Empir Software Eng (2011) 16:514–537
DOI 10.1007/s10664-011-9160-1

The search for a research method for studying OSS process innovation

Lutz Prechelt · Christopher Oezbek

Published online: 30 March 2011

Editors: Carolyn Seaman, Jonathan Sillito, Rafael Prikladnicki, Tore Dybå, and Kari Rönkkö

Abstract Medium-sized, open-participation Open Source Software (OSS) projects do not usually perform explicit software process improvement on any routine basis. It would be useful to understand how to get such a project to accept a process improvement proposal and hence to perform process innovation. We want to determine an effective and feasible qualitative research method for studying the above question. We present (narratively) a case study of how we worked towards and eventually found such a research method. The case involves four attempts at collecting suitable data about innovation episodes (direct participation (twice), polling developers for episodes, manually finding episodes in mailing list archives) and the adaptation of the Grounded Theory data analysis methodology. Direct participation allows gathering rather rich data, but does not allow for observing a sufficiently large number of innovation episodes. Polling developers for episodes did not prove to be useful. Using mailing list archives to find data to be analyzed is both feasible and effective. We also describe how the data thus found can be analyzed based on the Grounded Theory Method with suitable adjustments. By-and-large, our findings ought to apply to studying various phenomena in OSS development processes that are similarly heavyweight and infrequent. However, specific details may block this possibility and we cannot predict which details that might be. The amount of effort involved in direct participation approaches to qualitative research can easily be underestimated. Also, survey approaches are not well-suited for many process issues in OSS, because too few developers are sufficiently process-conscious. An approach based on passive observation is a viable alternative in the OSS context due to the availability of large amounts of fairly complete archival data.

Keywords Open Source · Methodology · Innovation introduction

L. Prechelt (✉) · C. Oezbek
Freie Universität Berlin, Berlin, Germany
e-mail: prechelt@inf.fu-berlin.de

C. Oezbek
e-mail: oezbek@inf.fu-berlin.de

1 Software Process Improvement as Innovation Introduction

The philosophy of lightweight processes considers software process improvement (SPI) a routine part of the development process itself (Highsmith and Cockburn 2001), although its proponents do not often use that particular term for it. Therefore, even skeptics of high-ceremony approaches such as CMMI (CMMI Product Team 2006) tend to concur that understanding how to *perform* software process improvement (SPI) is valuable.

Although Open Source software (OSS) development, one common class of lightweight process, has attracted considerable interest in the past few years (Scotto and Succi 2005), little so far has been written regarding SPI in this context—we only know of Dietze (2004) and Krafft (2009).

This lack of information is problematic, because results from other contexts, even if they concern lightweight process approaches, are not immediately applicable to OSS: Most SPI research tacitly assumes a closely-knit organization, in particular one in which there are individuals with formal power (often derived from organisational hierarchy) for making binding decisions. However, with few exceptions, no such individuals exist per se in the OSS processes we will concern ourselves with here, namely mid-sized OSS projects with 5 to 50 participants,[1] where any use of formal decision-making by even a project leader can easily alienate some of the volunteers (Shaikh and Cornford 2003; Raymond 1998).

In such typical OSS development processes, only a modest amount of authority is thus available; to a high degree, the process works by self-organization. OSS SPI thus depends largely on consensus or at least voluntary acceptance and support. And even though "binding decision" in a normal organization does not mean everybody (or even a majority) adheres to it, a formal lack of bindingness has profound consequences on SPI dynamics.

To reflect this OSS-specific SPI context, we take the view that each individual process improvement is an innovation introduction (Rogers 2003; Denning and Dunham 2006): There is a protagonist (the innovator), usually a member of the project, who advocates a certain change (the invention, i.e. the would-be innovation). The other project members (the target group) discuss and perhaps modify the invention and eventually adopt some form of it (or perhaps not). Adoption may occur gradually over time and turns the invention into an innovation.

The present article describes how we have studied OSS process innovation dynamics. The focus of the article is not on these studies' results, but rather on their methods: Which of them worked well, which did not, and why.

