IST FP6-004779

PYPY

Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community

STREP

IST Priority 2

D14.5 Documentation of the development process and "Sprint driven development" in particular

Due date of deliverable: March 2007

Actual Submission date: March 30th, 2007

Start date of Project: 1st December 2004 **Duration: 28 months**

Lead Contractor of this WP: Change Maker

Authors: Beatrice Düring (Change Maker)

Revision: Final

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level: PU (Public)

Revision History

Date	Name	Reason of Change
2007-03-02	Beatrice Düring	Redoing structure based on prior notes
2007-03-13	Beatrice Düring	Finalize all parts
2007-03-19	Beatrice Düring	External review input from James Shore
2007-03-19	Beatrice Düring	rewrite theme and conclusions
2007-03-22	Beatrice Düring	Final external language review done
2007-03-27	Beatrice Düring	Rewrite section PyPy after internal review
2007-03-29	Beatrice Düring	Polish format errors after pdf review
2007-03-30	Carl Friedrich Bolz	Add statistical diagrams
2007-03-30	Carl Friedrich Bolz	Publish Final version on the Web Page

Abstract

This document describes the practices created, adopted and evolved in the PyPy project. PyPy is a hybrid project, combining different aspects of Agile and Distributed Development within the context of an Open Source community. It is partially funded by the European Commission through the 6th Framework Programme.

Influences and adaptions of techniques such as “sprinting” have been core balancing acts for the project since its inception. Searching the neighbouring communities of Python, other OSS projects and Agile communities for best practices in order to tailor the process best suited for PyPy has been an important strategy for the project.

The challenges identified and the practices employed in PyPy to mitigate these challenges are topics that the project aims to disseminate to external parties, such as other OSS communities, companies, researchers and organizations in order to support the ongoing refinement of the methodologies of Agile and Distributed development.

The document is thus structured in the following way: in the background section we present the history and objectives of the EU-funded project in order to provide context. In the following sections we present the actual sprint driven methodology and its various supporting practices. Finally we present and analyze what aspects of sprint driven development are agile and if indeed it could be named an agile methodology suited for distributed environments. In a separate annex we present metrics on how the F/OSS PyPy community evolved and was impacted by its partial EU-funding.

Purpose, Scope and Related Documents

This document serves to document the findings regarding practices being employed in the PyPy project.

The report is written to target industrial/commercial entities and F/OSS communities in order to disseminate the experiences and evolved practices that has arisen from the PyPy project.

With this purpose in mind the authors have strived for a practical, hands-on presentation of the methodology, giving the report more of a nature of an “experience report” rather than a “research paper”.

This report focuses on the development and project process of the PyPy EU-project. This means mainly covering the EU-funded period (1st of December 2004 - 31st of March 2007) while referencing back to the period before the EU-funding (February 2003 - November 2004).

This report includes references to technical aspects of the development process and evolution of the PyPy platform but is not a technical presentation per se.

In order to better understand the project's context during the period covered we recommend reading the following documents:

- 14.1 Report_About_Milestone_Phase_1 [D14.1]
- 14.3 Report_About_Milestone_Phase_2 [D14.3]
- 14.4 Report_About_Milestone_Phase_3 [D14.4]

Contents

1 Executive Summary

The methodology objective of the PyPy project is to showcase Sprint-Driven Development as an agile methodology, providing a dynamic and adaptive environment, suitable for co-operative and distributed development. We can now summarize our results and show that the practice of sprints was indeed the primary means for providing a dynamic and adaptive environment and ensure project success. Especially when this environment meant balancing the needs of the F/OSS community of PyPy and Python and the plan-driven requirements of the European Commission, working distributed and dispersed.

One important insight is that Sprint-Driven Development is not a methodology, but rather a practice, and in general a practice is not inherently agile. Sprints are very suitable as core practices for agile projects that are forced to work distributed and dispersed, suitable for both commercial and F/OSS projects. What we show is how Sprint-Driven Development was implemented in an agile manner within the project context of the PyPy project.

The continued evolution and growth of the PyPy community can be seen in the metrics (see Annex 1). This is a concrete way of verifying the success of Sprint-Driven Development because these metrics represent the increased interest of the community – meaning the customer. The core steps of Sprint-Driven Development represent an iterative and incremental cycle of four steps with supporting additional practices. In this report we provide the necessary information needed in order to be able to adopt the practice of Sprint-Driven Development. We show extensive references to tools and infrastructure supporting sprints as well as real documentation examples.

After the EU-funding is over a new phase in the PyPy community starts. The community have initiated discussions on how to informally organize coordination and direction of PyPy development with the focus of a continued lively community. Likely concrete effects during this non-funded near future is that the frequence of sprints will slow down – maybe not more than four sprints per year will be organized. At least one of these will likely be organized around larger Python conferences such as EuroPython in order to make it easier for newcomers to attend the sprint.

Another important aspect already solved is that the development infrastructure will continue to be hosted in the current environment for now (thanks to community contribution). This ensures that existing and established communication channels and software infrastructure can be used even without EU-funding.

2 Content Sections

2.1 Introduction

2.1.1 History of the project

The concept and ideas regarding PyPy started as email discussions via mailing lists late 2002 between developers in the Python community. Inspired by but also frustrated with the results of two successful Python implementations, Psycho and Stackless, the ideas of a highly flexible and fast Python implementation purely written in Python was born.

The architectural discussions as well as the actual first community effort around PyPy started during a sprint (one week programming session) in Hildesheim, Germany, February 2003. Mailing lists, code repositories and automated framework for testing as well as a developer web were also kick-started at this time. The amount of developers involved during this start-up period, subscribing to the development mailing list, grew from 0 to 144 subscribers (2003-02-28).

During the summer of 2003, at the third sprint, understanding the efforts needed to achieve the visions and goals of PyPy made the team realize the need of funding in order to be able to work full time on the project. The team identified that the objectives of the second call for proposals in the 6th Framework programme was fitting the

scope of PyPy and decided to explore this option. The search for consortium members for this EU-funded work of the community as well as work on the proposal continued during autumn 2003. The proposal was submitted to the European Commission 31 October 2003.

Formal negotiations with the Commission started in March 2004, and some changes to the budget were requested. Changes to the original plan were also presented in order to integrate the EU-funded part of the PyPy work with the community.

The project within the EU-funded consortium of PyPy started 1 December 2004 and will end 31st of March 2007.

2.1.2 Objectives, strategy and structure

The original vision of a PyPy interpreter was broadened and formulated into three main objectives: a technical goal, i.e. an open run time environment written in Python; a research goal, i.e. studying Aspect-Oriented Programming code generation and abstract interpretation by writing a Just-in-Time compiler generator.

The third goal, the method objective, is to showcase a novel software engineering process: Sprint Driven Development. This is an Agile methodology, providing a dynamic and adaptive environment, suitable for co-operative and distributed development.

The strategy of the project is to leverage the community of PyPy and Python through open and transparent communications and working style. The challenge has been to implement this strategy not only in the F/OSS community part of the project but also in the partially funded consortium structure of the project. The structure of the project and the consortium needed to adhere to this strategy while also conforming to the actual requirements of the 6th Framework Programme.

