Evaluation of Extreme Programming Pilot Project at Labs2

Jonas Martinsson
Jonas.martinsson@gmail.com
Labs2 / Department of Computer Science, Lund University
May 22, 2002

Contents

1 Abstract

This document is a case study of the Community project, a sub-project within the BRIKKS 2.6 project at Labs2. The Community project was a pilot project, using Extreme Programming, an exciting new software development methodology. Both successes and failures concerning the software development practices are highlighted and discussed, as well as the advancement of the project on a biweekly basis in a diary-like outline. Suggestions for improvement of the actual implementation of the Extreme Programming development practices are also given throughout the text.

2 Overview

In the sub-sections below the background for the Community project will be briefly explained. A short introduction to the Extreme Programming formula will be given, along with pointers to find more information. The resources allocated for the project along with the current software development practices at the company is also outlined.

2.1 Extreme Programming

Extreme Programming (XP) is a method for developing software and operating software projects.

Developing bug-free software that is delivered on time has proven to be very difficult. Because of this a number of different approaches have been tried. Perhaps the most widespread and commonly known of these is the waterfall model[i]. But, as with most things in the software industry, this is not a path without obstacles to develop superior software. Numerous problems have been experienced with the waterfall method. XP tries to solve these problems by going for short development cycles in where the design and requirements are flexible throughout the entire project. Kent Beck gives an interesting comparison between the two approaches in his article “Embracing Change with eXtreme Programming”[ii].

XP stipulates a set of core values and practices, which are lucidly outlined in the article “XP Distilled”[iii]. The central core values of XP are communication, simplicity, feedback and courage. The practices, and their actual implementation in the Community project, are examined in greater detail in the section Core XP Practices below.

2.2 Software Development in BRIKKS

The BRIKKS[iv] product consists of a set of COM (Component Object Model) objects and ASP (Active Server Pages) scripts, which together form a web portal that either can stand on its own or easily be extended and integrated with existing backend systems. The tools used for development were the standard setup from Microsoft: Visual Studio, Visual Basic, Visual InterDev and Visual SourceSafe.

The BRIKKS project has historically never been using any well-defined software development methodology. The model actually used could be described as a far-off derivation of the waterfall model, where the analysis and requirements phases have been assimilated with the design phase. The problems with this approach are numerous, and the most common symptoms to date have been missed deadlines and scope reductions.

2.3 The Community Project

The introduction of Extreme Programming at Labs2 came at the start of the BRIKKS 2.6 project. The task at hand was to implement community support in the BRIKKS platform. 4 developers and one customer were assigned to the project. One of the developers (myself) was also taking on the extra XP responsibilities of being coach and tracker, which meant that only 3 of the 4 developers were working full-time writing code in the project.

When the project was launched, the requirements were extremely vague and no one knew what the word "community" actually was to connote. As the 3rd iteration drew to a close the requirements had been laid down by the customer and approved by management as more or less static. It was not until then that there was a first opportunity to plan the release scope.

3 Core XP Practices

In the following sub-sections I will give my personal view of to what extent the team followed the 12 Extreme Programming practices. There are plenty of areas where improvements can be found, but this should in no way be interpreted as criticism of any individual team member; the responsibility for the failure to follow the practices bears heavily on the coach.

3.1 The Planning Game

In the beginning of each iteration there was a planning meeting with all of the developers and the customer. It was decided early on that a representative from the BRIKKS project's architecture group should be present during these meetings. This turned out to be a successful decision, not only for the architectural feedback but even more so for the detailed knowledge of the coming functionality that the new BRIKKS release would support. It may not be a good thing that the customer will learn of new features from the architect, as it should be the other way around, but this was our cruel reality. These architectural requirements for the upcoming release was in no way documented but instead communicated during the initial iteration planning meetings. As the project proceeded, the meetings unfortunately continued without an architect.

