Trouble in Paradise: the Open Source project PyPy, EU-funding and Agile practices

Beatrice Düring
ChangeMaker
bea@changemaker.nu

Abstract

PyPy is an Open Source project, partly funded by the European Union, employing agile techniques evolved within the Python Community such as "sprint-driven development".The project started as a grass-root F/OSS effort in late 2002 and received EU-funding from December 2004 until November 2006. In this paper we will present the various influencing factors that creates the hybrid project process that is PyPy. These influencing factors are the F/OSS Python Community (climate of the community from which PyPy grew from), agile practices (such as the Python community evolved technique of "sprinting") and the EU funding practices (resource tracking and reporting) impacting the project. These influencing factors laid the foundation for the custom-made project process that makes this unique hybrid project work. The main factor for driving this process is the skills of the team of core developers instigating the project. PyPy, with its open and transparent communication and collaborative work style, is again a proof that the best agile practice is the people factor.

1 Introduction

This paper presents the story and the experiences gathered so far in the PyPy project. The challenge has been to integrate such diverse perspectives as agile practices being used in a distributed development style in an Open Source project that is partly funded by the European Union.

The PyPy project started as a grass-root effort among core developers in the Python language community, aiming at building a Python implementation purely built in Python. It was a language implementation project that was purely driven from a non-profit perspective which rapidly increased in complexity when the Open Source project applied for EU-funding.

Today the project has over 4 years of intense activity as an Open Source community (F/OSS, Free Open Source Software) effort and have completed a successful first year of the EU-funded effort. In order to understand how the PyPy project manages to strike a balance between being an F/OSS community, while also having the core work of the project being funded by the European Union (with all the structure and process requirements that entails) we need to look at the influencing factors of the project process.

The influencing factors of the PyPy project process are:

- The F/OSS/Python community factor. Different Open Source communities have different "flavors" regarding culture, communication climate, structure, decision-process etc. PyPy was born out of the Python Community and shares some of its characteristics which have had a huge influence on the project process so far. Aspects such as distributed/dispersed development and the supporting practices and infrastructure in place to quality assure software development in a collaborated manner are part of this important influencing factor.
- The Agile factor. Almost a subset of the F/OSS/Python community factor - agile practices that have been in use since the inception of the project, that in fact have been the main drivers of the project are very much

inspired by other projects in the Python community. A key aspect of this influencing factor is the Python version of "sprinting". We will present this "agile" practice, explain and trace its origins within the Python community and identify which of the main agile practices inspired the Python world to go "sprinting" - a key technique today not only for the PyPy project but also for all main Python projects. We will also touch on aspects such as Test Driven Development and explain how these are implemented in PyPy.

- The EU factor. PyPy is the first F/OSS community that organized themselves into an EU-project, receiving EU-funding. It has had a huge impact on the project, and it is important to identify how EU project and process requirements can be "weaved" into an Agile F/OSS project - a challenge indeed. The EU-funding has been and still is an enormous opportunity; the project would not have had the rapid progress and impact it is now having without the funding. It is also very important to note that without striking the right balance between the three main influencing factors (F/OSS, agile and EU-funding) this factor might also prove to be a "show stopper", something that might influence the PyPy community negatively. This is a constant challenge - we explain here how we created our process and added a "consortium-level" structure to the F/OSS community and the steps we took in order to mitigate this risk.

Our conclusion so far in the project is that we believe that the practice of sprint-driven development, as being used in the PyPy project, makes Agile and Distributed/dispersed work styles more possible to combine. We also believe it is a very useful practice for creating value and ensuring quality in projects with hybrid cultures and methodologies. It is our aim to start to showcase this with experiences from the first funded year of PyPy. We also feel that our experiences as a unique hybrid project would be of interest to other communities such as the agile community and we hope that this paper will encourage cross-community interaction and communication.

1.1 Project context

1.1.1 Objectives

Due to the dual nature of the project even such a thing as objectives can get complicated. If you, as a potential PyPy user, would visit out main developer server, Codespeak, you would find the following statement:

"The PyPy project aims at producing a flexible and fast Python implementation. The guiding idea is to translate a Python-level description of the Python language itself to lower level languages. Rumors have it that the secret goal is being faster-than-C which is nonsense, isn't it? more..."

