in the same way, with rules to ensure that the code communicates clearly.

3 Extreme Programming and Software Quality

There are a number of aspects of XP that have a direct impact on quality:

3.1 *The On-site Customer*

The Standish Group's ground-breaking CHAOS Reports between 1994 and 2001 consistently indicated that high customer involvement is critical to the success of a project, more so than the use of any methodology (Standish94).

Extreme Programming doesn't just advocate a close relationship with the Customer, it insists upon as much face to face contact with that Customer as possible. Indeed, the Customer is involved on a constant basis. This provides feedback to both the Programmers and Customer very quickly, and effectively communicates what is going right and what is going wrong. If there are problems, steps can then be taken immediately to resolve them. Similarly, if things are going well and the Programmers are ahead of schedule, the Customer can then add more stories to an iteration.

3.2 *Focus on Testing*

The image that the name Extreme Programming invokes is generally not one of rigorous, disciplined development. However, few other methodologies make testing as high a priority as XP.

3.2.1 *Acceptance Testing*

Acceptance or Functional Testing is traditionally performed at the end of a development cycle, just prior to releasing a system into production. This typically results in a mad rush to fix any problems, which often introduces even more bugs.

In XP, Acceptance Testing is performed as often as possible, ideally as soon as a Story has been implemented. Normally, though, Acceptance Testing is performed at the end of an Iteration. Since an Iteration is only 1-2 weeks in length, there is only a small slice of new functionality to be tested, and any problems discovered will be focused in the new functionality. Testing in small increments such as this also helps to identify any issues with regards to the system actually delivering the functionality that the Customer needs, as opposed to what was specified.

In XP, there must be at least one Acceptance Test for each Story. Typically, there will be a test for each scenario within a Story.

3.2.2 *Unit Testing*

Although Unit Testing has always been espoused as important, XP follows its name and takes Unit Testing to the extreme. Unit tests are written by the Programmers, and are automated such that they can provide immediate feedback.

Several Unit Testing frameworks are available for free, such as JUnit for Java, NUnit for the Microsoft .NET platform, etc. These frameworks allow the creation of hierarchical test suites that grow over time as a system is developed. Pressing a single button can run all of the Unit Tests for a system, and the results displayed as the tests are run.

Enabling Software Quality with Extreme Programming

These test suites support the practice of Refactoring by allowing the Programmers to confidently make changes to the implementation of the code and quickly determining the changes caused any regressions.

Under XP, no code is ever delivered to production unless all of its Unit Tests are passing.

3.2.3 Test-Driven Development

One of the most unique aspects of XP is the concept of Test-Driven Development (TDD). Typically, a Programmer will write code, then writes tests after the fact to determine if the code works as advertised.

Under TDD, a test is written first, then the code is written to make the test pass. While this may seem counterintuitive, it forces the Programmer to determine the “intent” of the code up-front, which is expressed in the unit test. The code ends up being simpler, since it is being written solely for the purpose to make the test pass, and the code grows “organically”, i.e. only as needed. TDD also ensures that the unit tests are indeed written, which isn’t always the case with traditional methodologies.

Test-Driven Development is an example of the high level of discipline that is required to properly use Extreme Programming. It is not an easy practice to use, but once mastered it results in a dramatic increase in code quality.

3.2.4 Pair Programming

Code reviews have long been considered a good practice for improving software quality. Again, XP takes the practice to the extreme in the use of Pair Programming.

Pair Programming isn’t simply one person typing while the other is day-dreaming, but rather both people are fully engaged in the development process. The person on the keyboard is typically thinking “tactically”, i.e. working within the current method or function. The person not typing is thinking “strategically”, i.e. within the current class or even at a higher level looking for opportunities for Refactoring or reuse. Additionally, both are actively ensuring that coding standards are followed, and unit tests are written and are passing.

The net result of these practices is effectively a real-time code review.