Throughout the project we failed to use the historical data to project our velocity, but instead employed a naive optimism and aimed for higher velocities for future iterations. While there is nothing wrong with aiming high, it does not serve for the most realistic project plan, which in my opinion should be the main goal. It is interesting to note the differing opinion of the coach and project manager on this matter. The manager argued that Parkinson's Law[v] ("Work expands so as to fill the time available for its completion") would require us to plan a little more work than we probably could complete, in order to work at our optimal pace. This argument is in no way compatible with XP's standpoint of using historical data for future estimations. I, as a coach, maintain that the motivation XP breeds is the key driving force behind carrying out efficient work; not an overloaded and unrealistic project plan, à la Parkinson's Law. In hindsight, I wish I had voiced stronger arguments for this during the course of the project.

The estimations of user stories proved to be very successful. After only one or two iterations we were able to very accurately estimate the size of a new user story, simply by comparing it to previously implemented ones.

3.2 Pair Programming

The developer's reaction and evaluation of programming in pairs were both positive and negative. What we all agreed to be beneficial was the increase in source code quality and a decrease in the number of bugs. The partner caught many mistakes right away. We also noticed a superior and more coherent design in the pair produced software.

It was a lot of fun coding in pairs, but at the same time very exhausting. Some of the pairs in the project drove each other to new levels of stress (and code production), and a heart attack didn't feel too far away at the end of some workdays.

The problems we experienced were mostly due to the difference in experience for the different programming languages used. The developers found it frustrating to team up with a partner with a different programming skill level. If they were more experienced than their partner they felt a lack of assistance and they needed to slow down and explain things to their partner regularly. The less experienced partner, on the other hand felt frustration because he couldn't keep up with the pace. This problem manifested itself clearly when it came to pairing up for tasks which were not connected to a developer's preferred language. The C++ programmers were especially reluctant to implement ASP engineering tasks. This caused the pairs to become much more static than would otherwise be desired. Changing pairs has the benefit of keeping the dynamics and information flowing within the project.

One interesting verdict from the programmers in the project was that pair programming offered about the same efficiency as traditional programming, in terms of invested man-hours.

3.3 Testing

XP relies heavily on testing. There are two types of tests in XP, unit tests for every method, and acceptance tests for the overall functionality connected to user stories. For the purpose of writing unit tests we used COMUnit[vi], and framework developed by one of our former colleagues at the office.

Like many good test frameworks, COMUnit is free and lightweight. The usage of this tool in our environment was a tremendous benefit and everyone was happy using it. The XP practice of writing the tests first, was successfully applied, and helped the programmers in defining the interface design of the components. As the test suite and the components grew it became increasingly time-consuming to maintain the tests, and if the interface changed the tests needed to change with them. But, at the same time, as a component grew and increased in complexity, the accompanying tests helped the programmers to feel confident about its stability, which alone justified the time spent on the tests.

During the last iterations came the implementation of security. This had the unfortunate drawback of making the test suite awfully slow to run through. Ideally, the tests should be run as often as possible, but execution times for the complete test suite in the neighborhood of 45 minutes caused the test runs to be increasingly dispersed. This problem was never solved satisfactorily; the solution most commonly used was to upload and run the tests on a more powerful development server, which offered a threefold speed improvement.

COMUnit is built for testing COM objects only, which means that the project was still missing a tool for testing ASP code. Much of the ASP code is GUI related, which is inheritably difficult to test in an automated fashion. While there exist tools for ASP and HTML testing (ASPUnit[vii] springs to mind) none of these tools were found to be beneficial for the Community project. Our experience with ASPUnit, which was still in beta at the time, was that the test framework itself often crashed and did not produce reliable results. The lack of an ASP unit-testing tool carried a heavy weight on the project, and most of the bugs later found in the code were not surprisingly traced to the scripting code. Instead of relying on automated tests, visual inspections were performed of the results, which proved to be both time-consuming and error-prone.

