Not So eXtreme Programming: Agile Practices for R&D Projects

Roberta Coelho
Laboratório de Sistemas Distribuídos
UFCG-DSC, Brazil
55-83-3310-1365

Esther Brasileiro
Laboratório de Sistemas Distribuídos
UFCG-DSC, Brazil
55-83-3310-1365
esther@dsc.ufcg.edu.br

Arndt von Staa
Computer Science Department
PUC-Rio, Brazil
55-21-2540-6915
arndt@inf.puc-rio.br

Computer Science Department
PUC-Rio, Brazil
55-21-2540-6915
roberta@inf.puc-rio.br

ABSTRACT
Though software projects can benefit from XP practices, not all projects can directly adopt them. Some practices have to be tailored to contexts specific to the projects. This paper describes the road followed when tailoring XP to R&D projects, in a time scale of 2 years. We describe our major challenges and the way we have solved them.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management --- *software process models*, *programming teams, software quality assurance;*

General Terms
Management

Keywords
Extreme Programming, process tailoring, XP practices, XP roles, R&D projects.

1. INTRODUCTION
There has been a continuing growth in the number of collaborative projects between universities and industry in the present decade. Many companies are not just sponsoring university research, but they are also considering efforts towards exploratory research, in conjunction with academic institutions. Under such collaboration between industry and university, everyone wins: *university researchers* amplify their knowledge and, potentially, turn research projects into commercial products without leaving academia; and *industry* accelerates research in non-trivial areas by conducting research concurrently in both the labs and within the company. In this paper we use the term 'R&D project' to refer to such projects.

The projects developed through this collaboration are characterized by: (1) technical challenges inherent to complex problem domains; (2) vague and minimum initial requirements which change often during development; (3) software development is done in academic environment - as a consequence, most of the development team is composed by undergraduate students with short experience in "real" software development; (4) lengthy time interval until the project is converted in a commercial product and reaches the market; (5) sometimes, there is not a user for the product to be developed.

Since some outcomes of these projects turn into commercial products, universities are interested in developing high quality software to meet industry needs. As is known by the software engineering community, a well defined software development process maximizes the chances of developing software according to quality specification. But the methodologies, conceived so far, cannot be directly adopted by a R&D project, due to some of its characteristics. Those methodologies should be tailored to this specific environment

The Distributed Systems Laboratory (DSL) of the Computer Science Department at the University of Campina Grande is made up of 41 members (18 undergraduate students, 8 MSc students, 5 DSc students, 5 research assistants, 2 engineers and 3 professors). DSL main software projects started two years ago. Our laboratory carries out software projects in research domains such as: grid computing, software-hardware fault-tolerant systems and process optimization in oil industry. The deliverables of these projects are both prototypes and products based on research results each comprising around 10-20,000 lines of code.

This short paper is structured as follows. Section 2 points out our experience tailoring XP to R&D projects; in this section we propose additional practices and roles to complement XP in order to make this methodology more suitable for R&D projects. In Section 3 we summarize our experience and address future work.

2. Defining Agile Practices for R&D projects
We had to overcome some obstacles in order to tailor Extreme Programming methodology to the context of R&D software projects. We needed to adapt well known roles, such as tester and application architect, from plan-driven methodologies [WC03] to an agile context. We also had to define new roles and practices to substitute or, sometimes, complement XP practices. The next section describes these roles and practices in detail.

2.1 Tester Role
According to XP methodology, every programmer should play the role of a tester. We agree with Kent Beck [Bec00] when he says that developers should have test skills. However, we think that developers should develop just unit tests. The acceptance tests should be implemented by a role resembling that of the client. Programmers naturally assume that, in general, "things work".

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

Testers, on the other hand, are *descendants of Murphy* and assume, in general, that things "don't work right" [CH02]. Testers live in a world where, no matter how many things are working right, some things always are not. We consider this world is similar to customer's world.

We chose one of the developers to play the role of a fulltime Agile Tester. An Agile Tester should spend time: (i) developing and running system tests; (ii) helping customers to write user stories and to define acceptance tests (iv) and automating those tests whenever possible – often, this task is more complex than developing a system functionality itself. Though this new role is useful to any software project it is *vital* to the success of R&D projects, because a set of rigorously defined tests encourage developers to incorporate new research ideas without the fear of introducing new bugs.