Some people argue that having 2 people at a workstation with only one typing is a waste resources. Studies have indicated that Pair Programming does indeed create about a 15% overhead in terms of time to produce code (as opposed to 100%, or a doubling of the development time), but in return it improves design quality, reduces defects, reduces staffing risk, enhances technical skills, improves team communications and is considered more enjoyable at statistically significant levels (WilliamsCockburn00), “The Costs and Benefits of Pair Programming”, 2000).

3.3 Iterative Development

Iterative development is an extension of Barry Boehm’s Spiral Model (Boehm88). It provides benefits such as the identification of risks early in development. Compared to the traditional waterfall process, iterative development provides the following benefits:

Enabling Software Quality with Extreme Programming

- Serious misunderstandings are made evident early in the lifecycle, when it's possible to react to them.
- It enables and encourages user feedback, so as to elicit the system's real requirements.
- The development team is forced to focus on those issues that are most critical to the project, and team members are shielded from those issues that distract them from the project's real risks.
- Continuous, iterative testing enables an objective assessment of the project's status.
- Inconsistencies among requirements, designs, and implementations are detected early.
- The workload of the team, especially the testing team, is spread out more evenly throughout the lifecycle.
- This approach enables the team to leverage lessons learned, and therefore to continuously improve the process.
- Stakeholders in the project can be given concrete evidence of the project's status throughout the lifecycle.

(Kruchten00)

4 Case Study

4.1 *IRRS – The Information Repository and Reporting System*

4.1.1 Previous projects

The Facilities Inventory System is the front end to the federal government real estate inventory (valued at \$6.5 billion), and is key to tracking asset and occupancy data managed by Real Property Services. It contains valuable information on program planning, assets, occupancies/vacancies, and tenancy agreements and available to clients via reporting tools.

PWGSC contracted to the private sector for the provision of property management services as part of the strategy to move to Alternate Forms of Delivery. The Information Repository System is the application that was designed and developed by PWGSC to capture specific data from the AFD service providers thus offers us a single point of access to the property management information provides by the private sector. This data was requested from the AFD service providers to enable RPS to monitor the contract; perform data and trend analysis; and facilitate strategic planning activities.

Integrated Warehouse Reporting Project known as Integrated Management Reporting System (IMRS) supports strategic and tactical decision-making by providing consistent management information for employees and clients. By integrating data from Real Property systems, users can answer important business questions in support of the branch's vision to deliver timely and affordable real property programs and services.

4.1.2 Methodology

The internal methodology for PWGSC was originally the Integrated System Methodology (ISM), and later DMR-P.

4.2 *The Client*

4.2.1 Situation

All of our systems require a timely answer to business requests, continuous changes in business practices with more flexibility in changing specifications during the maintenance and development processes and all clients have high expectations on the system functionality. The Information Repository and Reporting System (IRRS) that I managed when implementing XP deals with our department and the private sectors thus provide services to internal and external clients. The requirement included major changes to the application as well as new functionalities thus a major development project.

4.2.2 Project Team

Development and Support

2 Managers

2 Leaders

Enabling Software Quality with Extreme Programming

- 2 Business Analysts
- 3 Developers
- 2 Support Officers
- 2 Information Analysts
- 1 Documentation Analyst
- 1 Web Specialist
- Total: 15 people

Project Office

- Project Plan Maintenance
- Application of standards

Database Development and Information Management (DBDIM)

- Data Modeler
- Development Database Administrator

Application Certification (ACE) Lab

- Release package certification
- Integration/Interoperability testing

Rollout Team

- Database Operations
- Coordinator

4.2.3 Methodology

Before XP was adopted, the team followed a waterfall approach (planning, analysis, design, construction, implementation, testing and, rollout). The individual was working on a steep curve of activities and arrived at the end very stressed to complete the user acceptance testing with very little time to fix the bugs found during testing. As a result, less was delivered than expected. In addition, user documentation and training was prepared and delivered after the rollout of the software. That methodology works in a perfect world, but as usual, nothing is a perfect world.

In 2001, three new staff member jointed the team, essentially on the technical side. I had discussed with my Project Leaders how we could take advantage of this to involve the technical team at an earlier stage of the activities, and have better communications with the technical team. So, when Dave presented XP I realized that these practices were exactly what I was looking for.