On of the biggest mistakes in the project was to neglect to implement and make use of acceptance tests. Acceptance testing is a very important tool for the customer, as it will enable him to confidently verify that a user story has been successfully delivered. Here again, visual inspections and manually testable scenarios were used to verify the completion of stories. In my opinion, the introduction of acceptance tests, specified by the customer and implemented by the programmers, was the single most valuable XP practice that the project failed to apply.

3.4 Refactoring

The refactoring practices of the team were very successful. With the help of our automated unit tests we had the confidence to mercilessly refactor our code when we felt the need for it. While there exist several interesting and timesaving refactoring tools on the market that will assist in many basic code improvement techniques, we didn't have the opportunity of using them due to the simple fact that we were developing in programming languages without support for reflection (which is a prerequisite for these tools).

3.5 Simple Design

During the first iterations a base functionality came in place and was demonstrated to the customer. From this foundation the project was gradually steered into a direction where it a few months later finished. By keeping the options open the finished software design was much better than what possibly could have been accomplished from solely using a design phase before starting the implementation.

The programmers' general opinion about this approach was a sense of lacking direction. It is possible that they felt this because they were used to working from a big design up-front. Whatever the reason, it is true that the architecture could have been better defined in the beginning. This could have been accomplished by having a meeting where we sketched UML-diagrams (Unified Markup Language) or CRC-cards (Classes, Responsibilities, Collaboration) on a white board. A white board in the work area, a resource available to us in the first iterations while residing on the 3rd floor, would have helped greatly as a tool for communicating the design and other architectural decisions.

3.6 Collective Code Ownership

It was clearly communicated that everyone (still working in pairs) has the right to make modification to any source code produced, but this practice never really took off in the Community project. If a defect was found somewhere, it was always communicated to the person who had written that peace of code, and this person later made the relevant changes. If instead the pair finding the bug fixed it, the process would have gone quicker and everyone would have felt more confident about the entire source code.

Another reason for the failure of this practice was no doubt the static programming pairs employed. It is often wise to let one programmer sign up for many of the similar engineering tasks connected to a story, but at the same time it can also hinder the spreading of knowledge within the team.

3.7 Continuous Integration

Continuous integration can be viewed upon as two different practices in a sub-project, such as ours. Firstly, we have the challenge of internally integrating the source code, and secondly, the integration of our code with the other software teams' code.

For the internal integration we used Visual Source Safe and one of the programmer's local workstations as web server as integration machine and showcase to the customer.

As for the integration with the other teams, this was one of the more difficult practices for us to implement. The other teams were not utilizing XP and did not have an integration server at all. Furthermore, they did not plan any internal releases until very late in the project, if at all.

Setting up a separate integration server only for the Community project was discussed but the decision was ultimately postponed until there was external code to integrate with. Eventually, a joint integration server for the GUI team and the Community team was incorporated very late in the project.

3.8 On-site Customer

The Community project had the benefit of an on-site customer. Our knowledgeable customer gave us the benefit of an actual set of detailed requirements, a rare gift in many software projects. In those cases where the details were not defined or too technical for the customer, we worked out the details together. From the programmer's perspective working with the customer in this way was a very pleasant experience.

However, as the project drew to a close, we heard opinions opposing the customer's concerning the graphical layout of what we had produced. This would not have happened if the customer role had involved experts in usability and graphical design as well.

3.9 Small Releases

As it happened, there was only one release from the Community project. Since we had an executable platform with the community features up and running most of the time, it would not have been a huge overhead to release a snapshot of this between any two iterations. The customer and project manager had the opportunity to examine our results at any time through our internal integration environment, so there was simply no need for any intermediate releases.

3.10 Sustainable Pace (40-Hour Week)

We followed the XP tenet of no overtime to a large extent. Only some overtime occurred in the final stages, and people were allowed to take vacation anytime during the course of the project.

3.11 Coding Standards

During the early iterations a *de facto*-standard for C++ source code was set. However, the team members coming back from vacation after iteration 2 did not accept this standard, which led to some unnecessary overhead and refactorings later on. After this clash of wills, the coding standards were well adhered to. Concerning the ASP code, we followed a set of rules defined by the GUI team.