The consortium consists of 8 partners (DFKI, Ab Strakt, Logilab, merlinux GmbH, tismerysoft GmbH, Change Maker, Impara GmbH and Heinrich Heine Universität Düsseldorf) and 4 physical person partners (Laura Creighton, Richard Emslie, Eric Van Riet Paap and Niklaus Haldiman). The result of the 28 month's of funded project consists of 14 work packages and in total 30 deliverables.

The work in the consortium is coordinated between the partners via consortium meetings; issues and tasks on consortium level being managed (prepared and implemented) by a "Management team". The technical development and coordination and integration with the non funded parts of the community is being managed by the "Technical Board", consisting of the PyPy core developers instigating the project.

For more information about the consortium structure and the experiences gathered during the three phases of the project please see the milestone reports [D14.1], [D14.3], [D14.4].

2.2 What is Sprint-Driven Development?

First we need to define what we mean by sprints. A sprint, as they are implemented within the Zope community, is a:

> "...two-day or three-day focused development session, in which developers pair off together in a room and focus on building a particular subsystem. Tres Seaver at Zope Corporation (ZC) launched the concept of Sprints, based on ideas from the Extreme Programming (XP) community, for the development of Zope 3." [ZOPE].

However, sprints as they have been implemented within the PyPy community are seven days long coding sessions with a breakday included. Many Python projects use sprints with a variant of the above mentioned definitions.

With Sprint-Driven Development we mean projects that systematically use sprints as planned parts of their development process. In the case of the EU-funded period of the PyPy project sprints were organized every 6-8th weeks during the duration of the project.

2.2.1 Origins of sprinting within the Python Community

Originally sprints used in the Python community grew from practices applied by the Zope Corporation in 2001. Was this practice adopted from the agile community? In one of the established agile processes, Scrum, the term sprint occurs, in this case meaning the month long iteration of work being focused on [SCRUM].

When asked about the origins of the Zope sprints from 2001 Tres Seaver of the Zope Community, one of the instigators of the sprint practice, as it is used in the Python community, says the following about the connection to agile influences:

> "The motivation was to begin using as much of XP as would fit to accelerate Zope3 development. Given the different development culture of the Zope community (framework-centric, distributed, no "business user" present), some of the XP mantras /practices couldn't be done directly. We decided to try to do "as much XP as possible" for short, highly-focused sessions, with all active committers collocated."[TROUBLE].

So the origins are not related to Scrum. It was rather:

> "... because of the fact that we weren't doing XP in its proper sense, which is more like running a long-distance race, where pacing oneself is critical. Instead, we were pushing as fast as possible in order to maximize the benefit of limited face time. Of all the XP practices, the one we gave shortest shrift to is "no overtime": as sprints often go far into the evening (or even the wee hours of the morning." [TROUBLE].

Typical features of a Zope Sprint [ZOPE]:

- it is 2-3 days long
- it is facilitated by a coach, leading the session
- it involves pair programming
- Zope sprints can be of two different kinds: sprints open for anyone interested in attending, or sprints requiring experience in the subject matter
- it should not involve more than 10 people (although that has been the case several times)

Since then many Python projects use sprints in connection with major Python conferences and then sprints usually last 3-4 days. There are also cases of companies involved with the Python community arranging week long sprints and sometimes two week long sprints several times per year [SPRINTS].

In the case of PyPy sprints were from the start 7 days long with a breakday included. During the non-funded start of the project sprints were arranged when core developers felt the need and had the opportunity to arrange sprints. When the funded period of the project started sprints were planned for in a systematic manner with the goal of sprinting every 6-8th week. In total this meant that 14 sprints were planned and budgeted for during the 2 year project period. This systematic approach to sprinting is to our best knowledge unprecedented and the experiences of the PyPy project could therefore provide unique insights to others interested in trying out a similar approach.

In the following two sections we will describe the steps needed in order to implement Sprint-Driven Development and also the supporting practices which need to be taken into account when wanting to try out sprints as a practice. A more contextual description of how PyPy implemented Sprint-Driven Development and how this practice was combined with plan-driven requirements of EU will be presented in the section: "PyPy – a hybrid project: Agile, Distributed, F/OSS and EU-funding" further below.

2.2.2 A presentation of steps and practices within Sprint-Driven Development

Sprint-Driven Development is essentially an iterative and incremental approach based on four steps; planning the sprint, starting the sprint, running the sprint and closing the sprint. These four steps cover the iteration from start to end and a cyclic repetition would mean that the next iteration starts following the same pattern. A finalized iteration can be viewed as a full increment.

All steps but the first one, planning the sprint, take place when the group is co-located. During the entire time the group produces working code. There is no release scheme tied into the practice (although that could easily be the end result of a sprint and then the result of the iteration). There is nothing stated in the practice about the lenght of the iteration. In the case of PyPy one iteration would (in almost all cases) not be longer than 8 weeks, spanning from the closing of one sprint to the closing of the next sprint, covering the distributed and dispersed period between sprints. A longer lenght of iterations would, in a dispersed team, be a potential threat to cohesion and progress in the development process. In a distributed team this risk will probably be less of a threat.

Planning the sprint The high-level goals and "topics" of the sprint are discussed via mailinglists, IRC-channels and/or wikis (see also "sync-meetings" below). The topics are chosen based on the need for progress in certain areas but also trying to match the interest and skillsets of the possible participants.

The "venue" and logistics are organized. It is important to verify that the venue is capable of supporting connectivity as well as electricity outlets. Facility aspects such as functioning (and comfortable) chairs and tables, whiteboards and projectors need to be taken into account. If cost is an issue among the participants it is also useful to have food/coffe/tea facilities available.

Planning the venue as well as "accommodation" and "travel" to the sprint (if the sprint is in a different city and/or country) is much easier done with a "local organizer" available. How to easily travel to the sprint venue (trains, busses, flights) as well as maps for finding the venue and some recommendations about accommodations available are typical examples of information gathered by the local organizer. Also information about currency, fares, some short background on the city is appreciated.

When logistics as well as topics are agreed on the team need to write a "sprint announcement" [SPRINTANN] (see here for an example of a sprint announcement in the PyPy project]. This is a good practice regardless of if it is an internal or external sprint and the primary purpose is to facilitate focus among the participants. If it is an external sprint – available for all interested parties to attend then the sprint announcement also has a marketing effect, spreading news about the event. It should also state the length and the dates for the sprint. If the sprint needs to be more than 3 days it is recommended to plan for a "break day". The break day is a good way to show the importance of rest and social activities as an informal way of spending time together.

The sprint announcement should be sent out on relevant mailinglists and published on a project related website at least 3-4 weeks in advance. The reason for this is to ensure that people will have time to book flights and plan their schedules etc.

It is recommended that people planning to participate "announce attendance" on related mailinglists or wikis and present themselves briefly and their specific interests. It is also useful to know when people plan to arrive and depart so the organizers have an idea of how to plan for the logistics. This also helps the team to orient themselves with whom they will be cooperating with during the sprint and will hopefully make collaboration easier.

