Sprint-driven development: working, learning and the process of enculturation in the PyPy community

Anders Sigfridsson, Gabriela Avram, Anne Sheehan and Daniel K. Sullivan
Interaction Design Centre
Department of Computer Science & Information Systems
Engineering Research Building, ER1002
University of Limerick, Ireland.
{anders.sigfridsson, gabriela.avram, anne.sheehan, daniel.sullivan} @ul.ie
WWW home page: http://www.idc.ul.ie

Abstract. In this paper we examine sprint-driven software development as it occurs in a specific Open Source community, PyPy. Applying a situated learning perspective, we report the findings from a study focused on the activities leading up to, taking place during, and following after sprints. The study included analyses of sprint reports, email archives and other documents available on the community website, as well as a one-week period of direct observation of a specific sprint. The objective of the study was to elaborate on how the practices of sprint-driven development in the PyPy community facilitate learning, the dissemination of knowledge among its members and the expansion of the Open Source community. This paper aims to assess how sprint-driven development can facilitate situated learning in distributed software development by describing the practices applied in PyPy.

Keywords: Distributed software development, Open Source communities, sprints, situated learning.

1 Introduction

Software development is a complex task. It is an activity which not only requires people with highly specialized technical skills, who are capable of working with highly abstract constructs and keeping up to date in an uncertain and rapidly developing area, but it also requires a high degree of collaboration. A software development project is often characterised by large scale, uncertainties, and complex interdependencies [14]. Further adding to these difficulties is the fact that software development is increasingly carried out in a distributed manner, fuelled by the complexity and large scale of modern software systems, by the trend toward globalization and the search for an educated yet inexpensive work force.

While the challenges facing globally distributed software development are not unique, certain difficulties – technical and managerial, as well as social and cultural - are further exacerbated by geographical and temporal distance. Three often mentioned issues are cultural differences, trust, and communication [e.g. 10, 19, 22]. But these are by no means the only difficulties, as more traditional concerns such as coordination, control and software processes are also affected by the distribution [e.g. 4, 11]. Currently there is significant interest in both academia and industry to gain a better understanding of these key issues and, above all, to discover ways of addressing the difficulties and thus improving practice [9].

A number of the most complex and successful software products nowadays – Linux, Apache, Firefox, OpenOffice and Eclipse, to mention but a few – have been developed or enhanced by Open Source Communities. The growing success of Open Source Software has resulted in it becoming a focus for research into issues relating to distributed software development. What is interesting about this phenomenon is how these loosely organized and often ad-hoc communities, using mostly simple communication and development tools such as email lists, version control systems and simple text editors [e.g. 8], can manage to develop high quality software [e.g. 17, 13].

Whilst we must recognize that the practices of Open Source communities are by no means the "silver bullet" for developing software and that many of them may not be adaptable to the more rigid requirements of the corporate world, they still provide a valuable resource in terms of understanding the key issues relating to distributed software development thus potentially providing guidance in the improvement of practice. Within the Open Source arena it is quite common that novel development methods and ways of working in cooperative projects are tried out. Some projects not only aim at producing an operational end product, but also actively investigate and improve software development techniques and attempt to find improved ways of running software development projects. One such Open Source project is PyPy.

The PyPy project[1] evolved from within the Python Open Source community and is focused on re-implementing the Python programming language using Python itself. The end-product will be an open run-time environment for the Python language, but this is not the only goal. It also focuses on investigating novel techniques for implementing practical dynamic programming languages and aims to showcase a software development method called "sprint-driven development". In this paper, we are focusing on this latter aspect.

We have conducted a study of the activities in PyPy consisting of document analysis of mailing lists, archives, sprint reports and other documents available on the community website[2] as well as a direct observation of one PyPy sprint, which took place in August 2006 in Limerick, Ireland. The objective of the study was to examine the actual activities leading up to, taking place during and following after sprints and to elaborate on how sprint-driven development facilitates learning, the dissemination of knowledge among its members and the expansion of the Open Source community. This paper will present a brief introduction to the PyPy project and the principles of sprint-driven development, and will then provide some specific accounts of the collaborative practices that occur in this community. We will apply a situated learning perspective to

[1] http://pypy.org/
[2] http://codespeak.net/

explain what we have observed and will draw conclusions about what lessons can be learned from PyPy regarding how sprint-driven development facilitates situated learning in distributed software development.