3.12 System Metaphor

With such a large project as BRIKKS, it can be very limiting to stick to only one metaphor for identifying the project. Different sub-systems will need their own metaphor to work as intended. One no-brainer metaphor is "brikk" for describing the visual layout of the application within the portal.

The Community project applied a "naive metaphor"[viii] – i.e. the actual object: communities. The drawback with using a naive metaphor is that it may not yield any extra insight into the domain. The metaphor is one of the most abstract and unclear concepts in XP, and the benefits of using one can also be questioned.

The lack of a more powerful metaphor might be the reason that one of the programmers complained about not seeing any big goal, and felt that the smaller goals, in form of new user stories, where changing each iteration.

4 Iterating Through the Project

Iteration	ESTIMATED VELOCITY	True velocity	Average velocity
1	5	6	6
2	6	7	6.5
3	13	6.5	6.5
4	8	6.25	6.5
5	8	7.75	6.7
6	8	6.5	6.7
7	8	7.5	6.8
8	8	7.5	6.9

Table 1. The velocity of the Community XP project measured in "Units".

4.1 Preparing the Project

-2001-07-20

After lobbying the project manager about the benefits of the XP way since December 2000, the go-ahead sign finally came in July 2001.

As a preparation for the project I held an introductory XP course for everyone interested at the office. All of the team members of the Community project participated. Shortly after this, on July 19, we had a brainstorming session, trying to come up with functionality that would later be transformed into user stories for the Community project.

Since the user stories at this early stage had no founding in management or sales, it was important to choose only the most essential stories for the first iterations until the requirements had been verified. Note that this kind of approach would have been impossible in a more traditional development cycle with a big design up-front of the whole system. XP enables the requirements to be changed during the lifetime of the project without impairing the development speed or quality of the product.

The iteration length was set to 2 weeks, in an attempt to make the iterations short enough to continuously steer the project in new directions, and long enough to overcome the overhead time of the iteration planning meeting at the start of each iteration.

We decided to measure all of our programming efforts in an abstract unit we named "Unit". By comparing user stories to each other, estimations of new user stories were made. In the beginning of the project there were no historical data or estimations to compare with, so the decision was to map a Unit to one ideal working day for a programmer. One ideal working day is 8 hours of work without interruptions from e-mail, phone calls, meetings etc. This mapping would only be relevant for the first iteration, after this, Units would become a measurement that defined itself based on previous estimations. This approach will increase the quality of the release plan, where one Unit will be a constant measurement and not change over time. The main benefit from using an abstract unit over for example man-hours is psychological. If we would have used man-days instead it could have become very depressing for the team to see that they constantly were completing 7 man-days worth of development with 4 people during 2 weeks. Estimation is a very difficult art, and using an abstract unit mapped to actual time tremendously improved the estimations.

For the first iteration we decided to estimate that we could complete 5 Units. There is no scientific foundation for this claim, it is simply an educated guess, and the accuracy of the first velocity estimation does not matter very much, as for the rest of the iterations the historical data will be used instead.

4.2 Iteration 1 – 2

2001-07-23 – 2001-08-17

The first two iterations were being executed in the middle of the summer vacations. I, and another senior programmer were away during the second half of the first iteration and during the whole of the second iteration. Trying to do XP without a coach and doing pair programming with just two people is of course not the optimal way of achieving success with XP. However, the team had to make do with the resources available and did the best of the situation. The tasks during the two initial iterations were to implement a photo gallery, a shell or template for "My Brikk" (a center for each user in the BRIKKS platform), a guest book, and a simple administrative interface.

In the first iteration of an XP project, it is important to produce basic functionality that can present a flow of the system. This base is then augmented throughout the project and will serve as a foundation for customer demonstrations. The tangibility of the base functionality is often a very lucrative environment for ideas and visions of new features. This goal was successfully accomplished during the first iterations of the Community project.