Starting the sprint The sprint starts with a "startup planning session". Here the group will present themselves briefly and revisit the high level topics. These are readjusted to fit the current status of progress and the people present (skills, experience and interest) and are also broken down into a planning file [PLAN] (see here for an example of a planning file used in a PyPy sprint]. Then people pair up based on prior knowledge in the task, hopefully also matching their interest.

It often happens that people pair up in groups with 3-4 persons, one core developer and several "learners". Usually task allocation lasts for a day. If a pair completes a task they will go trawling for work in the planning file. During the start up planning session, which usually does not take more than 30 minutes, questions about

status, design discussion needs and clarifications can show up. If questions cannot be answered during this session (i.e being large of scope) people can pair up for discussions for a few hours before starting to write code.

If there are newcomers to several of the topics or the system in general a "tutorial" [D14.2] (for references to a sprint tutorial and videodocumentation of a sprint tutorial used in the PyPy project) can be held for one hour. This usually helps understanding of the architecture. Pairing up with experienced developers will generate a more in-depth insight into the details of a specific area of the code.

Running the sprint When the sprint is running "daily status meetings" are a good way to check the progress of the planning file and re-pair people. Daily status meetings usually do not take more than 15-30 minutes in the morning and they are also a good forum for stating blockers or problems that need attention.

When running sprints there are two ways of organizing the tracking of work. Some sprints use a "coach" [ZOPE] for "leading" and "directing" the work, other sprints have a "self-organized" team just doing daily status meeting. If one goes for the option of the self-organized team there is still a need for informal leadership from core developers and people experienced in running sprints, moderating minutes and stepping in to help resolve blockers.

One challenge of working co-located, having other team members and contributors "off location", is that the usual traffic on mailinglists and IRC-channels is reduced signicantly (see metrics on IRC-channel traffic during sprints in the PyPy project in Annex 1). This needs to be adressed - sprints are in their nature extremely productive and working "off location" means missing out on both the discussions and being able to follow progress easily. This risk is reduced by posting the daily planning results and the updated planning file together with a midweek summary, a "sprint report", for longer sprints.

Closing the sprint During the last day of the sprints it is useful to have a "closing session", summarizing status and brainstorming about high-level goals and topics for the next sprint. During this session the team assessess the planning for the venue (if changes) and who is the organizer (and local organizer) for the sprint. A "sprint report" [SPRINTREP] (see here for an example of a sprint report produced in the PyPy project] summarizes the technical results. The sprint report is sent out to relevant mailinglists, helping non-participants to get a perspective on what results were achieved and what the current status is.

The team splits up and starts a new iteration, working distributed and/or dispersed and planning the next sprint – the cycle starts again.

2.2.3 Supporting practices of Sprint-Driven Development

Test-driven development and automated test framework In order to facilitate working software at all times during the iteration test-driven development and automated test frameworks are an important supporting practice – please read PyPy 2.3 Testing framework, EU report [D02.3] for more information on this .

This quality-assures the period of the iteration that is distributed and/or dispersed. It also facilitates an open and transparent approach towards newcomers to the system participating in the sprints – reducing the jeopardy of giving commit rights and thus risking costly mistakes. Automated test frameworks also support rapid prototyping and refactoring – typical tasks done during sprints. A coding guide is also helpful in order to help people understand how tests are designed and how code should be written in order to be consistent.

Efficient version control support A tool equally important as an automated test framework is an efficient version control system (such as CVS, Subversion or bzr). Again the main purpose is to quality-assure work done during the sprint and work done when distributed or dispersed. When combining these two supporting practices you get cumulative effects that have "social" consequences – such as making it easier to allow new contributors commit rights, supporting openness and transparency as well as the ability to influence the development process.

Communication channels (mailinglists, wiki, IRC) Communication infrastructure is important for facilitating communication when distributed and dispersed. The sprint planning step can be organized more easily if there is support for IRC/chat communication - mailinglists and wikis are suitable for questions, decisions, informing about sprint announcements and sprint reports. Automated mailinglists for tracking changes to documentation

and code help both during the sprints for people "off location" to follow progress as well as the keeping track of the planning status during the time spent working distributed and/or dispersed.

Here publicly available IRC-logs and mailinglists archives help newcomers to get a better understanding of the history and evolution of work and prior sprints results. For more examples of useful supporting tools and infrastructure in a Sprint-Driven Development environment - please read PyPy 2.1 Development Tools and Website EU-report [D02.1].

Syncronization meetings Sync-meetings are weekly short coordination meetings between developers when the team is distributed and/or dispersed during the iteration. These 30 minute IRC-meetings serve as a weekly synchronization for regular discussions and integration of ongoing work and are also useful for planning the sprint.

Sync-meetings is a really useful complement that steer up work. They are helpful for maintaining cohesion when people work distributed and/or dispersed. The fixed timelength help keep the amount of topics low which makes it easier to maintain focus and attention.

The fixed format seem to be inspired by "daily Scrums", although this has not been verified [SCRUM]. All teammembers participating in the meeting present a very short statusreport and answer the questions LAST (last week), NEXT (their focus for the upcoming week) and BLOCKERS (if they are stuck and need help). This helps the team to track progress in an informal way while being distributed and/or dispersed. The round robin aspect makes sure that all team members voice their individual input. If blockers are identified the team tries to resolve them together or appoints people between meetings to solve them.

Agenda and minutes are posted to mailinglists together with the IRC-log in order to support openess and transparency – useful for people not being able to participate in the meeting [SYNC] (for an example of a sync-meeting protocol from the PyPy project).

2.3 Is Sprint-Driven Development Agile?

Why does this question exist and why is it worth answering? In Extreme Programming Kent Beck presents that working co-located (in the same room) is the main means for teams to facilitate understanding and communication. Such co-locatedness is a non-negotiable aspect of XP and agile methodologies. However, Kent Beck himself states that you can work XP style while still being geographically distributed if it involves:

> "two teams working on related projects with limited interaction" [XP].

Due to the F/OSS nature of PyPy and other F/OSS projects that use sprints the nature of work is rather more dispersed than distributed, meaning that individuals are working geographically distributed [BRIDG]. So the question is in its essence whether distributed and dispersed teams could be viewed as agile when they employ Sprint-Driven Development?

To highlight this dilemma we can quote Jim Shore (Agile Alliance 2005 Gordon Pask Award winner) in the IEEE Software article "Sprinting towards Open Source Development" [IEEE] when talking about PyPy and Sprint-Driven Development:

> "One of the big challenges of F/OSS projects is not that they´re F/OSS – what is relevant is that most F/OSS projects have people working at long distance and not necessarily able to give it their full attention. That´s a big challenge for keeping everybody on the same page and I think that sprints are a fantastic innovation for making that happen."

When commenting on whether Sprint-Driven Development in general is agile from an Agile Manifesto point of view he says:

> "I think it would be a mistake to say it is a combination of agile and open source development../I think it has applied one of the ideas of agile development to the very difficult problem of distributed software development." [IEEE]

2.3.1 High level agility: values and principles

In order to answer the question whether Sprint-Driven Development as it is employed within the PyPy project is agile one needs to analyze Sprint-Driven Development from a high-level perspective. What high-level values and principles are guiding the PyPy development process and how do these match the high level-level definitions emanating from the Agile community?