2 Structure and Contribution of this Article

The contribution of the article lies in describing a research method for studying phenomena occurring in OSS development processes. The method is applicable to many other process-related (rather than product-related) research questions in the OSS context besides just process innovation.

[1]This is opposed to the few large high-profile projects such as Apache, Linux, OpenOffice, or Mozilla, many of which *are* carried to a large degree by a formal organization (O'Mahony 2005).

In principle, the present study is a case study (in the sense of Yin (2003)) of a single case, namely our five-year attempt at understanding how process innovation takes place in Open Source Software projects. The case study database contains the inputs, chronology of events, and results of five scientific studies on the topic that we have pursued since 2004, as well as subjective opinions of the participants. The propositions investigated in the case study regard the feasibility, effectiveness, and efficiency of three different approaches to studying OSS process innovation.

However, we do not emphasize the formal structure of the case study here and rather frame the article like an experience report, because the narrative format is far easier to digest.

Note that reading this article can be somewhat confusing, because two very different levels of discussion constantly mix:

- The research on research methods for studying process innovation, which is the topic of the paper and
- the research on process innovation performed by these methods, which is *not* the topic of the paper, but needs to be mentioned frequently, as it is the data source of our empirical investigation.

We will first, in Sections 3.1 to 3.3, describe three studies (attempt 1, auxiliary study, and attempt 2) involving a method that can be characterized as being somewhere between Participant Observation (Jorgensen 1989) and Action Research (Avison et al. 1999; Davison et al. 2004). We conclude (Section 3.4), that such an approach is far too slow and too costly to collect a sufficiently large pile of observations on process innovation.

We then describe our attempt of using an alternative approach (attempt 3, Section 3.5): Polling many OSS participants for process innovation stories and then interviewing them in depth. This attempt failed entirely, yielding hardly any data at all.

Finally, we describe the successful third data collection approach of manually searching for innovation episodes in mailing lists (Section 4). We describe the qualitative data analysis process (based on the Grounded Theory Method of Strauss and Corbin (1998)) in Section 5, discuss the nature and validity of the results (Section 6), and conclude how and why this approach appears superior at least for the exploratory part of the long-term overall research process for our and other process-related research questions (Section 7).

In this narrative structure, there is a tendency to notice the process innovation research aspects only and loose track of the methods research aspects. To help you fight this tendency, Table 1 makes explicit the methods research roles that each of the sections serves.

While we do provide some detail on our application of Grounded Theory Method (GTM), it should be noted that details of applying Participant Observation and Action Research are explicitly *not* a topic of this article—although some aspects might be interesting, e.g. whether it is sensible to talk of Action Research in OSS at all, if no formal agreements are made.

The contribution of our article is threefold:

1. We point out the usefulness of passive qualitative analysis for understanding phenomena occurring in OSS projects. Such qualitative analysis is a valuable

Table 1 Research-on-research-method roles of the various sections, whose names rather suggest process innovation research

Section		Method	Results
3.1	Attempt 1: active innovation introduction of an information manager role	M	R
3.2	Auxiliary study: method comparison for triggering innovation from the outside	M	
3.3	Attempt 2: active innovation introduction of automated testing	M	R
3.4	Interim conclusion from the participatory approach		R
3.5	Attempt 3: polling for process innovation episodes	M	R
4	Successful attempt: finding episodes manually	M	R
5	Elements of the episode analysis method		R
6	Nature of the innovation research results obtained		R

M (for methodology) means the section describes a research method for understanding process innovation. *R* (for results) means the section describes what we found out about research methods

complement to the currently popular Mining Software Repositories (MSR) approach with its strong quantitative focus.

2. We show how such passive qualitative analysis (both data collection and data analysis) can be done for OSS process innovation.
3. Our case is a warning against over-optimism (and a consolation for those who have already suffered from it). Tool-building SE researchers may believe that OSS projects are a far more accessible playground for evaluating their research prototypes than are companies. Our studies suggest that neither access to a project nor acceptance of a tool are as easy as some would hope. This is in sync with prior results of others (Sarma et al. 2009; Barcellini et al. 2008).