4.3 Iteration 3

2001-08-20 – 2001-08-31

The third iteration was the first iteration when the team was fully manned. Vacation time was over and we finally got a chance of trying out XP in a larger context. The pairs could now change dynamically during the day, and we started having short stand-up meeting each morning, where we went through issues that had been encountered the day before, aired problems, and planned the work for the day ahead.

Now when the development resources had been doubled it felt natural to be able to accomplish twice as much work as before, so the estimation for iteration 3 was set to 13 units, double our average historical velocity. This proved to be a huge mistake. In reality, the team did not complete more than 6½ units, exactly the same velocity as with just half the number of developers! Why did this happen?

The answer is that when the two developers, who were more experience in coding C++, came back from vacation, they found that they needed to do many large-scale refactorings of the code. The tests that had been written for the production code during the first iterations were also found to be more or less non-existent; a very logical course of events, considering the absence of a coach, and programmers with no previous experience of test-driven development.

The user stories picked out for iteration 3 were: extending the photo gallery with capabilities for moving pictures between folders and presenting a slideshow, and basic administrative tasks for managing and creating communities. As the iteration proceeded and it became evident that the developers had committed themselves to too much work, the slideshow and some of the administrative functions were removed from the scope of this iteration, on request from the customer. It was also during this iteration that most of the functions for the community were defined and verified.

It was also becoming apparent that many of the new requirements were depending on functionality that was going to be delivered by the core team. Especially a new security implementation that supported sharing of data between registered users of the platform was going to be needed in order to successfully implement a large number of the requirements. The core team was not running as an XP project and had not yet started to specify the interfaces or the functionality that was to be included in many of the relevant objects.. Because of this our customer had no choice but to postpone the implementation of some the more desired user stories. The priority of the user stories was now not dictated by customer value, but more by the technical dependencies present at the time. One interesting approach had been to implement the functionality anyway but instead of the actual security objects using mock objects, emulating the functionality we were waiting for, and later when the real objects were ready for use the implementations could be switched. This would have been a very attractive approach and perhaps better than the one actually used. However, the reason for not choosing this approach at the time was that the desired functionality was reported to come available to us much sooner than they actually were.

4.4 Iteration 4

2001-09-03 – 2001-09-14

During the iteration planning meeting, the project team members raised the opinion that we probably could complete 13 Units during iteration 4, and that the reason for the low velocity last iteration was due to the large amount of refactorings and test cases that needed to be post-manufactured. However, after some discussion regarding Yesterday's Weather[1], we decided to make a compromise: 8 Units. The lesson that should be learnt from the previous iteration: Doubling the development resources does not necessarily mean a corresponding increase in velocity.

This iteration's user stories involved improvements to the community handling and administration. The development during this iteration went very well and I, as a coach, felt that we finally had gotten many of the practices down pat. The pair programming had become more of less second nature, and the unit testing was really starting to show its advantages. There were also a lot of refactorings taking place, which continuously increased the code quality.

With no interruptions I am confident we would have had no difficulty meeting our estimates. Unfortunately for our velocity, there were a lot of small events, which stood in our way during this iteration. Firstly, there was a big reorganization of our office space, which forced the whole XP team to move down two floors. Secondly, this move also made our working environment worse. Thirdly, some of us attended a Microsoft course in Malmö and finally, many people were home sick during the iteration.

4.5 Iteration 5

2001-09-17 – 2001-09-28

Again, at the start of the fifth iteration, we felt that we had a reserve to take from. This caused us to once again write up the velocity compared to our historical data. For this iteration we estimated 8 Units, a number we would have reached during iteration 4 if we had been undisturbed. It is hard to say if this was a mistake or not, since there obviously had been a lot of disturbing elements, and we felt that we had gotten wind in our sails.

For this iteration our customer decided, among other things, that we should implement a wizard for creating new communities, and a brikk with community management functionality. Our work proceeded very well, and this was our most productive iteration during the entire project. We more or less met our estimates.

[1] If you predict that tomorrow's weather will be just like today's, you will right in about 70% of the time, which is a better estimate than most weather forecasting services. This is an often-used analogy in XP.