The values stated in the Agile Manifesto [AGILE] summarizes the 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

One could easily argue, on this level, that all four "left-sided" statements have been guiding principles in Sprint-Driven Development as it was employed by the PyPy project, making it agile[1]:

Individuals and interactions over processes and tools: PyPy is the product of people within a F/OSS community choosing to collaborate and thus choosing practices such as sprints in order to facilitate face-to-face interaction even when the project was not funded by the EU.

To compensate for working distributed between sprints infrastructure was designed to ease interaction and by the same time providing open and transparent communication channels (mailinglists, IRC-channels, automated mailinglists summarizing commits and changes to the code and documentation as well as email archives and IRC logs, IRC sync-meetings etc).

One could summarize this as: Sprint-Driven Development, with its supporting practices, makes a distributed reality as agile as is possible.

Working software over comprehensive documentation: Here it might be argued that commercial and/or agile software projects could learn from successful F/OSS projects. Because of the dispersed nature of these projects the infrastructure set up to provide "working software" are great examples of efficient version control configuration management. In the case of PyPy this is supported by the py-test tool facilitating test-driven development to a largely automated extent [D02.3].

Customer collaboration over contract negotiation: In the case of PyPy EU is the "paying" customer, or rather the funding party. Many other F/OSS projects recieve partial or full funding. In all these cases the "real" customer is the community – representing people who use the product for professional, commercial and/or private reasons. The open nature of sprints as it has been adopted within the Python community (Zope3, PyPy, Plone etc) is geared towards customer collaboration; working together in developing working software.

Again in the case of PyPy heavy time-consuming contractual amendments were done in order to act on requirements and needs being expressed by the community around PyPy. The collaborative nature of the Python and PyPy communities involves the real customers in bug reporting, testing, benchmarking, validation of both code and documentation. It should be noted, though, that there is is no outspoken relationship geared towards providing business value nor dialogue concerning the matter of business value with the community in question.

To summarize: PyPy collaborated with the community and negotiated the contract with the Commission.

Responding to change over following a plan: Due to the technical nature of PyPy and in which manner the high-level functional and non-functional requirements had been written there was room for flexibility concerning what the community wanted to focus on. The various backends and frontends developed and discussed in the PyPy community were all results of ideas and changes proposed by the community.

Sprint-Driven Development facilitated this dialogue together with the Summer of Code and Summer of PyPy initatives[CODE]_,[PYPY]_. The PyPy Consortium chose to invest large parts of its management budget in order to make sure that community interaction and change was facilitated – to this date 5 contractual amendments have been done which in one way or the other was initated to facilitate change regarding process and content of the project.

[1] Nothing in these four core values states that working co-located is mandatory for matching the definition. However, co-located is the best solution in order to facilitate the values stated.

Besides looking at the Agile Manifesto another useful comparison for validating the agility of Sprint-Driven Development and PyPy is to look at the derived definition from the VTT Publication 478 "Agile software development methods - review and analysis" by Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta [VTT]. Their conclusion, based on analyzing research in the area, was that common to all the main agile methodologies were the following traits:

> ".. when software development is incremental (small releases, with rapid cycles), cooperative (customers and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last moment changes)." [VTT].

Development within the PyPy project is both incremental and iterative in its nature. Although only 6 "formal" releases were done during the EU-funded period the nature of the infrastructure (efficient accessible version control systems and automated testing suite) and sprints made sure that there was always working software available. One could view the results of a sprint as the released software within a specific cycle of increment – the truth is that each commit made provided a new release, and the formal releases was done for EU-deliverable purposes with a community "marketing" benefit tied to it.

The aspects of "cooperative" and "straightforward" could best be answered by quoting the socGDS team at Limerick University who in their paper summarizes their studies [LEARN]:

> "The sprint driven development methodology as it occurs in PyPy is interesting because, while it is a way of accelerating the development of written code, it also serves as a mechanism to expand the community and facilitate enculturation of its members."

By the nature of open sprints and participation one is not only achieving situational learning regarding the system and the software. One also participates in learning the practice. An important example of this impact is the fact that sprints started the PyPy community. This was inspired by core developers having participated in Zope3 sprints. The practice is by now well documented online and it is the hope of the author that this report is yet another useful documentation on how to adopt Sprint Driven Development. There is also videodocumentation available from the PyPy website [D14.2], produced in order to provide more creative and unorthodox ways of disseminating both technical and methodology aspects of the project.

As to "adaptive", sprints in combination with test-driven development supported by an automated testsuite are designed to accelerate development and allow for rapid prototyping, all done with the purpose of identifying need for changes and refactoring.

One final way of adressing the high level question of the agile nature of Sprint-Driven Development is to look at the research paper " What Lessons Can the Agile Community Learn from A Maverick Fighting Pilot?" by Steve Adolp, presented at the Agile 2006 Conference in Minneapolis [OODA].

In his paper he asks the question what it means to be agile. His answer is that it is a question of people and culture rather than if a specific agile process or agile practice is employed. Building on the theories of Colonel John Boyd´s OODA loop (Observe, Orient, Decide, Act Loop):

> "The agile methodologies are designed for operating in rapidly changing uncertain environments. But this does not mean that larger methodologies cannot be agile if they correctly match the environment" [OODA]

Building on this one could view the F/OSS community as a specific "environment" and Sprint-Driven Development as a means to create agility and a faster OODA response within that specific environment. Adolphe also says that:

> "A key lesson from Colonel Boyd´s theory is the importance of team culture to quicken the decision cycle." [OODA]

This also fits with what Alistair Cockburn explains in his paper "Characterizing People as Non-Linear, First-Order Components in Software Development":

> "The fundamental characteristics of "people" have a first-order effect on software development, not a lower-order effect." [PEOPLE]

These two quotes could be summarized as: agile minded people will create agile projects, (almost) regardless of project context. Seen from that perspective the team behind PyPy could definitely be viewed as agile.

Connecting the thoughts of James Shore and Steve Adolphe, it would seem that regarding the dilemma of defining what is agile and what is not agile the agile community needs to adress the "environment" that is distributed and dispersed. And if environments other than the always co-located one is percieved as not agile due to this specific aspect one might argue that the agile community have lost focus on their core values – "Individuals and interactions over processes and tools" [AGILE].

This does not seem to be the case though. The agile community have since 2002 adressed the issue how to adopt XP and other agile methods in a distributed environment. An example would be Scott Ambler who in his paper "Bridging the distance"[BRIDG] states:

> "Unimpaired communication between developers and project stakeholders is a fundamental aspect of agile software development, so is it possible to take an agile approach to dispersed software development? Yes, but it takes work."

This particular paper was written 2002 – does the agile community of 2007 share this view still? There are many experience reports since then that shows that not only is it possible to have agile projects working in a distributed environment – the agile community is doing so [SUN].

