A great challenge: XP in a typical dot-com

Ye Yongqing
IT department, Europeloan Bank
Gulledelle 92
1200 Brussels, Belgium
+32 2 4020120
yongqing.ye@europeloan.com

Winston Wolff
IT department, Europeloan Bank
Gulledelle 92
1200 Brussels, Belgium
+32 2 4020119
winston.wolff@europeloan.com

ABSTRACT
This paper describes the implementation of XP in a young, dynamic, growing global web bank, and an environment with rapid changing user requirements. This paper typically describes the problems, challenges and solutions we encountered during the period we implement XP.

Keywords
WebBank, XP,Test-First Programming, Pair-Programming, Velocity, J2EE

1 INTRODUCTION OF EUROPELOAN BANK

EuropeLoan Bank is a young, dynamic and rapidly growing global web bank. It was created in November 1999, with only three people and one office located in Brussels. This company has been growing very quickly: currently we operate on five countries, with different offices in each country and about 50 direct and indirect employees. Its capital reaches 3,000,000million BELGIUM FRANCS. EuropeLoan Bank is offering residential mortgage loans at very competitive rates. This is possible because we have designed our company to be more flexible, more efficient through computerization of all areas of our business, and timelier through use of the Internet to communicate with customers and suppliers. All of this means reduced costs for us and therefore lower interest rates and initial costs for the customers too.

2 WHAT IS SPECIAL ABOUT EUROPELOAN

EuropeLoan Bank is a new growing bank and we have to continue opening business in more and more countries. This means that we have to continue launching our website in more and more countries. At the end of last year, we were operating in 5 countries. This is a historical event: there has never been a bank that has opened branches in 5 countries during one year. Some of these countries have several languages, and different countries have different special situations. For example the type of the offered products and the way the conditions for the products are calculated differs for each country.

Because EuropeLoan Bank started from scratch, we didn't even have a BackOffice in the beginning. Currently, we only have a very simple BackOffice, which provides only some functionality. There's enough functionality present that the customers can do their job, but there are great possibilities to improve it. We have different types of BackOffice end user: we have helpcenter users who answer customer's phone and email and help the customers fill in their application, etc. We have account managers who are in charge of the account information of the customers and who keep in touch with the customers. We have country managers who are responsible for the operation in one country and who make the underwriting criterion decision. We have approval committee persons, who make the final decision for applications. And finally we have system administrators.

We do have BackOffice end user on team, but the different types of BackOffice end user have different needs, and therefore we still have to communicate with people who are working in other countries.

Because it is a new online bank, none of us know very well about everything. We cannot predict requirements. We have a very strong feeling of being stuck in a circle: we encounter a problem, solve this problem and find a new problem, solve the new problem... User requirements change very rapidly. There are always a lot of requirements from different countries and different roles.... This is a typical dot-com situation.

Also technology changes very rapidly. We use a lot of new technology in our system: Java, Jsp, Weblogic, EJB, XML, and Dcom.

3 BEFORE XP IN EUROPELOAN

We started the development of our system in the beginning of January 2000. At that time, we let a consulting company start the development of the system for us. The consultants knew XP, they developed the system in the most simple and quick way, and also wrote test cases for our system. We released our first version in February 2000.The system did go online on 17 February 2000 (!). This first production version did have a minimal number of features. After that time, our system has become bigger and bigger. Each release, more features are being implemented and brought into production. The development team expands: more and more programmers are recruited in order to keep on develop and maintain the system. Currently, we have 11 programmers, and 3 of

those are working in another country. The consultant is planning to quit; she reduces work time at EuropeLoan Bank to 2 days a week.

At that time, we decided to adapt XP methodology, and the consultant helped us to become familiar with the code and to teach us XP.

4 IMPLEMENT XP NOW

Description of a development cycle

We fix our development cycle as three weeks (21 days). At the beginning of the development cycle, the different end users (country manager, account manager, helpcenter etc.) will send their function requirements and brief description to our team leader. We collect all the requirements; write them one by one on cards. Then we set up a conference meeting with the CEO and the country managers (some of them work in a different country). At that time, we try to understand exactly what they want, what the story is. Then we organize a development meeting, all the developers will attend this meeting. All of us will estimate the amount of perfect days on the cards. The next day, we have another meeting, and let the CEO and the country managers prioritize the cards, and to choose the card with the limitation of our velocity. Later, each programmer chooses one card. We start our implementation. At this moment, the person who picks up the card will be responsible for contacting the person who proposed this task.

All the members of development team and HQ officer and Belgium Management group work in a big room on the fifth floor. This really makes life easier, because we can exchange information very quickly. Programmer can change pairs very easy; we can ask each other for help easier because all of us are face-to-face.

There are a few developers, however, who are working a different country. Also, most (but not all) the customers work in another office, in the different countries. We do a lot of conference meetings to keep in touch with them.

When we make a big change, which will affect our database, architecture or other people's work, we always have a short meeting, which all the developers will attend. All of us will discuss the design. So every one will remember and understand the change.

Before Jan 2001, we make the last week of the iteration is our testing period. All of us should stop coding. We start to test and fix bugs. But soon, we realize it is not a good idea to test at the end of release cycle, we always get a lot of bugs during the test, we always are busy on fixing the highest priority bugs, we always leave other bugs into next cycle and always forget to fix it in next cycle. Now we hire a tester to responsible for testing. Whenever we finish one part of story, we will ask the tester to test it. So the story will be fully tested when it is done.

Statistics analysis about our velocity

Normally, when system becomes bigger and bigger, it will take more time on maintaining and development speed will be slow down rapidly. But because we implement XP, developers get more and more familiar with code and confidence on refactoring code. So the actual perfect days still stay on a stable level.