1.1 The socGSD project

At the University of Limerick, Ireland, a group at the Interaction Design Centre has received national funding as part of a software engineering research consortium to study the social, organisational, and cultural aspects of global software development (socGSD). The socGSD project aims to explore through case studies, how organizations attempt to manage the coordination of engineering work via a variety of mechanisms, from the formation of closely-knit, though distributed, teams in multinational companies through to Open Source communities, who act as self-organising bodies and manage to produce notable results without having formal management structures and too many well-defined rules. Our research is based on the findings of earlier studies on articulation and coordination work, information sharing, knowledge management and informal learning practices in distributed work. Our work is exploring the diversity of ways in which distributed teams shape their work practices and come to a joint understanding of their objectives. Our research also considers the various ways in which developers acquire new skills through their day-to-day practice and continuously improve their practice through learning and innovation.

Our research methods mainly rely on an interpretive, naturalistic approach to data collection and analysis. This means that we study the phenomenon in the actual settings where the work activity takes place, attempting to make sense of the work through the eyes of those actually doing it.

The study of an Open Source community for the duration of a sprint provided us with an excellent opportunity to observe the actual work practices of a team of developers who were collocated for one week, but also to consider these practices from the perspective of the context offered by the community's web presence and accounts of similar events.

1.2 Situated learning

Various theories of learning exist, each emphasizing different aspects of learning and embracing different fundamental assumptions regarding the nature of knowledge, learning and the role of the individual learner [15, 3, 23]. According to Lave&Wenger [15], situated learning can be considered as a bridge between a view according to which cognitive processes are primary and a view according to which social practice is the primary, generative phenomenon and learning is one of its characteristics. From this latter perspective, learning is viewed as an integral and inseparable aspect of social practice. Our study adopts a social practice theory of learning. In particular we are influenced by the concepts of situated learning, specifically legitimate peripheral participation, an analytical perspective introduced by Lave & Wenger [15] as a way of understanding learning.

According to Lave & Wenger [15] knowledge is learned by becoming a legitimate peripheral participant in a community of practice and by gradually

acquiring “mastery”, or knowledge and reputation, through a process of social interaction. Learning is thus not the result of direct and intentional teaching; rather it is enabled by participation in practice and access to the learning resources available in the community [24]. Active participation of newcomers allows them to interact with more knowleageable peers and provides access to the expertise available within the community. Learners acquire not just formal knowledge and skill, but also the ability to behave as members of a particular community of practice. In the words of Brown & Duguid [3] it involves becoming an “insider” or “*becoming* a practitioner not learning *about* practice.” This situates learning squarely in the practices and communities in which the knowledge takes on meaning and significance.

Both Orr’s [18] and Lave & Wenger’s [15] research emphasizes that to understand working and learning, it is necessary to focus on the formation and change of the communities in which work takes place. Based on his ethnographic research on photocopier repair technicians, Orr posits that “not only is learning in this case inseparable from working, but also individual learning is inseparable from collective learning.” The implication is that knowledge and learning are not simply the property of the individual, but are socially constructed and distributed. Hence what is learned is connected to the context in which it is learned and so learning can be fostered by fostering access to and membership of a particular community of practice.

The application of a situated learning perspective to distributed teams and Open Source Communities is not new [e.g. 24, 8, 21]. According to Ye & Kishida [24], an Open Source community requires a high degree of openness in terms of both process and product, as it offers more learning resources to encourage participation. In addition, the manner in which a software system is partitioned also has an impact on knowledge acquisition. By allowing newcomers to work on relatively independent tasks, each with progressive difficulty, it fosters the possibility of legitimate peripheral participation. In other words, it allows newcomers to participate peripherally by contributing to tasks at their current skill level and to gradually move on to take charge of more difficult tasks as mastery evolves. Furthermore, research by Gutwin et al. [8] on awareness in distributed software development highlights the importance of facilitating peripheral participation through email and chat. The mechanism of “overhearing” inherent in these text-based communication tools allows developers to become peripheral participants in each others conversations, thus providing valuable awareness and enabling “expertise” to gradually become visible.