In order to better understand if Sprint-Driven Development would be considered agile a phone interview was done with James Shore [SHORE] as a follow up on the IEEE Software article mentioned above. In the interview he defined Sprint-Driven Development as a practice and as such a practice he would rapidly choose to employ when facing a distributed and dispersed eXtreme Programming project.

When discussing the "agility" of Sprint-Driven Development he suggested that because Sprint-Driven Development is a practice rather than a methodology it is not inherently agile. There need to be other core practices and values in place for a project to call itself agile than just Sprint-Driven Development. The prioritized practices at the core of agile values according to Shore are the ones focusing on the customer relation, providing business value and the constant iterated focus of process improvement.Whether a project is distributed or dispersed (the "environment") has less impact on a projects nature of "agility" than the availability and use of the prioritized core practices involving customer interaction and process improvement.

To conclude and answer the question: is Sprint-Driven Development agile the answer would have to be:

- Sprint-Driven Development per se is not inherently agile because it is not a methodology. As a practice it lacks support for core agile principles and practices and can therefor by itself not be seen as an agile practice.
- Sprint-Driven Development is a practice that is very much suited for agile projects (XP, Scrum) forced to work distributed and dispersed and as such a practice to be highly recommended.

In the following section we will argue, based on this conclusion and the thoughts from Steve Adolphe, that the specific implementation of Sprint-Driven Development in the PyPy project was indeed an agile implementation.

2.3.2 PyPy - a hybrid project: Agile, Distributed, F/OSS and EU-funding

So far we have have focused on describing how sprints are done and in what way sprints and Sprint-Driven Development could be seen as an agile practice.

Another equally important feature was the actual development process, how the everyday development work was coordinated and done among the involved funded and non-funded developers. When describing sprints we have seen one dimension of the development work, how it was done during sprints when being co-located.

The 6-8th week period between two sprints was when the team was distributed. At full peak the team was in total ca 10-13 involved developers. During periods of the project, 2-3 people worked co-located between sprints which meant it was a mix of distributed but mostly dispersed development, single developers working from individual locations.

Usually developers logged in on the main IRC-channel for development work and issues related to PyPy, #pypy. The work hours on this channel is somewhat similar to those during sprints, around 10 in the morning until 19 in the evening (and later). The main bulk of the core developers shared the same timezone (CET) which also made communication and coordination via IRC easier.

This *virtual office* is the heart of the PyPy development process. Here involved developers meet up during the day to pair up through "shared editing sessions" on features of code development, to review the ongoing commits of code and testresults appearing, to discuss design problems or possible solutions, to bug report and bug fix, to informally discuss and distribute work and of course to socialize. A large time is also spent on mentoring newcomers that have questions regarding how to get started with PyPy or people that are starting to contribute to PyPy [PYPY].

In this forum the core developers of the Technical Board did not act based on their formal roles, they participated as individuals and their main work was code development, coordination and mentoring. Here as well as during pypy-sync meetings great care was taken to try to avoid EU specific jargong. Reason for this was to maintain the informal community interaction as independent from the formal structures of the EU-project.

The rough goals for the distributed period until the next sprint was what people worked on, informally discussing and breaking down the tasks and achieving them, testing and reviewing the progress. No formal distribution of tasks or resources was done - the bulk of the development work done in PyPy was *self-organized and peer-driven*. By that we mean that no formal roles or structures was involved in coordinating the work on this day-to-day workstyle[2].

On a regular basis discussions arise on the main development mailinglist, pypy-dev [D02.1]. Because the IRC-channel is the main forum the traffic on the mailinglist is not that high. Here people in the community present their testresults on running PyPy in various configurations as well as report bugs. Ideas for features and questions on whether PyPy will take a certain direction is typical for pypy-dev. These types of questions is the closest comparison to a requirement discussion, albeit loosely and informal in its nature.

Based on this dynamic day-to-day workstyle of online collaboration and discussions (IRC and mails), problems, issues and possibilities were identified and when necessary moved to the Consortium level by the Technical Board. As such the PyPy project was in a major sense technically driven and the main management tasks were to track project progress and results, to organise contract relevant decisions and to react to critical project events.

The challenge for the project management was to act more "operative" and "tactical" in this dynamic environment as opposed to overly relying on formal structures. It had to keep track of the technical progress according to the contractual objectives as well as to organize the Consortium related formal obligations and actions. All this had to be done with the goal of not handicapping the technical implementation with overly formal structures and approaches.

[2] As opposed to decisions made on which resources should be deployed within a certain task in a certain workpackage where specific partners where involved. The self-organized and informal way distributing work was more fitting to the dynamic background that was the commmunity but it also had its drawback from an EU-sense. The tracking of planned work and cost consumtion per partner did deviate both when it came to development work and management work which had to be explained and justified.

During the duration of the project, the management structure changed and evolved to better adapt to these needs. One main driving force was to rapidly shorten the reaction cycle of the project when it came to driving technical issues and changes to the formal Consortium level. The Technical Board played a key role here. Its members took care for assessing contractual technical work, determining problems and according actions while all the time being the main *user interface* towards the community. In order to facilitate the work of the Technical Board the management team had to evolve into a more open structure and focuse more on operative management issues.

This *tow-tiered* system of a Technical Board and a Management Team acting on two separate levels of the project worked, although it had to battle troubles and challenges that rise from coordinating a hybrid project such as PyPy (distributed,F/OSS and EU-funding.) The main success factor behind this was that a dynamic, informal and peer-driven development process was allowed to stay that way and by receptive technical management was to inspire a redesign of the formal structures of the EU-project. As such this means, as Cockburn and Adolphe states, that the project succeeded because of the "people factor" and because of the huge amount of work and energy core people was prepared to invest in order to make it work.

For a more specific information about structures and coordination activities used in the project - please see our EU-report "D01.2 Project organization" [D01.2]. For more context on how the project evolved during the time of the EU-funding, please read our milestone reports [D14.1], [D14.3] and [D14.4].