2.2 Application Architect Role

Agile methods emphasize that [CH02]: "the team knows best, so all decisions should be left to them.". However, if the project aims at complex problem domains – as most R&D projects do – a special attention should be given to the system architecture. Otherwise, if system architecture is neglected such projects, development problems may occur, from little code reuse to unrecoverable architectural problems and consequent project collapse.

Since XP instruments fall to avoid such problems, we supplement our process with the role of the application architect. He/she should be responsible for: (i) implementing the initial system architecture, based on the system initial stack of user stories as; (ii) maintaining system consistency and the interface compatibility between the system main modules. As a consequence, developers would be free to focus only on designing and implementing and unit testing the user stories allocated to them.

2.3 Customer Proxy Role

Using customer feedback to promptly check decisions and developments is a key principle of agile development processes such as XP. However, very often the customer is not available as part of the team. To overcome this major problem, in which the customer is not available full-time onsite, some development teams use to contact customers by telephone or e-mail, or schedule physical meetings when necessary. Although this can be efficient for software for which the problem domain is well known, it is not adequate for complex and imprecise applications domains, such as research outcomes. Since in this kind of software there is a great deal of uncertainty with respect to the requirements.

In all projects carried out by DSL we faced a situation where we did not have an on-site customer. To deal with this drawback, we split XP Customer role into two [Fin01]: customer "proxy" and actual customer. The customer proxy was an expert in problem domain who worked in a joint learning process with the development team - inexperienced undergraduate students - to understand the system requirements and to translate them into the software. The role of actual customer was played by projects sponsors; they should attend monthly iteration planning meetings to define the priority of each story and perform the final acceptance tests on delivered products.

2.4 Frequent Design Meetings

Although the System Metaphor XP practice was useful to make clients understand how the system worked, it was not sufficient to make the developers understand the overall functionality of the system, when it was associated to complex problem domains. To communicate the system's main abstractions and functionalities to developers, we used the system architecture, which evolved continuously during project iterations.

A group of senior developers (playing the role of application architects defined in Section 2.2) were responsible for assembling the first version of the architecture which was refined and evolved during *frequent design meetings*. Every developer ought to attend those meetings in order to understand, discuss or suggest changes in system architecture. These meetings happened every time a team member proposed a modification of the system's architecture. These meetings were extremely important to the success of the R&D projects carried out by DSL, since through this practice every developer got used to the system's main components and functionalities.

2.5 Metrics to Reflect Projects Health

We have used a set of metrics to evaluate R&D projects health. Some of them are the following: (i) the number of failures detected during client tests; (ii) the number of automated acceptance tests versus the number of user stories; (iii) the number of classes versus the number of unit tests.

Since, at *specific moments* along an R&D project, just few tasks result in software increments – due to the fact that some software increments depend on previous medium term experiments and researches. We realized that the software-based metrics, listed previously, were not sufficient to reflect the health of an R&D project. Hence, we have also defined a set of software-free metrics to reflect projects health: number of internal reports, seminars, articles, dissertations and thesis generated along the project.

3. CONCLUSIONS AND FUTURE WORK

Research labs can not get overloaded with processes, but on the other hand they can not adopt processes which extremely focus on programming tasks. This article presents a way to solve such tradeoff: set of practices and roles tailoring Extreme Programming to R&D projects. This work does not just describe our experience in tailoring XP practices but also shows "how to" information, which are useful for anyone interested in adopting an agile methodology on an R&D project. We have several new projects and we continue to refine our findings as we better understand and characterize R&D environment.

4. REFERENCES

[Bec00]K. Beck, Extreme Programming Explained, Addison-Wesley, 2000.

[CH02]L. Crispin, T. House, "Why XP Teams Need Testers November 2003", Testing Extreme Programming, Published by Addison-Wesley, 2002

[Fin01]M. Finsterwalder, "Problems with customer involvement", Experience Exchange Workshop at XP2001.

[WC03]L. Williams, A. Cockburn, "Agile Software Development: Its about feedback and change",IEEE Computer, 36 (6), 2003.