Codespeak housed the F/OSS project since its inception and it has been a driving strategy to continue a look-and-feel of a purely F/OSS project in order to not confuse developers interested in just that - PyPy as a Python implementation.

Digging a bit deeper you can find documentation which states the objectives of the EU-funded part of the project. The EU project´s title is "PyPy" which is an "acronym" for "Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community". The EU-project has three objectives: one technical, one research-oriented and one methodology-oriented objective. In short PyPy is about building an optimized compiler and interpreter purely in Python, applying Aspect Oriented Programming and abstract interpretation techniques. The methodology objective aims at showcasing the "sprint-driven development method" being used by PyPy and you can view this paper as part of fulfilling the methodology objective. For more references on sprint-driven development, please see our paper for the XP 2006 conference [1].

1.1.2 Strategy

The strategy of the project is to leverage the community of PyPy and Python through an open and transparent communication 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 F/OSS part of PyPy has no budget tied to it and no formal organizational structure. There

are no resources tied to it other than the volunteers participating in the development process. These are the core developers and architects behind PyPy but also people from all over the world, driven by a combination of professional need and personal interest in Python language implementations such as PyPy and choosing to contribute to PyPy. There is also a large group of students on PHD levels participating and contributing, using their areas of interest in PyPy as thesis materials.

It is difficult to estimate the amount of people involved in PyPy but we estimate that 300-500 people actively follow the progress of the project - this might mean reading emails, IRC logs and documentation in some cases, asking questions and sending bug reports in others. There are around 50 people who have commit-rights to the source code. The core group of developers consists of around 10 people.

The EU part of the project is organized through a consortium which consists of 8 partners: DFKI (Germany), Ab Strakt (Sweden), Logilab (France), merlinux GmbH (Germany), tismerysoft GmbH (Germany), Change Maker (Sweden) , Impara GmbH (Germany) and Heinrich Heine Universität Düsseldorf (Germany) and 4 physical person partners: Laura Creighton (Sweden), Richard Emslie (UK), Eric Van Riet Paap (Netherlands) , Niklaus Haldiman (Switzerland).

The project effort of work for the 2 years of funding consists of 14 work-packages and in total 58 deliverables which are high-level functional and non-functional requirements that were formulated in a proposal and form the "Description of Work" in the contract with the European Commission. The funding received for this effort is 1.3 million euro. Of the core group of developers mentioned above almost all of them (10 people) are involved in some sense in the partner companies and organizations of the PyPy consortium.

1.1.3 History

The ideas behind PyPy started via discussions on European mailing-lists in the Python community late 2002 by people who had been active in the Python community for some time; core developers interested in language implementation. They based the discussions and ideas partly on their experiences with developing and driving some well-known projects in the Python community. They met up in February 2003 at a "sprint", a one week working meeting,to draft their ideas and to start PyPy as an Open Source project. The sprint facilities as well as travel, accommodation and time was financed by all participants privately.

Mid 2003 the idea of trying to get EU-funding for the project was born. It became clear that the project had an arbitrarily large scale and that receiving some funding would dramatically increase the pace and seriousness of the project - because funded developers can dedicate more of their time to the project. The involved developers and people stretched outside of the Open Source ecologies to try to gather as much information and contacts as possible in order to answer the question: "Should we go for it?" to which the answer quickly became "Let's see how far we get!".

Acting on this strategy proved to be a more difficult task. The entire proposal and negotiation process took over a year (Autumn 2003 until November 2004).A proper description of planned work, necessary to satisfy formal requirements, had not previously been part of the development focus and both the EU and the parties involved had to adapt to the situation.

The contract with the EU was signed and the funded part of the project, with its consortium structure, started 1 of December 2004. The funded part of the project ends in November 2006.

The first funded year of the project ended with a review in Brussels in January 2006, hosted by the Commission, reviewing the 10 deliverables comprising the work of the first year. All deliverables were accepted; based on recommendations by the external reviewers. The consortium and the Commission is now restructuring the volume of the remaining the deliverables (although not changing the scope of the project).