2.4 Conclusion

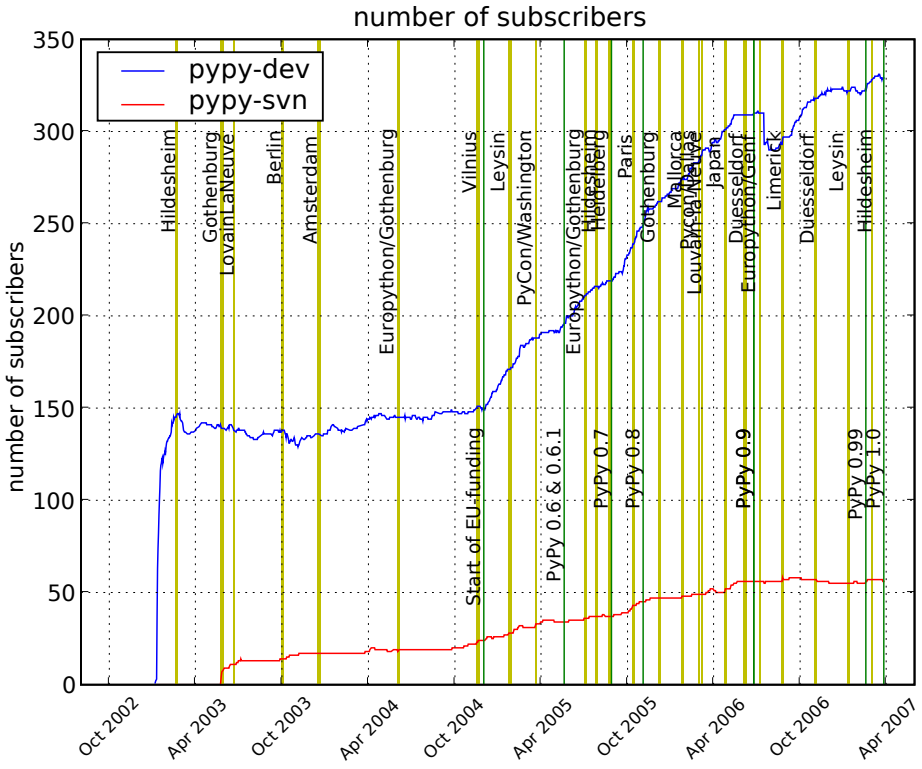
With our findings on Sprint-Driven Development we hope to show that during the progress of the project the team readjusted practices as well as actual processes in order to achieve useful and working results for the customer – the community. The practice of Sprint-Driven Development and the workstyle of the development process were affecting all levels of the project. They were also the primary success factor for managing to balance community interaction and demands with a plan-driven frame work.

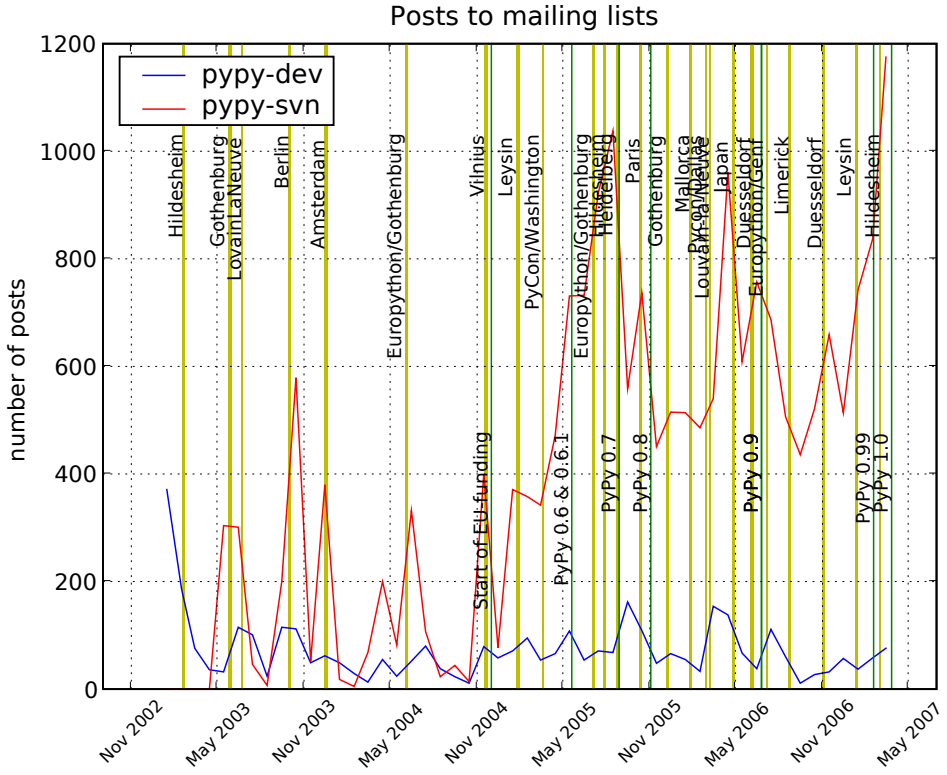
Although the practice of sprints and Sprint-Driven Development may not be inherently agile, in the PyPy project they were implemented in an agile manner. How come? Again we are referring to Adolphe [OODA] – in this unique context and "environment" the community took the role of the customer. Sprints were the *forum* for interacting, planning and adjusting scope together with the customer – all with the driving focus of providing working, useful software. The informal, dynamic and peer-driven F/OSS development process was the *culture* that made all this possible.

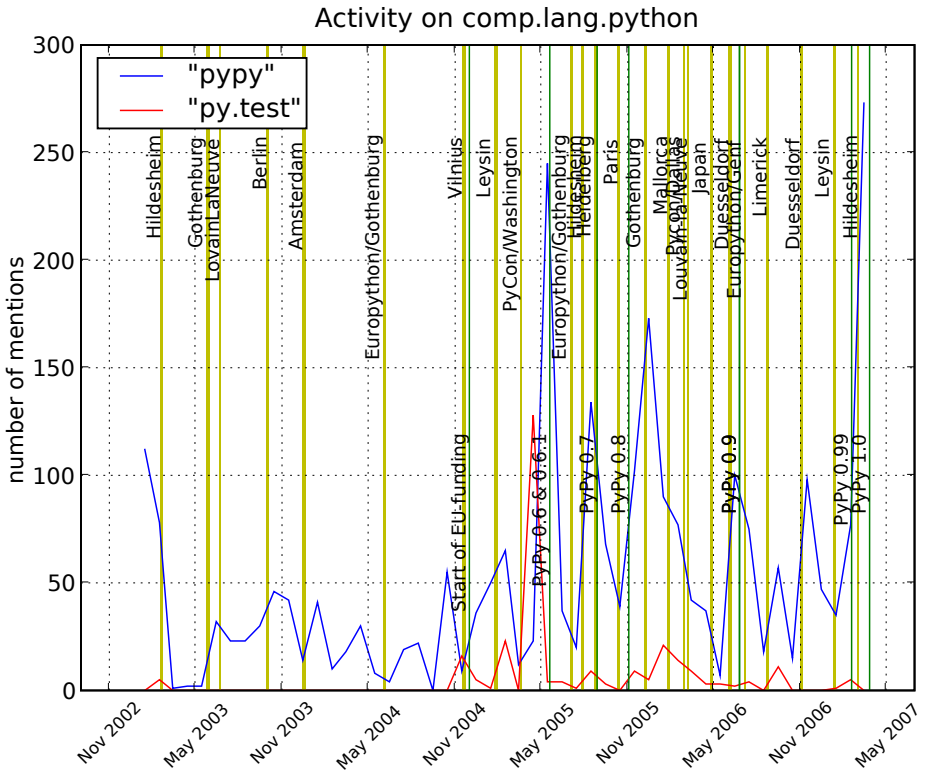
The F/OSS "environment" was combined with funding from a public body - this is what differentiates PyPy from other F/OSS projects that use sprints as a practice. The focus of the Commission is to support the needs of the Python community in order to strenghten the competitiveness of the European commercial IT sector. The common denominator for these different aspects, EU-funding and the F/OSS community, was the focus for providing working, useful software for the "customer"". The primary strategy for succeeding with this was the practice of Sprint-Driven Development which in this context (the development workstyle) were implemented in a fully agile manner. To conclude this reasoning we quote Adolphe [OODA]:

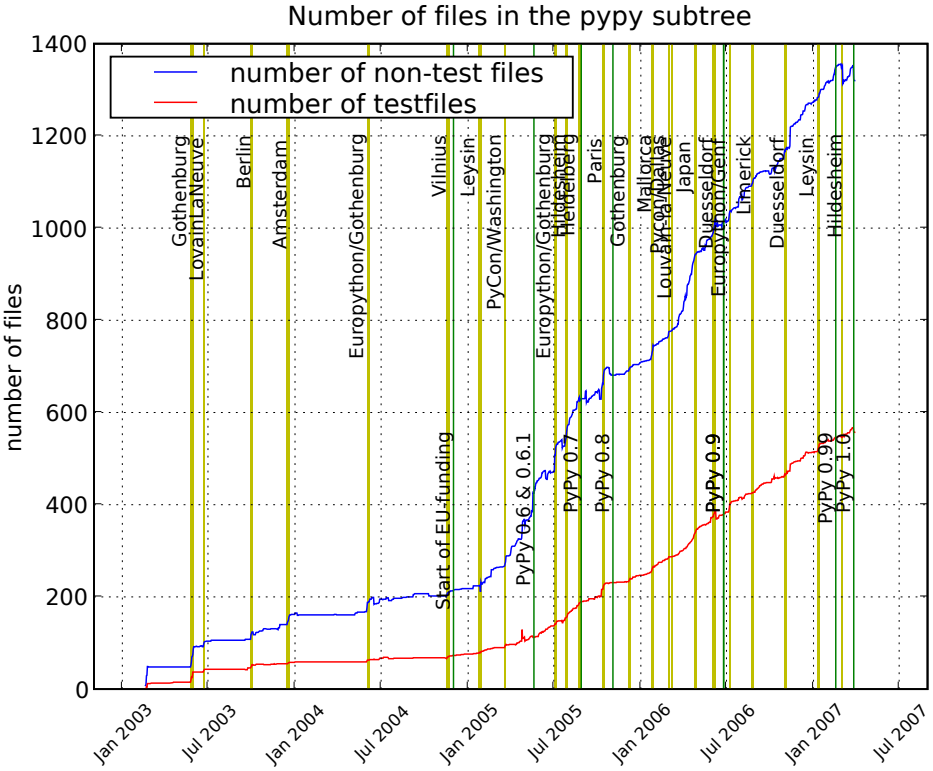
> "An agile project is one that works its OODA loop more quckly than changes can occur in the environment. A large slow moving project is agile if it can respond and shape changes in its environment in a timely manner. A small quick moving object may not be agile if it cannot stay ahead of and shape changes in its environment."

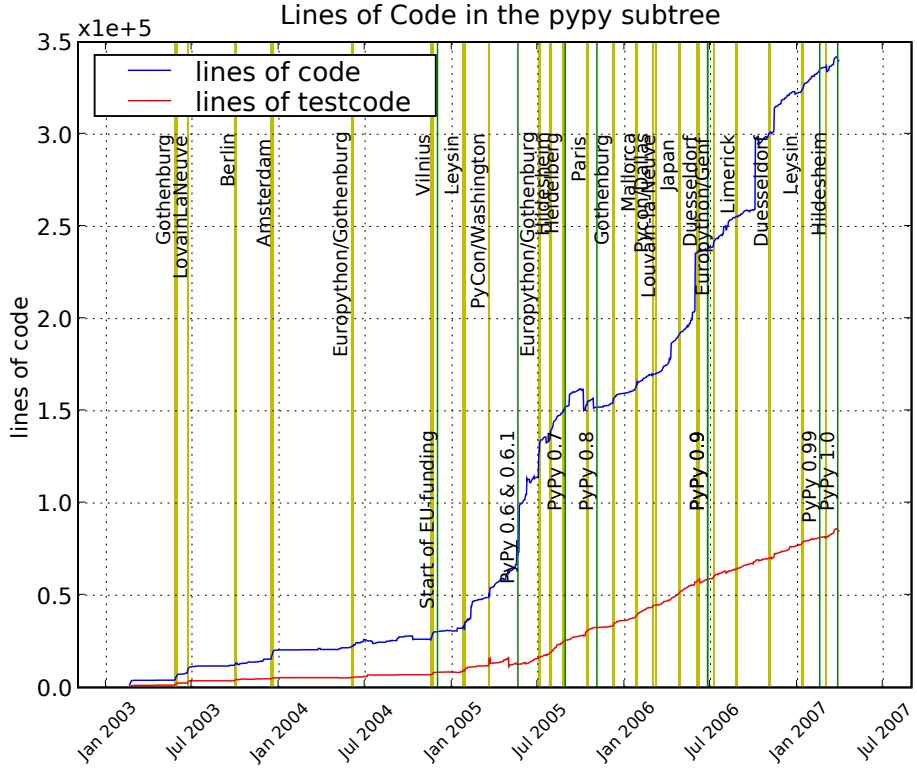
2.5 Annex 1: Metrics on the Community and the Project




Posts to mailing lists
pypy-dev
pypy-svn
number of posts
0
200
400
600
800
1000
1200
Hildesheim
Gothenburg
Louvain-la-Neuve
Berlin
Amsterdam
Europython/Gothenburg
Vilnius
Start of EU-funding
Leysin
PyCon/Washington
Europython/Gothenburg
Hildesheim
PyPy 0.6 & 0.6.1
PyPy 0.7
Paris
PyPy 0.8
Gothenburg
Mallorca
Pycon/Dallas
Louvain-la-Neuve
Japan
Duesseldorf
Europython/Genf
PyPy 0.9
Limerick
Duesseldorf
Leysin
PyPy 0.99
Hildesheim
PyPy 1.0
Nov 2002
May 2003
Nov 2003
May 2004
Nov 2004
May 2005
Nov 2005
May 2006
Nov 2006
May 2007


Activity on comp.lang.python
"pypy"
"py.test"
number of mentions
0
50
100
150
200
250
300
Hildesheim
Gothenburg
Louvain-la-Neuve
Berlin
Amsterdam
Europython/Gothenburg
Vilnius
Start of EU-funding
Leysin
PyCon/Washington
Europython/Gothenburg
Hildesheim
PyPy 0.6 & 0.6.1
PyPy 0.7
Paris
PyPy 0.8
Gothenburg
Mallorca
Pycon/Dallas
Louvain-la-Neuve
Japan
Duesseldorf
Europython/Genf
PyPy 0.9
Limerick
Duesseldorf
Leysin
PyPy 0.99
Hildesheim
PyPy 1.0
Nov 2002
May 2003
Nov 2003
May 2004
Nov 2004
May 2005
Nov 2005
May 2006
Nov 2006
May 2007