2 The PyPy project and sprint-driven development

2.1 The PyPy project

PyPy is part of the large Open Source community behind Python. Python is a programming language, published under an OSI approved Open Source License. The Python language was originally developed in 1990 by Guido van Rossum. Today, the *de facto* standard implementation of the language is the CPython implementation, which is also being developed as an Open Source project

The PyPy project also aims at producing an implementation of Python. But unlike CPython, which is developed and written in C, the PyPy project is developing an interpreter for the Python language in Python itself (hence the project name).[3]

However, creating a run-time environment for Python is not the only purpose of this project. The PyPy project came into being as an Open Source project in 2003 and in December 2004 the project received partial funding from the European Union (EU). As a result, the project objectives expanded to include a methodological goal, namely to demonstrate that the Open Source way of working in general, and the development methodology of choice in particular, are successful ways of undertaking distributed, collaborative work and hence can be of use in future EU projects as well as in large-scale development projects in general. The methodology adopted by the PyPy community is what has been called "sprint driven development".

2.2 Sprint-driven development

A "sprint" is a focused development session – developers gather in one place for a short period of time and work in pairs (or small groups) on specific parts of the software system. This type of event has become popular within some Open Source communities – for example, the OpenBSD and Linux communities - and has many names, such as "hackathon", "codefest", "sprintathon", "sprint", and so on. The primary purpose of these on-site meetings, which last from a few days up to one week, is to write and test code in a collaborative way. To facilitate access, these events are often collocated with conferences of relevance to the community's members, but they may also be hosted separately in various locations, usually organized by community members or hosted by sponsors.

The practice of using sprints for pivotal development was initiated by the Zope Corporation in the early days of the Zope 3 project[4]. In order to maintain focus, the traditional sprint is supposed to last for only three to four days and to involve no more than 10 people. A sprint generally incorporates aspects of eXtreme Programming such as pair-programming and test-driven development. In addition, it is usually led by a "coach", who sets the goals, organizes the event, coordinates the work, tracks the results and follows up.

The underlying concept is that a sprint is a good way to give a project "a boost by focusing the efforts of a group on specific development issues" [12]. Furthermore, sprints also offer valuable opportunities to maintain developer involvement, and to enable newcomers to get acquainted with the code base as well as the specifics of a project.

2.3 Sprint-driven development in practice in PyPy

The PyPy community describe themselves as a hybrid project, combining different aspects of Agile and Distributed Development within the context of an Open Source community [5]. In PyPy the developers are not just distributed but also dispersed, with no more than a few developers being located in the same place. The main strategy in PyPy to handle this challenge to the development

[3] For technical specifications, http://codespeak.net/pypy/dist/pypy/doc/architecture.html

[4] http://www.zopemag.com/Guides/miniGuide_ZopeSprinting.html

process is to "sprint" systematically, using sprints not only for software iteration purposes but also to provide an accelerated and collaborative physical practice that enables community building as well as the dissemination of knowledge and learning within the team. In fact the PyPy project itself originated from a one-week sprint held in February 2003.

The sprint methodology used with the PyPy community differs in a number of ways from the original Zope3 format described earlier. The focus of Zope3 sprints was to produce code and as such they tended to be rather closed events where only experienced Zope developers participated and they were usually arranged close to larger releases [5]. In addition, an appointed "coach" was used to coordinate the event and its outcome. However, within the PyPy community a sprint is an open event where newcomers are welcomed – indeed a sprint is seen as an opportunity to initiate newcomers into the project and clearly has a "tutorial" component. In addition, PyPy sprints are developer-driven and no formal role such as a "coach" exists. Instead, they have introduced a mechanism of initial and daily status meetings where the whole group makes decisions. A local contact will help to organize the logistics for the event based on the sprint location.