2 Influencing factors: the F/OSS Python community culture

2.1 The Python community

Python is an Open Source language, published under an OSI approved open source license. It was created by Guido van Rossum and is now one of the five most used languages in the world. Due to the nature of the language there is a strong focus on glue and integration with other languages such as C and Java. Typical aspects of

the Python community is that it houses four large language implementations as separate projects which communicate and discuss their experiences and approaches with each other. This intra-extra community focus and interest has created a collaborative atmosphere with an open and transparent communication climate.

2.2 The PyPy community

The PyPy project is one of the four large language implementation projects in the Python community. PyPy grew from the architectural experiences from previous successful Python projects and this prior community experience was vital for the evolving of the PyPy community due to the established trust in expert skills of the core developers starting PyPy - again the people factor. Thus making it easier recruit people into the PyPy community.

The PyPy project inherited the focus on collaborative approaches and open communication climate. The strategy of using sprints, as a core technique, to kick-start the project as well as moving the sprints to different locations had clear effects. It encouraged participation by meeting people locally. During the period of 2003-2004 6 sprints were arranged in various European cities. These sprints were hosted by universities and private individuals, participation was funded privately. The effect on the evolving community was a stable subscriber participation on the development list of between 140-150 people. After the EU-funding and the more systematic structure of sprinting every 6th week the amount of subscribers went from around 150 people to over 250 people. Thus turning the prior experience of separate sprints into a more systematic approach, sprint-driven development.

2.3 Supporting infrastructure

The amount of collaborative focus and open and transparent communication in an open source community will manifest itself in the supporting infrastructure. In PyPy version control is the primary means of providing a secure platform for incremental development. Subversion is used, covering both program files and documentation. Several automated mailing lists make sure that every person involved receives instant notification on changes - supporting peer review on all levels.

PyPy is also in some sense very much test-driven. PyPy is one of the primary users of a separate F/OSS project called the py.test, tools for automated testing with PyPy developers contributing improvements to py.test. Py.test contains a sophisticated system for running tests against the Python Standard Library. It provides automatic selection of overriding tests in case there is internally developed code that overrides the Python Standard Library. These are important compliance features when the project is about language implementation. There are also powerful mechanisms for disabling specific tests and for producing useful traceback output. There are also automatically generated web pages that show the status of all tests as per the latest checked in test results. This together with a public issue-tracker covering bugs keeps the development group focused on the quality of the code when doing continuous integration. It greatly reduces the need to coordinate and delegate refactoring work - this is handled in a self-organized way.

The main communication channels between developers involved in PyPy is to discuss over IRC (Internet-Relay-Chat) - open for all who are interested. Several mailing lists for discussions and information are also used. Web pages, documentation, tutorials, talks and papers etc are all available on the central developer server for everyone to access. A very useful feature that really supports information and communication are the public email archives covering the key mailing lists - going back to the start of 2003. As a newcomer to the project in the fall of 2003 these public and easily accessible mailing list archives was the primary means for me personally to get into the project. It provided answers to who had been key people in the process, what had been crucial topics, how had discussions and decisions been handled. It is also regularly being used with newcomers in the development team, providing fast answers to questions and also providing context to technical decisions being made in the past. A useful just-in-time documentation and a unorthodox but very efficient way of disseminating information and knowledge.

Now - all this infrastructure is being used in almost all larger open source projects and quite a few of them are much more hierarchical and non-transparent in their communication. In PyPy there is no hierarchy for receiving commit-rights. A newcomer to PyPy can instantly receive an ac-

count on the development server, with full commit rights to the version control system. The reason why there are no constraints is that the process is very much self-organized and managed by the developers from a social level (peer review, code review, supporting communication channels, coaching, mentoring) and with strong automated tools covering tests, versions and back ups covering the technical level.

An extensive coding guide, published on codespeak.net (development website) serves as an introduction to the set of specifications being used while coding PyPy. Yet another piece of the puzzle, the coding guide together with commit-access to both source code and tests supports cohesion in the development process. This increases readability and uniformity so that automated testing tools can be more efficiently used. Again - simple and useful support that is much needed when working dispersed in a larger team of developers that might never meet face to face.