Number of files in the pypy subtree
number of non-test files
number of testfiles
number of files
0
200
400
600
800
1000
1200
1400
Jan 2003
Jul 2003
Jan 2004
Jul 2004
Jan 2005
Jul 2005
Jan 2006
Jul 2006
Jan 2007
Jul 2007
Gothenburg
LouvainLaNeuve
Berlin
Amsterdam
Europython/Gothenburg
Vilnius
Leysin
PyCon/Washington
Europython/Gothenburg
Hildesheim
Paris
Gothenburg
Mallorca
PyCon/Negative
LouvainLaNeuve
Japan
Duesseldorf
Europython/Gent
Limerick
Duesseldorf
Leysin
Hildesheim
Start of EU-funding
PyPy 0.6 & 0.6.1
PyPy 0.7
PyPy 0.8
PyPy 0.9
PyPy 0.99
PyPy 1.0


Lines of Code in the pypy subtree
x1e+5
lines of code
lines of testcode
lines of code
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Jan 2003
Jul 2003
Jan 2004
Jul 2004
Jan 2005
Jul 2005
Jan 2006
Jul 2006
Jan 2007
Jul 2007
Gothenburg
LouvainLaNeuve
Berlin
Amsterdam
Europython/Gothenburg
Vilnius
Leysin
PyCon/Washington
Europython/Gothenburg
Hildesheim
Paris
Gothenburg
Mallorca
PyCon/Negative
LouvainLaNeuve
Japan
Duesseldorf
Europython/Gent
Limerick
Duesseldorf
Leysin
Hildesheim
Start of EU-funding
PyPy 0.6 & 0.6.1
PyPy 0.7
PyPy 0.8
PyPy 0.9
PyPy 0.99
PyPy 1.0

3 Glossary of Abbreviations

The following abbreviations may be used within this document:

3.1 Technical Abbreviations:

AOP	Aspect Oriented Programming
AST	Abstract Syntax Tree
CPython	The standard Python interpreter written in C. Generally known as ”Python“. Available from www.python.org.
codespeak	The name of the machine where the PyPy project is hosted.
CCLP	Concurrent Constraint Logic Programming.
CPS	Continuation-Passing Style.
CSP	Constraint Satisfaction Problem.
CLI	Common Language Infrastructure.
CLR	Common Language Runtime.
docutils	The Python documentation utilities.
F/OSS	Free and Open Source Software
GC	Garbage collector.
GenC backend	The backend for the PyPy translation toolsuite that generates C code.
GenLLVM backend	The backend for the PyPy translation toolsuite that generates LLVM code.
GenCLI backend	The backend for the PyPy translation toolsuite that generates CLI code.
Graphviz	Graph visualisation software from AT&T.
IL	Intermediate Language: the native assembler-level language of the CLI virtual machine.
Jython	A version of Python written in Java.
LLVM	Low Level Virtual Machine - a compiler infrastructure available from University of Illinois at Urbana-Champaign
LOC	Lines of code.
Object Space	A library providing objects and operations between them, available to the bytecode interpreter via a well-defined API.
Pygame	A Python extension library that wraps the Simple DirectMedia Layer - a cross-platform multimedia library designed to provide fast access to the graphics framebuffer and audio device.
pypy-c	The PyPy Standard Interpreter, translated to C and then compiled to a binary executable program
ReST	reStructuredText, the plaintext markup system used by docutils.
RPython	Restricted Python; a less dynamic subset of Python in which PyPy is written.
Standard Interpreter	The subsystem of PyPy which implements the Python language. It is divided in two components: the bytecode interpreter, and the standard object space.
Standard Object Space	An object space which implements creation, access and modification of regular Python application level objects.
VM	Virtual Machine.

3.2 Partner Acronyms:

DFKI	Deutsches Forschungszentrum für künstliche Intelligenz
HHU	Heinrich Heine Universität Düsseldorf
Strakt	AB Strakt
Logilab	Logilab
CM	Change Maker
mer	merlinux GmbH
tis	Tismerysoft GmbH
Impara	Impara GmbH

References

[D04.2] *Complete Python Implementation*, PyPy EU-Report, 2005

[D14.1] *Report_About_Milestone_Phase_1*, PyPy EU-report, 2005

[D14.3] *Report_About_Milestone_Phase_2*, PyPy EU-report, 2006

[D14.4] *Report_About_Milestone_Phase_3*, PyPy EU-report, 2007

[ZOPE] *miniGuide Zope Sprints*, http://www.zopemag.com/Guides/miniGuide_ZopeSprinting.html

[TROUBLE] *Trouble in Paradise: the Open Source project PyPy, EU-funding and Agile practices*, Agile 2006, Minneapolis

[SPRINTS] *Running a sprint*,Steve Holden http://www.onlamp.com/pub/a/python/2006/10/19/running-a-sprint.html

[SPRINTANN] *PyPy: example of a sprint announcement*, http://codespeak.net/pypy/extradoc/sprintinfo/trillke-2007/announcement.html

[PLAN] *PyPy: example of a planning file*, http://codespeak.net/svn/pypy/extradoc/sprintinfo/trillke-2007/planning.txt

[SPRINTREP] *PyPy: example of a sprint report*, http://codespeak.net/pipermail/pypy-dev/2007q1/003578.html

[D14.2] *Tutorial and guide through the source code*, PyPy EU report,2007

[D02.3] *Testing framework*, PyPy EU-report,2007

[D02.1] *Development Tools and Website*, PyPy EU-report, 2007

[SYNC] *PyPy: example of a sync-meeting protocol*, http://codespeak.net/svn/pypy/extradoc/minute/pypy-sync-2007-02-02.txt

[XP] *Extreme programming explained: Embrace change*, Kent Beck, 1999

[BRIDG] *Bridging the distance*, Scott Ambler,http://www.agilealliance.org/library, 2002

[IEEE] *Sprinting towards Open Source Development*, Greg Goth, IEEE Software, Jan/Febr 2007

[AGILE] *Agile Manifesto*, http://agilemanifesto.org/, 2001

[VTT] *Agile software development methods - review and analysis*, Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta VTT Publication 478, 2002

[SCRUM] *Agile Software Development with Scrum*, Ken Schwaber, 2002

[PEOPLE] *Characterizing People as Non-Linear, First-Order Components in Software Development*, Alistair Cockburn, 4th International Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, 2000

[OODA] *What Lessons Can the Agile Community Learn from A Maverick Fighting Pilot?* Steve Adolphe, Agile 2006, Minnepolis

[SUN] *Follow the Sun: Distributed Extreme Programming Development*, Monica Yap, Agile 2005, Denver, 2005

[SHORE] Phone interview between James Shore and Beatrice Düring, 19th of March 2007

[MYTH] *The mythical man-month, anniversary edition*, Fred Brooks, 1995

[D01.2] *Project Organization*, PyPy EU-report, 2007

[CODE] *Summer of Code*, http://code.google.com/soc/

[PYPY] *Summer of PyPy*, http://codespeak.net/pypy/dist/pypy/doc/summer-of-pypy.html

[LEARN] *Sprint-Driven Development: working, learning and the process of enculturation in the PyPy community*, Avram, Sigfridsson, Sheehan, Sullivan, The Third International Conference on Open Source Systems, Limerick, Ireland, 2007