A PyPy sprint is usually 7 days long, with one free day in the middle normally dedicated to social events. The sprint is initiated with a start-up meeting. Tutorials will be arranged during the sprint if there are new participants present or if a new tool or feature has been implemented. For the remainder of the week, each day begins with a status meeting. During the status meetings, progress is discussed, tasks are drafted, the direction of the sprint is set or altered, and developers pair up according to needs, skills and wishes. During sprints pair-programming is used systematically – not only between core developers sharing an interest in a specific task but also for mentoring newcomers by pairing them with core developers [5]. The pairs may change each day, or may continue to work together for several days. Apart from the actual code, the outcome of a sprint is also a sprint report. The sprint report summarizes the activities and the initial goals and results. It also serves as an orientation for focusing the work of the community until the next sprint.

In PyPy, there is a rough plan detailing future sprints for the coming months, enough to maintain a general awareness of the dates and sites of upcoming sprints and allowing people to plan for attendance. About a month before a particular sprint, its content and goals are discussed on the mailing list (pypy-dev) and on the PyPy IRC channel, mostly by the core developers, although the discussions are transparent and anyone can, in principle, participate.[5] Information is also distributed on the general pypy-sprint mailing list and through the project webpage. As the sprint approaches, a more detailed sprint announcement is sent out. People can announce their intention to attend either by checking in the information in Subversion (the PyPy code repository and version control system), or by posting on the sprint mailing list. Lately, the PyPy project introduced "pypy-sync meetings" (on IRC) and this has also become a major forum for discussing the content and goals of upcoming sprints where any member can participate.

[5] http://codespeak.net/pypy/dist/pypy/doc/dev_method

3 Research Method

In our study of collaborative work practices, the preferred methods are inspired by ethnography. We try, whenever possible, to observe people in their normal work environment as they engage with their work practice. Furthermore, we interview the participants (individually or in groups) and bring into discussion events we observed, complementing what we saw with the addition of their perspectives. An ethnographic approach typically includes field work done in natural settings, the study of the larger picture to provide a more complete context of activity, an objective perspective with rich descriptions of people, environments and interactions, and an aim toward understanding activities from the informants' perspective [1]. More recent studies [16] claim that by narrowing the focus of field research before entering the field, using key informants and multiple interactive observation techniques and collaborative iterative data analysis methods, one can obtain reliable data in a shorter period of time than was traditionally considered.

The study we conducted was mainly centred on the sprint that took place in Limerick, Ireland, between the 21st and 28th of August, 2006. The sprint was hosted at the University of Limerick, with the assistance of local contacts. Three researchers where involved in this observational study, but none participated actively in the coding efforts. For the most part, there were 7 participants in this sprint, mostly core developers. A local developer joined the sprint for the last three days, and two newcomers also visited and attended a tutorial that was arranged for them. Since there were mostly core developers present, the sprint was considered an opportunity to work on some of the more crucial technical matters, e.g. the JIT module, core optimization and distributed testing.[6]

We studied this sprint mostly through direct observation, complemented by informal discussion and a dedicated Q&A session. We observed and recorded (video and audio) the start-up meeting and the daily status meetings, as well as observed some of the actual work sessions. Because of the interest expressed by two different groups of researchers at the University of Limerick in the PyPy way of working, the project manager organized a workshop on the first day of the sprint where the PyPy project and the sprint-driven methodology was presented. One of the most senior members of the project joined the last half of the workshop and there was a Q&A session.

Prior to the sprint, we reviewed a number of sources referring to Open Source communities in general and sprints, the Python language and the PyPy project in particular, including papers and talks, as well as mailing lists, web pages, bios, sprint reports, blog posts referring to PyPy, and so on. In order to get an insight into the activities of the project and the dynamics of the community, since it's inception, we studied the PyPy community's extensive online documentation (such as project descriptions and both sprint and EU reports), as well as mailing list archives and chat transcripts that are available on the website. After the sprint, we continued to observe the community for an

[6] http://codespeak.net/svn/pypy/extradoc/sprintinfo/ireland-2006/limerick_sprint-report.txt

additional four months, mostly through continued document analysis of email lists, sprint reports and other documents.