3 Initial Attempts

3.1 Attempt 1: Active Innovation Introduction of an Information Manager Role

Robert Schuster, a student of ours, approached the GNU Classpath project (of which he was already a member) to introduce the role of an *Information Manager* (Schuster 2005). The process improvement goal was to improve the project's capabilities to handle the increasing amount of information and reduce the management overhead which had become noticeable as the project grew in size.

This sort of problem appears because of the strong use of mailing lists and IRC (Yamauchi et al. 2000), which are ill-prepared for information management, and because of the volunteer nature of participation, which makes it difficult to find project members willing to perform "arduous" housekeeping activities in comparison to "fun" coding tasks (Torvalds and Diamond 2001; Hertel et al. 2003). The information manager was thus created as a lightweight role-based process improvement to set up tool support (in particular a wiki) for managing information and to structure the tasks associated with its use, such as collecting decisions, to-dos, or how-tos. The actual execution of the tasks could be performed either by the Information Manager or any other project member; the main intention of the Information Manager role was motivation and orchestration. Robert contacted the project leader in January

2005. When he received positive feedback, he sent a proposal to the mailing list. In the ensuing discussion Robert agreed to set up a wiki and to assume the role of Information Manager for three initial months. In this time, he loaded the wiki with information taken from mailing list posts and advertised its use.

When Robert's engagement in the project ended in April 2005 (which he announced explicitly), project members confirmed in an exit survey that the Information Manager had been a valuable addition to the project. In September 2006, we studied the project's mailing list to see whether the use of the Information Manager role had been sustained after Robert's departure. We found that indeed it had. Its usage had become well accepted in the project. Furthermore, it had been *adapted* to project needs. In particular, much fewer decisions were recorded explicitly than we had originally envisioned. We found evidence that project members perceived the explicit formalization of general decisions as inefficient compared to the flexibility of common-sense ad-hoc decisions.

3.2 Auxiliary Study: Method Comparison for Triggering Innovation from the Outside

Sometimes, a would-be innovator will *not* be a project member already. This is particularly true for research settings. For instance in our innovation introduction research, having a project member such as Robert Schuster above was more of an exception rather than the rule. Likewise, other researchers will often want to evaluate tools they have built and would like one or more OSS projects to serve as users (Oezbek and Prechelt 2007). Ousider innovation might also be useful in non-research settings. Consider a company that is very interested in a particular OSS product and wants to contribute to it. This company may view making a process-improvement contribution as more efficient than making a code-writing contribution.

We therefore decided to perform a study on the following research questions:

1. How does an Open Source project react to an innovation proposal made by a non-member?
2. How much does it help when the proposal is accompanied by an upfront investment of work contributed to the project?

In order to have some of the control needed for answering research question 2, we decided to approach from the outside two comparable projects in the same domain (namely KDE[2] and Gnome[3]) with the same proposal and include an upfront investment in one of the cases (KDE) in the form of a prototypical implementation.[4]

Lacking a good idea for a process-related innovation (in particular regarding the question how to perform and package the upfront investment), we chose a product-related innovation proposal instead: a usability improvement for the Unix shell command line interface.

Specifically, we proposed to add assistance functionality to bridge the gap between a textual shell and a graphical user interface (GUI) for helping users to learn how to

[2]http://www.kde.org/

[3]http://www.gnome.org/

[4]Bergquist and Ljungberg (2001) call such behavior towards OSS projects a *code gift* and consider it an important means for gaining influence.

work with a textual shell. We spelled this idea out in a detailed and concrete proposal including a number of mock-up screenshots. For KDE, we also invested 150 hours of work to implement a subset of the proposal as a demonstration prototype.

We submitted the proposal to the Gnome mailing list, submitted both proposal and prototype (as 1200 lines of source code) to the KDE mailing list, and observed (and participated in) the ensuing discussions on both lists. We evaluated the reactions thus obtained via content analysis (Mayring 2002).

For Gnome, the proposal provoked 20 statements from project members, only two of which where positive. The proposal was not taken up. For KDE, the proposal and accompanying code gift provoked 22 statements from project members, 10 of which were positive. The proposal was not taken up and the code not used.

