Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project (PyPy)

Beatrice Düring
Change Maker, Järntorget 3, 413 04, Gothenburg, Sweden
bea@changemaker.nu

Abstract. This document describes the practices created,adopted and evolved in the PyPy project. PyPy is a hybrid project, combining the different aspects of Agile and Distributed Development within the context of an Open Source community being partially funded by the European Commission through the 6th Framework Program. Influences and adaptions of techniques such as "sprinting" has been a core balancing act for the project since its inception. “Sprints” in the Python community differs from the Scrum version of sprints and in this paper we will present how this evolved agile method acts as a primary method of quality assuring the aspects of distributed and dispersed workstyle of the PyPy project and insures an ongoing interaction with the Open Source aspects of the project.

1. Introduction

There are different methodologies and practices in use in PyPy – such as Agile and Distributed development, F/OSS culture and the management practices in use in EU-projects in the 6th Framework Program. It should be noted that most of the techniques in use in the PyPy project evolved into practice inspired by success stories from other projects in the F/OSS community and "word-by-mouth". It has rarely been the case that methodologies have been researched and applied through formalized decision procedures in the PyPy project. Rather the approach has been that of trial and error and customizing certain practices to fit the needs of the project when actual need arose.

PyPy has the goal of implementing a highly flexible and fast Python implementation written in Python. The project received EU-funding 1 December 2004 and will continue for two years. The partial funded part of the project concists of 14 work packages and 58 deliverables. A consortium of 8 partners was constructed to fullfil the contract (for more information about the project, see http://codespeak.net/pypy, www.pypy.org).

2. Agile influences

Within the Agile development portfolio there are a multitude of techniques, tracing it ´s roots to the software experiences of the 70´s and 80´s. During the late 90´s the first agile development methodologies were published such as eXtreme Programming, Crystal and others and the collaboration between the instigators and authors resulted in the Agile Manifesto, published 2001.

The values stated in the Agile Manifesto (2001), stating the following central traits for Agile Development:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan

Research on two established Agile Development Methodologies, eXtreme Programming and Scrum shows large similarities between practices used in PyPy and practices advocated in these methodologies, although there are also crucial differences based on the unique environment of the PyPy project.

In Scrum the following similarities can be found with the PyPy project regarding practices and processes - the key one being "sprints" [1]. Although none of the roles in Scrum or documentation such as Product Backlog and Sprint Backlog are implemented in PyPy sprints. (see more on this in Section 3. Sprint Driven Development).

In eXtreme programming the following practices can be found which are also employed in the PyPy project:

* simple design
* testing
* refactoring
* pair programming
* collective ownership
* continuous integration
* coding standard
* just rules

The PyPy project have since it´s inception been "test-driven" and has employed automated test suites for language compliance tests (Python) and unit tests. This test framework together with an extensive "coding style" guide (covering style of code, style of tests, naming conventions etc) and version control support (Subversion) created the platform that allows for continuous integration into the code base. During sprints “pair programming" is used systematically - not only between core developers sharing an interest in a specific task but also for mentoring newcomers by pairing them with core developers.

The aspects of "simple design" can be found within the Python community (Zen of Python) as well as being supported by the iterative approach being used within PyPy (iterations from the end of one sprint until the end of the next sprint - ca 6 weeks). Some PyPy-specific rules regarding design and testing such as focusing on rapidly achieving functioning semantics and concepts and then, during refactoring focus more on optimization of the working code.

As for the aspects of "collective ownership" and "just rules", the PyPy development process is open for anyone who is interested in participating:

- The sprints are open for any developers interested in PyPy and Python (although experience as well as costs could be limiting factors).
- The automated framework for testing and version controls allows for a more relaxed approach regarding distributing commit rights to newcomers.
- The open and transparent communication in the development process (on line via mailing lists and IRC as well as during sprints)
- The accessability of the core developers for answering questions and mentoring (on line via mailing lists and IRC as well as during sprints)
- The weekly synchronization meetings via IRC, open for all interested developers to participate
- The documentation and tutorials available on line

These are all key factors, creating and maintaining an atmosphere of "collective ownership". This has also been crucial for evolving the community of PyPy from a few core developers to almost 350 subscribers to the development list as well as increasing the amount of developers with commit rights to access and make changes to the code base from a few core developers to almost 50 people (during the period of 2003 to 2006).