4 Sprints as a way of working, learning and innovating

Several authors speak about the various roles assumed by the members of Open Source communities [20, 24, 7]. The traditional evolution based on perceived levels of expertise, is from the periphery of the community to the centre: the majority of people start as users, get involved by discovering and later fixing some bugs, make occasional contributions to the source code, and only after gaining a reputation as an "expert" can they be accepted as core members of the community. The apprentice often has a long (and sometimes lonely) way to go before becoming actively involved in development. The PyPy community is, in this respect, quite different. There is no single leader or visionary – just a core group of passionate Python developers. Anyone who has the skills and motivation can rapidly become an active contributor, because within the PyPy community there is a welcoming attitude toward new participants which originates in the strong belief of the community members regarding the benefits of collaborative work. There is a strong culture of openness and transparency, or as described in [8] a culture of "keeping it public". Access to the PyPy online mailing lists and IRC is freely available. Core developers are accessible to answer questions or act as mentors both virtually via mailing lists and IRC and in person during sprints. The fact that the PyPy development process incorporates an automated framework for testing and version control allows for a more relaxed attitude regarding distribution of commit rights to new developers [5].

Several studies on Free and Open Source Software mention learning as one of the core motivations for participation [24, 7, 13], but in many cases, this simply means "lurking" and using the available code. While "lurking" - or in effect being a peripheral participant in the community - can provide valuable awareness information [8], in PyPy newcomers are encouraged to become directly involved in development from the very beginning. The PyPy community has developed a comprehensive and detailed repository of documentation, guides for beginners, talks, sprint reports, mail and chat archives in addition to its main code repository. While an important part of the PyPy community's body of knowledge is freely available on the web, becoming a member of the community is made quicker and easier by participation in collocated events such as sprints. Newcomers can make a decision about staying or leaving after being offered an immersion in the practices, social events and personal contacts that usually arise in a sprint.

Before deciding to join their first sprint, newcomers are encouraged to get accustomed to the work being done in PyPy. The architecture of the interpreter, the code itself, extensive coding guidelines, the available documentation, the tools used, configuration and various tutorials are all available on the PyPy website. Furthermore, newcomers are also encouraged to start socializing with the others by participating in email and IRC conversations and accessing the mail and chat archives. For example, the following excerpts from the PyPy mailing lists show how the community greets newcomers:

"Cool! Contributions are of course very welcome! I guess the most immediate step would be to read through the documentation and ask any question you might have (here – on the mailing list- or on the IRC channel). It certainly won't be a problem finding work for you :-)"

"In addition, note that this sprint is […] a coding sprint, and we specifically welcome newcomers.
If possible and interesting for you, feel invited :-) That's the best way to grasp the basics of PyPy and discuss. Also feel free to say hello in the #pypy IRC channel (irc.freenode.net) and discuss your interests."

Subsequently, during the actual sprint, newcomers are given tutorials and then "taken by the hand" usually by pairing up with an experienced developer, working together on a chosen topic and getting detailed feedback.

The participants in the Limerick sprint in August 2006 were in the majority part of the core PyPy group, with one exception: a young and enthusiastic developer who was funded through the "Summer of PyPy" initiative[7] (although it was not his first PyPy sprint). Pairs were formed and topics were chosen in an extremely flexible way. The start-up meeting highlighted the list of topics that needed attention resulting from previous sprints and discussions, the participants announced their intentions to the group, paired up according to these, and simply started working on them. Although the project manager (who has an administrative role and is not involved in coding) and one of the core developers chaired the meeting, their role was more one of facilitating the sprint, and not imposing anything on the group. At the end of the week, this role was taken over by another member of the core group. Every decision was taken collectively, and the initial program changed several times to accommodate people and events. Usually, there's a day dedicated to social activities in the middle of the sprint week, but this time the group decided to continue working through the dedicated break day because of a slow start on Monday morning, and to have a night out on Friday instead, when the local developers were planning to join.