In a distributed and dispersed work style these two (social and technical) levels needs to be consistent and support each other. Discrepancies would be immediately noticed. As stated before - the people factor is again evident. If you wish to encourage participation and contribution there has to be trust as well as a supportive environment in order to manage a virtual collaborative workspace. The main feature of such a supportive infrastructure is to reduce the cost of information since coordination and communication is even more difficult in a dispersed environment. If this easily accessed information of the status of the software is combined with trust (that is commit-rights) you have the basis for a proactive, self-organized culture.

2.4 Supporting practices

Another good example of the collaborative nature of the Python community is the way in which the various projects share best practices. Core developers in the PyPy project picked up the practice of synchronization meetings as a powerful way of supporting distributed and dispersed development. This practice was inspired by the experiences of development processes at Canonical.

Sync-meetings are weekly short coordination meetings between developers. These are open to all developers active in the PyPy community, usually but not necessarily involving aspects of EU-funded work on deliverables. These 30 minute IRC-meetings serve a weekly synchronization for regular discussions and integration of ongoing work. Meetings are prepared with an agenda sent out to pypy-dev and minutes are distributed on pypy-dev and archived in the repository with publicly accessible links. The work of preparing, managing and documenting the meetings is rotated between active developers and is self-organized as well.

Sync-meetings have proved to be a very good complement to a process in which the project sprints every 6th week. Sync-meetings keep cohesion as well as a team focus-and-feel for the dispersed work style between sprints. There are three crucial aspects of this practice. The first one is that our experience as well as that of Canonical points to keeping the time for the meeting brief. The reason for 30 minutes limit is that it gives priority on what topics to choose. However complex your development situation is you should choose not more than 3 topics, topics that could be discussed and decided upon during these 30 minutes. Not having this time limit would create long and tiresome IRC meetings which would affect motivation people and also create more confusion than results.

The second aspect is that it has a fixed format in which the meeting starts with all developers participating in the meeting presents a very short status - answering the questions LAST (last week), NEXT (their focus for the upcoming week) and BLOCKERS (if they are stuck and need help). This means that the group get overview and can track progress, albeit loosely. All individuals get an opportunity to voice their personal situation and the group jointly discuss potential blockers and how to solve them. The fixed format help to keep focus and creates a rhythm that makes it easier for newcomers to get into the flow of the meeting.

The third aspect is also the main challenge with the practice. In PyPy we rotate the organization of the meetings between the developers. But because meeting moderation has to do with experience, people skills as well as a certain overview of the project not everyone can actually do this task. Our experience shows that it is also very important to have core people involved in identifying topics needing to be discussed - having the progress of the project in mind. It is very easy to choose topics that do not need the group mind to discuss, that are too detailed, that has

too much focus on information distribution rather than discussions.

Here is what one of the new core developers of PyPy says about sync-meetings:

> *"Sync-meetings are useful because they enable developers to discuss and clarify issues among themselves and to provide a common focus when working distributedly. They are also a lightweight way to synchronize activities, resolve blockers and to get an overview about who is currently doing what work."* – Carl Friedrich Bolz

3 Influencing factors: agile practices in PyPy

3.1 Sprints

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".

Tres Seaver of the Zope Community, one of the instigators of the sprint method as it is used in the Python community says the following about how they evolved the method during 2001 and 2002:

> *"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."*

The Zope community as well as other Python projects such as PyPy have seen that sprints generate results beyond the creation of software:

- It helped to evolve the community through participation and hands-on contact with the core developers.
- It is a live training session not only in producing code but also in the development methods being used (the sprint method in itself, pair programming and TDD).
- It supports activities such as design decisions and high-level requirements discussions, creating cohesion and understanding as well as minimizing risks with dispersed/distributed work where these kind of activities can be really difficult to manage.

In order to provide examples - in PyPy these discussion sessions have been done in various different ways during sprints. The proposal for the EU and the work package descriptions were drafted through group discussions during sprints. Later sprints in which the process needed design decisions, developers paired up in several groups of 2-4 people per group for daily discussion sessions. These sessions ended with brief documentation and presentations for the other groups where decisions were made. Usually what happens is that the daily sprint planning results in a pair of core developers choosing a topic to discuss. These discussions have even acted as tutorials for other developers sitting in and following the discussions. Some of these discussions have also been filmed for dissemination purposes.