4.6 Iteration 6

2001-10-01 – 2001-10-12

It was not until the second half of iteration 6 (October 9) that we finally got a chance to test a first version of the sharing functionality from the core team. This enabled us to schedule very important user stories to this iteration; stories that in a different project would have had much higher priority, but not here because of the dependency to the other internal sub-projects. Postponing user stories like this might not seem like a big deal. What difference does it make if a feature gets implemented sooner or later as long as it will get done during the scope of the project? The big danger with postponing important user stories is that there might be too many important user stories left when the project approaches its deadline. This makes for a very volatile and shaky release plan; if something unexpected happens in the project, important user stories may need to be cut from the scope or alternatively the deadline moved.

It must be said that the estimated velocity for this iteration was a bit naive, considering that one of the developers would be on vacation for half of the iteration (and a bit into the next iteration). On one hand we have seen from before that adding two extra people to the iteration did not dramatically change the velocity, but on the other hand we have also seen that absences due to illness and courses did affect it. The actual numbers showed that we were less productive during this iteration than the week before but it is difficult to retrospectively pinpoint the reasons for this.

4.7 Iteration 7

2001-10-15 – 2001-10-26

During the end of the sixth iteration and throughout the seventh we suffered from problems using the functionality produced by the other non-XP team. It goes without saying that producing high quality, bug-free code is extremely difficult, even more so in such a complex piece of software as BRIKKS has become. However, the objects that were delivered had many bugs that would have been easily caught using the XP practice of unit testing.

Looking at our release plan, throughout the project we had had 5-10 units too much to squeeze in before the deadline. This had been communicated to the project manager for the whole BRIKKS 2.6 project well in advance and we were all aware of the fact. However the decision was to wait with the cutting of scope because we were all hoping to pick up our pace and increase our velocity. Now, it was not possible anymore to close our eyes to the fact that Yesterday's Weather was holding the truth about our velocity. While this might not normally have been a mistake, together with the fact that we were postponing our more important user stories till the last iterations made for a dangerous combination. Now was the time to face the problem, and the decision was to move out some of the user stories to other teams.

We also discovered that we needed to develop a new COM object to take care of the friendship functionality in communities. Up till now we had believed that we could use the existing Contacts object, but it proved to be lacking some of the features we needed, and was logically not working in the desired way. We decided to develop a completely new Friends object.

All of the above affected our release plan. The developers were beginning to feel the pressure of the approaching deadline and we were starting to ease the requirements of programming in pairs and writing tests for everything that could possibly break. This was not wise. Sacrificing the long-term benefits of high quality to produce short-term gains is a practice that is very easy to succumb to under pressure. Besides risking quality, the team members suffered from a chaotic environment, where the XP values and practices were greatly devalued.

As it happened, this solution worked out for everyone in the end, but it was still a very risky development of events, considering the high quality requirements in the project (zero bugs).

4.8 Iteration 8

2001-10-29 – 2001-11-19

The last iteration of the project contained some of the most important user stories, for reasons already presented. As the iteration proceeded it became clear that the BRIKKS 2.6 release project plan most likely would have to be pushed forward. It is my belief that this had the psychological effect of making us lose some of our focus. During the project we had the iterations and engineering tasks as our means of keeping focus on deliverables. This had helped us to concentrate on a goal. Now when the deadline approached for the last iteration we had changed our focus from the end of one iteration to the major project deadline. With the deadline not fixed, it felt as if we did not have any date to focus on any longer. However, we finished our user stories assigned for this iteration and were more or less finished at the time originally set out.

4.9 After Iteration 8

After the last planned iteration we were left with time to fix a few defects in our code, defects that most likely was the result of our lack of XP adherence in the last two iterations.

An art designer consultant was also hired after the planned deadline for redesigning much of the graphics that had been produced. While it was not surprising to us as developers that our graphics design was not fully approved (we never claimed any greater skills in this field), it was more disturbing that this GUI expert had not been a part of the customer team from the beginning. Our customer had from the start expressed a wish to have a graphical design that stood out from the rest of the applications in BRIKKS, but since no GUI authority had been part of the project, this was the result.