This is one illustration of how flexible the working style of PyPy sprints are and it shows that agility and the ability to incorporate continuous change and adaptation are highly valued by the PyPy community. They innovate continuously, looking for both solutions to make their software more efficient, and for practices that would allow them to enhance or improve the form but keep the spirit of their activities.

The lack of formality and the relaxed atmosphere are probably the first striking aspect when observing a sprint. During the Limerick sprint, participants spoke to each other, moved around asking questions, joked and had fun. They were all located in the same room and maintained a certain awareness of what was happening in the other coding groups. They made the decision to take a break – or continue an hour more than planned – by consulting each other. Peers were invited to have a look when unexpected errors occurred or a new solution was tried out. Priorities were permanently shuffled, concepts re-invented, new routes adopted, tried out and sometimes abandoned.

In the Limerick sprint, different working styles could be observed in the pairs. In the first pair, one of the participants distributed his attention, switching between multiple windows, reading through his emails or keeping an eye on the

[7] http://codespeak.net/pypy/dist/pypy/doc/summer-of-pypy.html

chat channel while listening to a new solution proposed by his team-mate. His (more experienced) companion explained every step he was taking, made his reasoning transparent and asked a lot of provoking questions. A dialogue went on throughout the session: when the first developer had an idea he preferred to try it out instead of explaining it, while his colleague watched the screen, waiting to see the result. The second pair did not display as much interaction, perhaps because the tasks were divided more clearly between them. They seemed to work independently each on his own laptop, showing each other errors or successes and exchanging ideas only once in a while. The third pair was sharing a laptop. Most of the time, the laptop's owner was the one using it, but his actions seemed to result from their joint discussion. The conversation was vivid and emotional, accompanied by a lot of gestures.

The participants in a PyPy sprint benefit not only from their mutual knowledge sharing, but there's also a recognisable flow of enthusiasm. When speaking about the core group of developers during the methodology workshop, the project manager described them as "soulmates", who have much stronger bonds than the current EU project framework and want to continue working together after the end of this project. Sprints provide the opportunity for a process of learning and enculturation, where new participants get the chance to become directly involved not only in problem solving, innovation and planning, but also in the social life of the community.

5 Discussion

5.1 Learning facilitated by sprint-driven development

A major issue in distributed software development projects is how to facilitate learning about programming techniques, technology and project specific matters among project participants when direct interaction is limited due to geographical and temporal distance and, often, affected by national, social and organisational cultural differences. A sprint offers a good opportunity for the dissemination of knowledge, both among senior members of the community and to new members. However, being able to contribute to a software development project does not just require technical skills.

From a situated learning perspective, learning cannot be seen as an isolated activity, separated from the practice it is meant to enable [3]. Instead, learning involves becoming an "insider", not just absorbing a discreet body of individual knowledge, but learning to function within the community of practice. So learning the necessary skills needed to participate in a project like PyPy also involves learning about the dynamics of the community, what norms and interpretive schemes are dominating, and what range of behaviour is acceptable, as well as developing an identity in the community.

PyPy sprints are a perfect illustration of situated learning, as conceptualized by Lave & Wenger [15]: newcomers begin by reading the information online and joining mailing lists and IRC channels and then eventually join their first sprint and get more and more involved in the general development effort, learning happens in a community of practice, by participation (a peripheral one in the beginning), and by gradually acquiring knowledge and reputation through social interaction. Brown & Duguid write that the "central issue in learning is

becoming a practitioner not learning *about* practice" [3]. From what we have seen during this study, this is precisely what the PyPy people are supporting when welcoming new participants to sprints, arranging tutorials for them, and pairing them up with more experienced developers to do the work. This mechanism is further enhanced because the new participants are encouraged to participate in the mailing lists and IRC channels and to get acquainted with the system architecture and the code base prior to their first sprint, and thus have already started to form an identity within the community when arriving at the sprint.

Learning the concepts of the Python programming language does not mean one knows how to program in that language. Applying those concepts to a specific project and actually writing code is when learning happens. Sprints accelerate this process for distributed teams, recognising the important situated aspects of learning and supporting them.