(Only show recent period)

Index	Start date -- end date	Estimated perfect days	Actual perfect days
1	27/09 -- 18/10	25	20
2	19/10 -- 15/11	20	13
3	16/11 -- 06/12	20	21
4	07/12 – 28/12	21	20

What problems do we encounter?

XP is new for EuropeLoan Bank, and we have encountered some problems:

Overall architecture

The overall architecture is probably too complex. The very first iterations were very TTSTTCPW(Do The Simplest Thing That Could Possibly Work) and YAGNI(You Aren't Going to Need It). Unfortunately, this happened for the wrong reasons: because of simple deadline pressure. After a few iterations, things started to slow down, and we got the idea that we needed to 'clean up' some of those 'obviously too simple' things. That's probably where it went wrong. In this period, we developed some things that were much more complex than they should have been. Note that at this time we weren't doing the Planning Game. Luckily, we were test-infected from the beginning, so that all our code is Unit Tested well. Lately, we've been throwing out some of this 'legacy' code. Currently, we're talking about refactoring out the EJBs, because it would simplify the whole thing enormously.

Modifications to the overall architecture

Modifications to the overall architecture are in some places a big chunk of work. Some simplifications have been postponed for a long time, because there's the feeling that it would cost too much time to clean it up. This is a shame, because in reality this unnecessary complexity consumes a lot of time too.

Few senior developers

Most of the developers don't have a lot of experience in a business environment. And they always would like to add

more code in existing system instead of refactoring and simplifying existing system first. As a consequence, we don't always have the courage to simplify the code if we think that it's unnecessarily complex.

Stories too big
Stories too big. Sometimes, a story is too big to implement in one cycle, but we have difficulties splitting them up.

Prioritize stories badly
It happened a few times that, at the beginning of iteration, all of us start with their stories simultaneously. Therefore some of the stories were not finished in time. The stories that were finished were not the most important ones.

Not enough knowledge about stories
Business people don't always provide necessary information in time. It has happened several times that they propose a story that seems simple, and that's estimated as a simple story. When the implementation starts, the story turns out to be much more complicated than we originally thought it would be. This is about bad communication with the customers. The customers themselves don't know enough about the details of these stories up front.

Problems finishing stories
Besides the coding, there are other aspects needed to finish the stories. For example, when we develop a website for a new country, we need translation texts for all the different languages. Usually, we get the translation texts only the last minute.

Deadline pressure
Because of the pressure of the deadlines, some developers tend to quickly finish their task instead of refactoring the existing code into something really clean. As a consequence, some of the code has become big and messy. In the end, we have to arrange some days for fixing.

Pair Programming
Some people do pair programming a lot, but some people seem not to pair a lot, they still think that pair-programming will waste time.

Ownership of tasks
Some people have the feeling that they give up their own tasks, when they pair and help other people to finish their task first.

Acceptance tests
No automated acceptance tests. This is a big problem. Functionality increases rapidly, so the amount of work needed to test the entire system also increases. Currently, we foresee part of the iteration simply for testing and bug fixing.

What benefits do we gain from XP

Define development cycle. Business people can see clearly what tasks have been done. They can see the progress of development. They can estimate the cost, and do cost benefit analysis.

There's more focus on import tasks

Presentation layer
No testing of the presentation layer. We have Junit to test java code and Httpunit to test Jsp pages. But we cannot test presentation layer.

How do we solve these pr oblems?

Few senior developers and overall architecture: We hire three senior programmers; they have several years' experiences. One of them is architector in his previous job. They don't know XP at the beginning, but they know design very well. Now, whenever we start to do task, to add new feature in existing system, these experience programmer really do a lot help to simplify design, refractor legacy code. We feel coding better now.

Modifications to the overall architecture: We adapt J2EE (java 2 enterprise edition) architecture, and make clear distinction for each layer. We also make the name convention for each layer. Now, we are slowly recapturing the existing application to this goal architecture.

Stories too big: when stories are too big, we have the feeling that we don't understand the user requirements very well. So we go to end-user, talk with them, try to identify the requirements. Now, we do find that stories can be split into several small stories if we understand the user requirement well.

Prioritize stories badly: now, we start with the most important story first, split it into several small tasks, programmers will pick these small tasks and works in pair. So we ensure the most important tasks done first. But we still have one problem: if there are two many dependence between the tasks, we cannot distribute it to different pairs.

Problems finishing stories: We give deadline to business people; ask them to provide the necessary document before certain date. But they don't respect this deadline very much. So we have to force them to do it.

Pair programming: we do pair programming a lot now, because we really get the benefit of leaning knowledge from other programmers. Coding become better and better.

Presentation layer Test: We ask end-user to do the test. But it is really not enough. If there are some tools which can be used to test Presentation layer, it really will be great helpful.

Deadline pressure: We honestly tell business owner that we cannot finish the story before the deadline, ask them to drop some stories. We really need coverage to do it, because business owner don't like to hear it.

All the java classes and methods have their test case. Code quality is ensured.

We prioritize tasks to ensure the most import ones will be finished before deadline.

Prioritize tasks: developers have a clear idea about the things to focus on

Speed up the learning.

We do better information from business people, because we force them to prepare themselves for the planning meeting. We force them to get more organized, because they can only ask stories in the beginning of cycle.

We organize 10'oclock meeting. This implies that we always know what the other people are working on, and what problems they encounter.

We have automated a lot of the build and release procedure

Refactoring is seen as an important part of the development process, instead of a loss of time.

5 CONCLUSTION

We already enjoy a lot of benefit from implementing XP in our company, but we still encounter a lot of problem. It is a long way to go.

REFERENCES

1. Extreme Programming Explained (Kent Beck 2000)