The reason for naming the method sprinting, is not at all connected to Scrum and the usage of sprints in that context, according to Tres Seaver is:

> *"... 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."*

To summarize the results of this first try out of the methodology shows that sprinting was the driver for making Zope3 a community-driven project: Zope Corporation in fact ceded control over the development process to the new "Zope3

core" group created via sprinting. Less than 2 years later the same effect was to be found in the F/OSS project PyPy - the community evolved through sprint driven development.

The sprint method, as well as key aspects of F/OSS supportive infrastructure and practices was established within the project before PyPy received its EU-funding. Thus, in the proposal,we put much emphasis on the methodology in itself when designing the consortium level process. Examples of how this was done was that we created a methodology objective, specific deliverables regarding documentation of the sprint-driven methodology as well as designing a process during the 2 years of funding in which the project and the developers sprints every 6th week. This was taken into account when estimating the budget, adding costs for travel and accommodation for all partner organizations - covering 14 sprints during the 2 years. So far during the funded part of the project we have organized 10 sprints - successful sprints from both a development, recruiting, dissemination and networking aspects. Another important fact is that the sprints are organized by the developers and as such is another good example of the collaborative manner of the culture of the project and the community.

Why did PyPy choose sprinting as a key technique? 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, continuous integration). 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.

Another insight, worthwhile for other more discipline based projects to ponder, is how an agile process like sprinting is much more suited for creative work between groups of distributed people. Traditional software development, as well as traditional project management techniques have a tendency to hinder creativity due to the inbuilt over-structured, segmented and control-oriented approach which in most cases ends in less quality when results are being measured.

4 EU-project practices

4.1 Consortium structure

Key requirements regarding organizational structures of a Framework Programme 6 EU-project are:

- a consortium of partners performing the work described in the contract
- a project co-coordinator managing contract administration and communication between consortium and the Commission
- a project manager, responsible for the project reaching its goals within the time frame and budget

The challenge was to design a project process that created a minimal amount of changes to the structure being in use in the F/OSS project. It was especially important to maintain an open and transparent communication, even for decision on consortium level and such. There was a fear of the EU-project adding "top-down" structures on the F/OSS work and on the core developers instigating the project.

We identified a minimalistic approach of management roles (project co-coordinator, project manager, assistant project manager) and created a management team to make sure that there was collaboration between these roles. Although the responsibility rested on the different management roles and the management team the strategy implemented was to delegate as much as possible of the responsibilities and decision-making to the core developers.

The strategy was to keep "conceptual integrity" [2] of the vision and the idea in the hands of the core developers. As Brooks stresses, a uniform system design is better than uncoordinated and independent design ideas. The core developers had a clear and agreed vision - but would they be allowed to implement it within a fixed contract work style, with new partners involved that had not been involved in the process from the start? The core developers were organized into a technical board, responsible for planning and coordinating the development work between the partners, with the mandate to make decisions. A somewhat negative result was the added workload and responsibility on developers regarding EU related work.

This structure was the most difficult to design and implement due to the very different nature of its purpose compared to the collaborative, self-organized nature of the F/OSS project - again it succeeded the way it has because of the core developers acting as an interface between the developer group and the consortium level work - again we see the trust aspect and the people factor in play.

Sprints were budgeted for and designed into the process, together with a time plan for all deliverables. The project was divided into three different phases, depending on the nature of the work flow. In that sense we could describe the EU-part of the project as a fixed-contract style of work, but with time plan, deliverables and work package descriptions on a high-level, not broken down into more granular tasks.

4.2 Communication and documentation

All documentation and supporting communication infrastructure is hosted on the same developer server as the source code, covered by the same version control system. Only a few repositories are not publically available (such as the resource tracking repositories) - but they are fully accessible for everyone employed by a partner organization.