5.2 Sprints as a way of sustaining and renewing the community

Previous research on Open Source software development has shown that learning is, in fact, a major motivational force for participants [24, 13]. It has also been argued that for Open Source projects to sustain themselves, the community must co-evolve with the system developed [24]. The community must be able to regenerate itself through both concrete contributions of code and the emergence of new members who can carry on the work. The sprints in PyPy, through conscious mentoring efforts, attract new members and enable them to both achieve the necessary technical skill and to create an identity within the community, thus ensuring the sustainability of the community.

However, regarding the formation of the community, there are also possible hazards with driving development through sprints. During the sprint, the centre-periphery relationship, usually based on experience and contributions resulting in a hierarchy in most Open Source communities, is altered: the collocated participants become the centre, while all the others move, in a way, to the periphery because they are missing from that specific location. The danger is that this leads to the formation of in-groups. The active PyPy coding effort is a subgroup within the wider Python community and those who participate in sprints are again a further subgroup (although temporary) of the overall coding effort. This situation can lend itself to the formation of in-groups and the exclusion of others and the eventually fragmentation of the group.

A previous experimental effort [2] to consider the in-group/out-group effect was concerned with the mixed media working environment whereby access to resources is not equally distributed. In part this is a consequence of having a substantial component of the development team collocated. The hypothesis which they examined was that individuals collocated together will interact more and form an in-group. We heard this concern voiced by the project manager herself. Since the progress is so rapid and so much happens during a sprint, they are aware that there is a risk that the non-participants can't keep up and can become passive. For example, this is acknowledged in one of the EU reports[8] where it is stated that, "due to the projects fast pace and its many developments,

[8] http://codespeak.net/pypy/extradoc/eu-report/D14.3_Report_about_Milestone_Phase_2-final-2006-08-03.pdf

it requires substantial effort for the average community member to contribute to the project." However, in the PyPy project, there is conscious effort to ensure the community doesn't fragment and so "the mentoring and supporting activities from the EU project members have increased accordingly."

The strategy has been to host sprints at different locations to encourage and facilitate participation from as wide a group as possible. During the period 2003-2004 6 sprints were arranged in various European cities (since then there has been a more systematic structuring of sprinting every 6th week) [6]. Sprints have also been organized on other continents whenever possible. For example, there was a post-PyCon PyPy sprint in February 2006 in Dallas, USA, and another one in Tokyo, Japan in April 2006. Also, during the recent Leysin sprint, in January 2007, a remote participant worked constantly with two others participating in the sprint to accomplish a specific task. Non-European developers whose participation in sprints is more difficult to organise have raised the possibility of doing a "virtual sprint" that would enable them to get involved as well.

6 Conclusions

Our study has focused on the actual activities leading up to, taking place during and following after sprints and the purpose has been to elaborate on how sprint driven development facilitates learning, the dissemination of knowledge among its members and the expansion of the Open Source community. The aim of this paper has been to illustrate how sprint-driven development can facilitate situated learning in distributed software development by describing the practices applied in PyPy.

The observations indicate that the sprint-driven development methodology as it occurs in PyPy is interesting because, while it is a way to accelerate the development in terms of written code, it also serves as a mechanism to expand the community and facilitate the enculturation of its members. In PyPy, we have seen how new participants are welcomed to sprints and how a real effort is made to include them in the community by encouraging participation in the online activities prior to their first sprint and arranging tutorials and pairing them up with experienced developers to work during the sprint. This attracts new members and enables them to both achieve the necessary technical skill and to create an identity within the community, thus enabling them to contribute. It also contributes to sustaining and renewing the PyPy community through the inclusion of new participants and the emergence of new core members and active developers.

7 Acknowledgements

The authors would like to extend thanks to the PyPy community and to the participants of the Limerick sprint for allowing us access and for being so accommodating. This work is part of the socGSD project at the University of Limerick. socGSD is one of the LERO (the Irish Software Engineering Research Institute) cluster projects funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI).