Detailed analysis of the email discussion revealed a number of effects, two of which are particularly important for judging the chances of a non-member to trigger an innovation. First, there were quite a number of misunderstandings (and quite some skepticism, too) that occured due to the proposing outsider's lack of project knowledge. For instance, the outsider-would-be-innovator was not aware that the project used the terms *beginner* and *newbie* in a differentiated and well-defined manner to mean different kinds of users. The resulting sub-discussions distracted a lot from the core topic. Second, even those project members who were positive about the proposal naturally assumed that it must be the proposer who would carry the proposal through; there was a strong sense of individual ownership.

Summing up, we conclude from the study that the chances of non-project-members for triggering an innovation are low, even if they invest substantial upfront work. Details of the study can be found in Quintela García (2006).

3.3 Attempt 2: Active Innovation Introduction of Automated Testing

Determined not to be discouraged by the results of the auxiliary study, we interpreted them from a different angle. If non-members can hardly trigger an innovation, we would have to become members first in our future studies. So the question would have to be: Can this be done efficiently?

We thus now attempted to introduce an innovation with only a fixed and rather limited amount of effort despite the fact that we started as a non-member of the project. As the innovation to be introduced we chose automated regression testing (Whittaker 2000; Jeffries and Melnik 2007) and the JUnit framework. Where applicable, automated unit testing promises strong benefits in terms of project stability, but it also requires a large and continuous amount of effort. We chose the project FreeCol.[5] because it offered the right size and would benefit from automated unit testing but did not yet use it.

To limit the effort in a systematic way, we prescribed ourselves the four-stage activity model shown in Fig. 1. Christopher Oezbek conducted the introduction in April and May 2007.

His activity resulted in 73 test cases being created as a gift for the project, integrated as an optional target in the build infrastructure, and advertised to all project members. Yet when he left the project, the test suite completely broke down

[5]http://www.freecol.org/

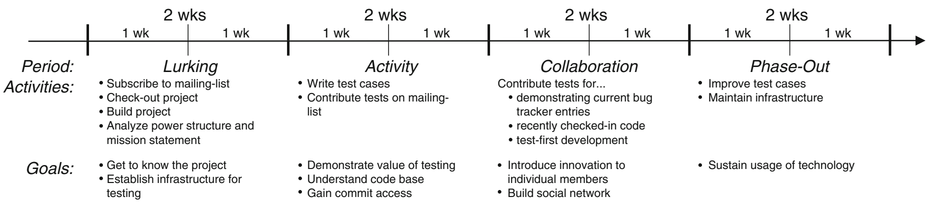

2 wks
1 wk
1 wk
2 wks
1 wk
1 wk
2 wks
1 wk
1 wk
2 wks
1 wk
1 wk
Period:
Activities:
Goals:
Lurking
• Subscribe to mailing-list
• Check-out project
• Build project
• Analyze power structure and mission statement
• Get to know the project
• Establish infrastructure for testing
Activity
• Write test cases
• Contribute tests on mailing-list
• Demonstrate value of testing
• Understand code base
• Gain commit access
Collaboration
Contribute tests for...
• demonstrating current bug tracker entries
• recently checked-in code
• test-first development
• Introduce innovation to individual members
• Build social network
Phase-Out
• Improve test cases
• Maintain infrastructure
• Sustain usage of technology

Fig. 1 Phases in the introduction process of automated regression testing in the project FreeCol

within three months due to API changes in several core modules. At the end of this time, not a single test case would execute correctly (most would not even compile), and no project member did anything to repair them. The innovation had clearly not been adopted.

After an explicit request by the project maintainer in September 2007, we repaired the test suite (thus breaking the intended 8-week timeframe), but again no immediate testing activity ensued. At this point we considered the introduction a failure and started to pursue other options, as described below.

However, in August 2009 we returned to the project to take stock and found, much to our surprise, that automated testing had now flourished: more than 270 test cases had been created overall, exercising 23% of the code. Writing or modifying testing code was now a regular activity and occurred in more than 10% of all monthly commits. Details of this study can be found in Oezbek (2010).

Summing up, we found that innovation introduction by an outsider is possible, yet takes longer than desired and is full of surprises, both negative and positive.