The communication infrastructure being used on the consortium level of work mirrors that of the development work - having years of experience on the distributed work style. In the case of PyPY there are mailing lists as well as IRC channels for consortium level work. We even implemented a procedure in our internal consortium agreement to allow for virtual meetings for decisions in the consortium. So far this have been the primary meeting form and it has worked well as a channel for making decisions. IRC-logs and minutes support the procedure. In some cases decisions have also been made via email on the consortium mailing list.

Although not the primary focus it is also useful to have the regular sprints to coordinate specific issues between some partners or between all partners, have physical meetings and discussions between the management team and the technical board etc.

Our funding have also resulted in the possibility to have a more unorthodox approach to documentation - the project have experimented in filming sprints to show the development method as well as filming talks and discussion. Our film material is planned to be released before summer 2006.

5 Troubles in Paradise: striking a balance

5.1 Developer driven versus formal project structure

The fear of a top-down, hierarchical decision process of the consortium was a justified one. It is interesting for us to note that now, having the experience and more contractual overview to note that there is nothing in the requirements from the Commission that forces a project to a more traditional and strict project process. It is mostly a question of the experience and customs of the partner companies. In that sense there was much flexibility in designing a process that allowed for a developer driven process regarding not only software development but also sprint administration and planning as well as consortium work.

The majority of the partners with key roles in the consortium organization had been working together since before the funding. In that sense procedures and best practices had been tried out. The results from the first year showed that a minimalistic path could be identified and that the important work would be to review and adjust the process when it did not support the work any more, or new situations arose that we had not planned for (and that did happen!).

Already year two has another look-and-feel to it on a consortium level when it comes to how the management team and the technical board works, because of the different nature of work in year two.

5.2 Agile strategies versus formal EU-contractual requirements

Our main agile practice, sprint-driven development, was successfully integrated into the formal contractual requirements, allowing for the same free-floating agile process of self organizing and decision making as existed before the funding, but with a more systematic and documented style.

But here is also our biggest failure, the main "un-agile" aspect of the entire project having a large negative effect not only on the EU-part of

the project but also on the community. We had planned for a process that kept an "open-door" policy that allowed us to fund non-consortium persons from the community to attend sprints. The budget was allocated, the procedure within the sprint context of handling more newcomers were known to us - the main show stopper was that the PyPy sprint funding did not fit within the customs and practices of contracts for cost claims in the Commission.

Every time we want to encourage participation and fund non-consortium people to participate in sprints and contribute, the procedure now is that they have to join the consortium as a full partner. This creates the need for a contractual amendment with the Commission which adds administrative work to the project as well as for the persons in question. A too blunt instrument and today we still do not have a working solution to this problem. It is an unfortunate example on how the influencing factor of F/OSS, agile practices and EU-funding collides and creates negative impact on the project.

5.3 F/OSS community versus hierarchies for "conceptual integrity"

There are many examples of F/OSS projects and communities that have dried out because of the fear of the architects to allow the contributors to fully participate and also influence features and visions of the software. In PyPy this challenge comes twofold. Not only is there a challenge to let the core developers keep "conceptual integrity" while also allowing the community to influence the direction and features of the software. There is also the consortium level of work in which the core developers could fear to "lose" the mandate of driving the architectural and design work. And yet another layer of risk and complexity would be if the consortium level would "shut the door" on the community, enforcing a more closed development process.

As in many cases - being risk aware is the first step to mitigation. Because the project is so deeply rooted in the F/OSS Python community, with its specific culture and climate, and because so many people involved in the core work on both the development and consortium level also shared this background they took great pains to avoid situations like this to happen. In fact, what we have realized only some months ago is that there is another risk, already partly having effect on the community. Because of the funding the development process is progressing much more rapidly in PyPy than more normal F/OSS projects. The speed and the dramatically increasing learning curve could make members more passive because they do not have the time to follow full-time IRC discussions, postings on mailing lists and the actual source code and increasing test suites.

This is a challenge and it is the focus of the developer group to try to distill and package information in order to help people to better navigate the progress.

6 Conclusion

The important question is the following (and is also part of the methodological objective of the project): is there enough room to manage a project and Open Source community within the plan-driven inspired methods that are required in EU-funded projects, while still working agile and distributed?