8 References

1. Blomberg J. et al. (1993) Ethnographic field methods and the relation to design. In D. Schuler and A. Namioka, (Eds.) *Participatory Design*, Lawrence Erlbaum, pp. 123-155.

2. Bos N., N. S. Shami, et al. (2004). In groups/Out Group Effect in Distributed Teams: An Experimental Simulation. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, Illinois, USA, pp. 429-436.

3. Brown J. S. & Duguid P. (1991) Organizational learning and communities-of-practice: towards a unified view of working, learning and innovation. *Organization Science*, vol. 2, no. 1, Special Issue: Organizational Learning: Papers in Honour of (and by) James G. March (1991), pp. 40-57.

4. Carmel E. & P. Tjia (2005) *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, Cambridge, MA.

5. Düring B. (2006A) Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project (PyPy). *The 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Oulu, Finland.

6. Düring, B. (2006B). Trouble in Paradise: the Open Source Project PyPy, EU-Funding and Agile Practices. *AGILE 2006* Minneapolis, Minnesota, USA IEEE Computer Society's Digital Library.

7. Ghosh, R. A. & R. Glott (2002). Free/Libre and Open Source Software: Survey and Study- summary report. *Workshop on Advancing the Research Agenda on Free/Open Source Software*. Maastricht, Int'l Institute of Infonomics, Univ. of Maastricht.

8. Gutwin C., Penner R. & Schneider K. (2004) Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pp. 72 – 81, 2004.

9. Hargreaves E., Damian D., Lanubile F. & Chisan J. (2004) Global Software Development: Building a Research Community. In *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, September 2004, pp. 1-5.

10. Herbsleb J. D. & Moitra D. (2001) Global Software Development. *IEEE SOFTWARE*, March/April 2001, pp. 16-20.

11. Herbsleb J. D. & Grinter R. E. (1999 A) Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, 16(5): 63-70

12. Holden H. (2006) *Running a Sprint*. ONLamp.com, Python Development Center. Available:

http://www.onlamp.com/pub/a/python/2006/10/19/running-a-sprint.html (15/01/07).

13. Kim E. E. (2003) *An Introduction to Open-Source communities*. Blue Oxen Associates. Available: www.blueoxen.com/research/00007/BOA-00007.pdf (19/01/07).

14. Kraut R. E. & Streeter L. A. (1995) Coordination in Software Development. In *Communications of the ACM*, vol. 38, no. 3, March 1995, pp. 69-81.

15. Lave J. & Wenger E. (1991) *Situated Learning: Legitimate Peripheral Participation*. New York: Cambridge University Press.

16. Millen D. R. (2000) Rapid ethnography: time deepening strategies for HCI field research. *Conference on Designing interactive systems: processes, practices, methods, and technique*, New York, ACM Press.

17. Mockus A., Fielding R. T. & Herbsleb J. D. (2002) Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 11, Issue 3 (July 2002) , pp. 309 – 346.

18. Orr J. (1996) *Talking about machines: An Ethnography of a Modern Job*, Ithaca, New York, IRL Press.

19. Prikladnicki R. et al. (2003) Global software development in practice: lessons learned. *Software process improvement and practice*, vol. 8, 267-281, 2003.

20. Rahtz, S. (2004) Building Open Source Communities, OSS Watch, University of Oxford, Available: http://www.oss-watch.ac.uk/talks/2004-11-19-bodington/ (20/12/06).

21. Robey D, Huoy Min Khoo & Powers, C. (2000) Situated Learning in Cross-functional Virtual Teams. *IEEE Transactions on Professional Communication*, Vol. 43, Issue 1, pp 51-66.

22. Sahay S., Nicholson B. & Krischna S. (2003) *Global IT Outsourcing: Software Development Across Borders*. Cambridge, UK: Cambridge University Press.

23. Wenger E. (1998) *Communitites of Practice: Learning, Meaning, and Identity*, Cambridge University Press.

24. Ye, Y. and K. Kishida (2003). Toward an Understanding of the Motivation of Open Source Software Developers. *International Conference on Software Engineering - ICSE2003*, Portland, OR.