The drawback of redesigning the GUI at the very end of the project was that we had to go into the ASP scripting code and make changes. Due to the lack of unit tests for the scripts, these changes could not be made with full confidence, and took about a week to implement and test. While it is true that XP allows for design changes throughout the project, this relies on automated and thorough tests, which serve for confident refactorings. Without these tests there will always be a greater overhead in the redesign.

4.10 Post-Mortem

Looking back on the project's release plan and the dynamics that allowed the remaining requirements to be changed between each iteration, I consider the plan to have been very successful.

The biggest idiosyncrasy of the plan was no doubt the velocity's constancy, between iterations 1-2 and the remaining ones. The possible reasons I can think of are:

- Not enough testing during the first iterations. The user stories delivered in the first iterations were not really completed, because they did not have adequate tests.
- The Mythical Man-Month[ix]. More developers does not immediately cut the development time, it's quite the contrary.
- The disturbing elements that were decreasing our velocity in iteration 3-8 were constant.

From a customer perspective the project was a success. Between 88-93% of all the desired community functions that our customer had originally specified were implemented.

We had aimed for a zero-bug delivery and that requirement was not fully met. The reasons for this is most likely due to the difficulty writing automated tests for the GUI and scripting code. Looking at the defect reports that were submitted to our bug database, the vast majority of issues are related to our ASP development.

5 XP roles

5.1 Coach

As the XP aficionado of the company I took upon myself the role of coach in the project. The coach's responsibilities in an XP project can be summed up in 5 words: "Keeper of the Holy Grail". His job is to make sure the team follows the core XP practices and are happy working in the team. As a consequence, this means that I take responsibility for many of the things I consider went wrong in the project.

5.2 Tracker

The tracker is the person measuring the progress of the iteration, and flagging for tasks being delayed or problems with meeting the deadline. According to Ron Jeffries the tracker "needs to have a nonthreatening approach to getting information, needs to be sensitive to body language and other nonverbal behavior, and needs to be willing to track on a regular basis."[x] As a tracker I kept the information about the team's velocity[2] and other statistics in an Excel file that was publicly accessible for the team members, as well as for management.

A couple of iterations into the project I dropped keeping track of the developers' individual velocity, simply because it did not seem to carry any relevance for the project. If one of the developers where behind schedule during an iteration, the others were not late in helping him out, which made it very easy for the tracker in this respect.

[2] Velocity is the speed of which user stories units are finished, and kept as a measurement of how fast the team is working

The real work for the tracker laid in the tracking of the team's velocity, and updating the project release plan. One of the often reiterated rules of XP is to use "yesterday's weather" when it comes to planning for future releases. The biggest mistake of the tracker was to yield to the management's, the customer's and even the programmer's queries for an increase in velocity. The velocity of the team stayed very stable throughout the project (6-7¾ units/iteration), but there were still no estimation of an iteration that was below 8 units after iteration 2. While the team felt that they could pick up their speed, it was clear that the historical velocity should have been used to produce a more realistic project plan.

5.3 Developers

The development of the user stories for this project required a varied set of skills from the programmers: C++/COM, ASP, SQL, etc. While all of the developers had adequate knowledge on C++/COM and SQL, only one of the four had any previous experience writing ASP code. The ASP code proved to take a large amount of the coding effort, and the lack of knowledge and interest in learning the language, and its best practices, seemed to slow down the velocity of the project considerably. More ASP expertise in the project would have been desired.

5.4 Customer

Our customer had extensive previous experience of using and managing communities, and was therefore well suited for his role. However, the BRIKKS project is a huge beast, with close to 200 000 man-hours invested to date. This implies that a customer must also have to pay close attention to a lot of other issues, e.g. usability and adherence to the existing graphical design. For this reason it would have been beneficial to have a group of customers sharing the responsibility for the requirements where each one would have his own field of expertise.