Some of the practices in eXtreme programming have created challenges in the PyPy development environment:

* small/short releases
* 40 hour week
* on site customer
* open workspace
* pair programming

The shared denominator regarding these challenges is that they in most cases are tied to the fact that PyPy is working distributed/dispersed as well as agile. During sprints the work style is both developer-driven, self organized as well as collaborative (pair programming and open workplace). Between sprints this process remains developer driven and self organized but the open workspace shifts into virtual workspaces.

If the community can be viewed as the actual customers (developers interested in a flexible and fast Python implementation, written in Python) then there is constant communication regarding prioritized functionality in current iterations and upcoming ones (both during sprints and in between sprints – on line). Due to both the community interaction as well as the continuous integration of code (as it is being written) there have only been 3 major releases in the PyPy project during the period February 2003 and October 2005.

The reason for having larger releases and so few during this period was that PyPy is a language implementation project (not application level) and this created the need to reach a "stable" platform (release 0.6, May 2005). After this was achieved two more releases followed quickly (release 0.7 August 2005

Aspects such as process terminology found in eXtreme Programming are not used in the PyPy project (planning game and metaphors) as well as the phases and roles specific to XP.

A open question regarding the comparison of practices in eXtreme Programming and those employed in the PyPy project is the reference to Kent Beck´s focus on teams being situated physically close in order to facilitate understanding and communication. This is an non-negotiable core aspect of XP, although Beck himself states that you might still be working geographically distributed and XP-style if it concerns "two teams working on related project with limited interaction" [2].

In the case of PyPy this is made more complicated because not only are the core developers working distributed, sometimes in pairs of two at the same location - they are also working dispersed - as is the rest of the PyPy community. The main strategy in PyPy to handle this challenge and risk to the development process is to sprint systematically, using sprints not only for iteration purposes but also to provide an accelerated and collaborative physical practice.

The question, whether this sprint driven approach in a distributed F/OSS team still would be considered as being within the scope of eXtreme Programming, is an open one and should be studied together with other aspects of hybrid practices evolving around Agile, Distributed and F/OSS teams.

3. Sprint Driven Development

PyPy first started during a one-week meeting, a *"sprint"*, held at Trillke-Gut in Hildesheim February 2003. The sprint was inspired by practices used by other Python projects such as Zope3. Originally the sprint methodology used in the Python community grew from practices applied by the Zope Corporation. Their definition of a sprint was: *"two-day or three-day focused development session, in which developers pair off together in a room and focus on building a particular subsystem"* [3]. Inspired by practices such as pair programming in eXtreme Programming sprints were first used within the commercial work and later tried and used within the Open Source context around Zope development. There seems to be no specific sources relating the Zope/Python version of sprinting to the terminology used in Scrum, signifying an iteration around a specific increment – lasting up to a month [4].

The Zope sprint approach focuses indeed on just writing code and has one “formal” role tied to it – the role of the “coach”. The coach prepares the content of the sprint and manages and tracks the work during the sprint. Tutorials are done during the first day and the suggested limit of people is to be no more than 10 people participating during a sprint.

The method evolved rapidly and sprints done in connection to conferences were more “open” and tutorial oriented, as opposed to sprints were only experienced developers in the Zope domain participated. Sprinting spread through the Python community and

today almost all Python projects in the Python Open Source community sprint at least once every year.

Sprinting up to a week became the initial driving factor in developing the code base and the community/people around PyPy. Sprints gave the opportunity to both help, participate and influence the ideas within PyPy. PyPy sprints was then as now a developer driven effort and the role of coaches are not in use in PyPy. Sprint preparation and planning as well as the actual organizing rotated between the developers, using their contacts and networks to identify locations and facilities to sprint in to as low costs as possible for both travels and accommodation for the sprint attendants. Already from the start the strategy to travel and sprint, visiting different local communities and "recruiting" contribution was a conscious one – also for the reason of "justly" distribute the load of travel costs in the developer community.

Why did PyPy choose sprinting as a key technique in the beginning of the project? It is a method that fits distributed teams well because it gets the team focused around visible challenging goals while working collaboratively (pair-programming, status meetings, discussions etc) as well as accelerated (short increments and tasks, "doing" and testing instead of long startups of planning and requirement gathering). This means that most of the time a sprint is a great way of getting results and getting new people acquainted - a good method for dissemination of knowledge and learning within the team.

References

[1]"Agile project management with Scrum", Ken Schwaber, Microsoft Professional 2004
[2] " eXtreme programming explained", Kent Beck, Cynthia Andres, 1999
[3]http://www.zopemag.com/Guides/miniGuide_ZopeSprinting.html
[4]"Agile project management with Scrum", Ken Schwaber, Microsoft Professional 2004