We believe so. The one clear dominating factor to make all this succeed is, as always, the people factor, the CRACK performers as Boehm and Turner calls them ("Collaborative, Representative, Authorized, Committed, Knowledgeable") [3].

The core developers of the PyPy project had the right mix of various skills in order to succeed in setting up a hybrid environment - enabling them to work full time on a project they strongly believed in. The most crucial mix of skills for making this possible was/are:

- **Social:** The ability to communicate open and transparent, to mentor and tutor dispersed as well as reinventing different collaborative work styles of the sprint method, "manage" groups and community as well as consortium, and handle conflicts)
- **Leadership abilities:** The ability to step into formal leadership roles in technical board structures, manage sprints and sync-meetings as well as the more informal management of the community of developers. Managing the balance between encouraging participation but still holding true to their vision, their "conceptual integrity".
- **Ability to network:** To be open to other communities, inviting new partners in or-

der to create a working consortium structure in the EU-project, curious and collaborative towards other Python implementations and other languages and research approaches, sharing knowledge and experiences and seeking bets practices of other projects.

- **Entrepreneurs:** To risk the community through pursuing the idea of EU-funding in order to fulfill the ambitious vision, managing to not only create a consortium with innovative structures of cooperation but also to create new companies.
- **Technical skills:** Programming language and implementation aspects,frameworks, mathematics, computer science - core skills for the project. Also the ability to design, setup and effectively manage supporting infrastructure for the development process.
- Managing the balance between encouraging participation but still holding true to their vision, their "**conceptual integrity**" This while working agile with open and transparent communication through sprints, documentation, tutorials, mentoring, syncmeetings. Resulting in a lively and growing the F/OSS community around the project.

So, could it be said that for an agile software development process, especially one that is distributed and community oriented, within a framework of EU-funding, that it is heavily people dependent? Or to stress it even further, sprint-driven development as a methodology does not exist and function without an agile group of people, Crack performers. The people are the methodology in some sense and if you wish to draw upon the experience of the PyPy team you need to look at the supporting practices around the people in order to find what can be duplicated and tested in another project environment. This conclusion matches what Alistair Cockburn writes in his paper "Characterizing People as Non-Linear, First-Order Components in Software Development" [4]:

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

If we accept this conclusion then we can also, thanks to the people, start to get innovative regarding practices. Designing the project process based on the specific needs of the unique project environment you are facing. In the case of PyPy this means that we are exploring the methodology as we go along, adjusting and fine tuning the process as well as the software.

So, when drawing from these different skills within the community of developers, the people, in the PyPy project one possible conclusion would be that a truly agile approach dominating the work style of an Open Source project will increase the ability of the community to spread the strategy of agility to other domains.

By this we mean that what started as agile practices in the development process quickly became influencing factors when designing other project processes. Examples of this in PyPy is how the sprint-driven development acts as a focal point not just for the development work (colocated as well as dispersed) but also for the formal and informal management of the project. Sprints together with the CRACK performers was what made the community grow and evolve. It was the foundation for a hybrid project where agile practices and EU-funding can fit within a distributed Open Source context.

7 Acknowledgments

The author would like to thank the following people who have in various ways helped with the creation of this paper: Angela Martin, Emily Bache, Tres Seaver, Carl Friedrich Bolz.

I would like to dedicate this paper to my dear friend and mentor of the Open Source Python and PyPy community Holger Krekel.

References

[1] Beatrice Düring, "Sprint-Driven Development: Agile methodologies in a Distributed Open Source Project (PyPy)", XP 2006

[2] Fredrick P. Brooks, Jr, "The mythical man-month, anniversary edition", Addison-Wesley, 1995

[3] Barry Boehm,Richard Turner, "Observations on Balancing Discipline and Agility", (drawn from the book "Balancing Agility and Discipline: A Guide to the Perplexed", Addison Wesley, 2003)

[4] Alistair Cockburn, "Characterizing People as Non-Linear, First-Order Components in Software Development", Presented at the 4th International Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, June, 2000, http://alistair.cockburn.us/crystal/articles/cpanfocisd/characterizingpeopleasnonlinear.html