4.2.4 Why use the practices?

The system manager and myself had a first meeting with the Leaders and Dave to explore the practices and the implementation of them. I decided to implement the practices and

Enabling Software Quality with Extreme Programming

specifically to adapt them to our environment. We always have strict schedules, major expectations from the clients, and very small time frames for implementation. I considered that XP would be a better discipline as a whole for all team members. In the past, we had problems packaging the software with all of the components, especially documentation such as the test result report, user documentation and the training documents (as well the translation of all of them). So at the end of this new process I expected to have a complete package including documentation, in addition to no carry over items. This was achieved successfully.

4.3 Benefits

4.3.1 The Team

Even with a bit of initial resistance, all the team members adopted Extreme Programming quickly. We had always prioritized all requests for each release, but the division of the project tasks into stories and iterations was the first advantage that was seen by the members. These would be completed and tested in prior to the User Acceptance Testing stage, and indicated to the team the possibility of success using XP.

Before XP was adopted, weekly status meetings only included the team leaders. The decision was made to involve all team members in the meetings, which helped them to better understand the workload of each member and made the process less stressful for all. The constant communication between the team members during the process added greatly to their motivation, and gave them confidence and security about their role. In fact, the XP practices increased the productivity of the team, allowed testing at all times during the cycle, increased communication between the team members, increased the spirit of teamwork, permitted the rollout of a full package without bug fixes at a later time and certainly contributed to the great success of the project.

4.3.2 Individual

Individuals working more closely with the client helped to clarify the change requirements during all phases, and allowed flexibility in understanding and changing the specifications. Frequent tests that involved viewing system design as evolving and a feedback process contributed to allow them to discover any problems earlier thus decrease their stress. The stress level was there but was considerably lower, like a straight line rather than a steep curve right at the end. Using XP increased their influence on the project considerably.

4.3.3 Other teams

We also involved all other team members, such as staff from the project office, the data modeler, the development database administrator, the database operator and the rollout coordinator at our weekly status meeting. This constant communication with the other teams helped them understand our priorities thus allowing them to provide better service. All of them were enthusiastic about these new practices.

Enabling Software Quality with Extreme Programming

4.3.4 Managers

As with most clients, ours have high expectations, and the addition of the private sector components increased those demands. It was important to the team to deploy a high quality product in terms of software, coding and final result. Since XP allows the customers to change their requests, at any time at any stage, it gave them more confidence that we could deliver a product that would meet their expectations, and accommodate changes in their business practices without additional cost of time and money. The end result of our experience marks a huge advance in the field of quality assurance.

This experience was a huge success for several reasons:

- The division of the project into iterations reduced the risk associated with the project by allowing the discovery of any problems early and put a plan in place to mitigate risk;
- The constant communication helped build a sense of teamwork among staff and assisted in the transfer of knowledge;
- The main point of XP practices are the constant feedback to the programmers and the on-site business manager which can make most if not all decisions about the system;
- The flexibility minimized the risks of not meeting what the client needs and allowed the delivery of a high quality end product.

In conclusion if you ask me if I would use XP again, I will not only say yes but I will insist on it.

5 Conclusion

Through its mutually supportive practices, Extreme Programming addresses the issue of Software Quality from the very beginning of the project. Rather than trying to impose rigorous proofs that the code is perfect, XP instead relies on simplicity, clarity and working incrementally to produce the highest quality possible.

6 References

IEEE90 - IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

WilliamsCockburn00 - “The Costs and Benefits of Pair Programming”, Laurie Williams and Alistair Cockburn, 2000

Standish94 - The CHAOS Report –1994, Standish Group International Ltd.

Boehm88 - "A Spiral Model of Software Development and Enhancement," Barry Boehm, IEEE May 1988, pp. 61-72.

Kruchten00 - “From Waterfall to Iterative Development -- A Challenging Transition for Project Managers”, Philippe Kruchten, The Rational Edge, Dec. 2000