As it were implemented in the actual community project, there were initially 3 customers. Björn Flintberg was our main point of communications with the customer group, and actually the only one of the customers ever working with the project. But it is doubtful if the project would benefit from the additionally allocated customers, as they more or less were responsible for coming up with visionary ideas and concepts for new functionality in BRIKKS, a task that our one customer pulled off with flying colors. What the project really needed to be able to confidently implement that functionality was requirements and design of a more artistic kind. Art designers and usability experts would have meant an important addition to the customer group, and is one of the more important points where improvement can be accomplished in future XP projects.

For such a group it would also be important to synchronize their overlapping demands, to prevent them from communicating different messages to the developers. Having regular meetings, based on honest communication, between the customers, could most likely accomplish this.

6 Summary

The unprecedented mixing of C++ and scripting expertise in the project proved successful. Suddenly the C++ developers, who are defining the interfaces, got an understanding of how their objects would be used, and could design and adapt the design accordingly. The script programmers finally found it possible to influence the design of the COM interfaces to better suit their needs.

I identify the most important areas for improvement in our project as:

- Introduction of acceptance testing;
- Broadening the customer group;
- Finding an efficient test framework for ASP code.

Compared to the other sub-projects within BRIKKS, I consider our main advantage was the usage of an actual methodology, and that we had a clear road map of how to complete the task we were given and a set of practices to follow. But I also claim that the usage of Extreme Programming as a software development methodology for our project was very successful. Firstly, it does not require a great deal of theoretical knowledge of the method. It is easy to learn, and in actual practice it becomes more transparent than many other models. Secondly, instead of processing and working with meta-information, in the form of, for example, specification documentation, the team will stay more focused on the core of the product – the source code. Thirdly, the flowing of oral communication within the team, together with the united design decisions, led to a tighter team with a considerably stronger identity and pride for the finished product. Fourthly, the pair programming practice led the team members to enjoy their everyday work much more than before. A common situation for a single programmer is to get stuck on a problem for a long time, and sensing a growing frustration. This is very rare in a pair-programming environment, since the collaboration between the programmers almost invariably will resolve the problem swiftly. In the Community project I can't recall a single situation with a pair stuck on a problem for longer than 15 minutes.

The humanistic and professional values of Extreme Programming might not revolutionize the world, but I consider them to have improved the development environment for the members of the Community project dramatically.

7 Further Reading

This case study has unveiled the inner mechanics that drove an actual Extreme Programming project forward (and sometimes sideways), how the cogs interacted with each other, the programmers and their environment. Hopefully, it might have given you inspiration to learn more about XP. If you want to read more case studies from other authors you can try the resources below:

- Daniel Karlström: *Introducing Extreme Programming – An Experience Report;*
- James Newkirk & Robert C Martin: *Extreme Programming in Practice.*

While both are excellent case studies of XP projects, they also have their individual approach of telling their story. The main difference between their reports and mine are that I have chosen to center my story along the lines of a diary, where each iteration, and its unique challenges are described individually.

[i] Waterfall methodology http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm

[ii] Embracing Change with eXtreme Programming *by Kent Beck* http://www.computer.org/SEweb/Dynabook/WhatIs.htm

[iii] XP Distilled *by Roy W. Miller and Christopher T. Collins* http://www-106.ibm.com/developerworks/java/library/j-xp/

[iv] BRIKKS http://www.brikks.com

[v] Parkinson's Law http://www.heretical.org/miscella/parkinsl.html

[vi] COMUnit http://comunit.sourceforge.net/

[vii] ASPUnit http://aspunit.sourceforge.net/

[viii] Xplorations: The System Metaphor http://users.vnet.net/wwake/xp/xp0004.shtml

[ix] The Mythical Man-Month http://www.amazon.com/exec/obidos/ASIN/0201835959/qid=1008780126/sr=8-1/ref=sr_8_7_1/104-4212967-1571923

[x] Ron Jeffries "Extreme Programming Installed"