3.4 Interim Conclusion from the Participatory Approach

Of our four active and participatory innovation attempts (Information Manager, Gnome Shell Scaffolding, KDE Shell Scaffolding, JUnit introduction), which involved approximately 3, 1, 2, and 2 person months of effort on the researcher side, respectively, only the first and last were successful.

A lot of interesting information about process innovation in OSS projects was gained in these four studies. Nevertheless, we conclude that the participatory approach is not a viable basis for our research goal for four main reasons:

1. Innovator competence. For good chances of success, the innovator should be a highly competent individual, with deep knowledge of the project, strong technical capabilities, and excellent communication and social skills. However, such people are rare. In particular the required project knowledge severely limits the possible research scope.
2. Innovator variability. Given this first problem, we could still attempt to observe a large number of innovation attempts performed by just a very small number of suitable innovators in a small number of different projects. But this would seriously threaten the validity of the process innovation research: Not only would we rather look at a variety of projects (rather than look at the same few several times over), we also, and much more importantly, need to abstract from the specific personal characteristics of a particular innovator, or else the generalizability of the results might be totally unclear.

3. Scaling. The above two problems concern the innovation execution capacity. But even if that was large, there would still be limitations with respect to episode design capacity (deciding on the context, innovation, and introduction approach to be used in the next episode for maximal insight) and episode evaluation capacity (analyzing what has happened for extracting the insight). For instance, in the JUnit episode, the data analysis that determined how many test cases were modified when by whom took about four person weeks of effort. Even when assuming an average effort of just one person month for design and evaluation together, the participatory approach will scale only to a rather modest number of innovation episodes.
4. Waiting time. Even if all *three* of the above problems did not exist, say, due to additional researcher capacity, the sequential progress through a long series of episodes would be too slow: If the design of episode $N + k$ depends on the insights gained in episodes N through $N + k - 1$, the overall progress is limited by the time it takes for the results of the earlier episodes to arrive. However, this takes very long. In the Information Manager episode, we waited 17 months (from April 2005 to September 2006) before we could see the results. In the JUnit episode, we even waited 23 months (from September 2007 to August 2009).

We conclude that a different research approach is needed that involves much less effort for obtaining information on any single innovation episode, that involves a variety of innovators, and that avoids long waiting times.

3.5 Attempt 3: Polling for Process Innovation Episodes

The obvious answer to the above requirements is to collect innovation episodes that have already occurred all by themselves. There are thousands of eligible OSS projects, and even if only one in five of them performs process innovation occasionally, it should be possible to collect a broad array of episodes and analyze them.

In our attempt 3, we decided to poll OSS project participants for such episodes. We designed a web-based survey (Oezbek 2010) in October 2007 asking questions regarding the process innovation goal, roles of involved people, success factors and barriers, notable experiences with other project participants, etc. The survey was designed to take no more than 30 minutes to fill out. We asked for participation by sending emails to mailing lists frequented by rather diverse populations of Open Source developers from many different projects.[6] Yet, even though every single of these lists had more than 500 subscribers, over the course of eight weeks we received a total of only six responses,[7] and even the content of these was disappointing. Rather

[6]In particular user lists of tools and APIs commonly used for Open Source development such as http://lists.boost.org/mailman/listinfo.cgi/boost (a C++ library for common application development tasks), http://lists.gnu.org/mailman/listinfo/autoconf (a package for generating platform-specific scripts to be used in the build-process of applications), http://subversion.tigris.org/ds/viewForumSummary.do?dsForumId=1065 (a software for source code version management), http://lists.libsdl.org/listinfo.cgi/sdl-libsdl.org (a multimedia library for accessing input and graphics devices), and http://lists.mozilla.org/listinfo/support-bugzilla (a software for bug-tracking).

[7]We also received a small number of critical comments by email, for instance *“1) This is lame. 2) You get paid for this and we don’t”*.

than talking about how changes to the development process were achieved, three of the replies focused on changes to the code base rather than the process and two were process-related but lacked abstraction. Only one reply was of the type we were looking for. For instance, in reply to the question *"What was helpful to reach your goal?"*, the respondent wrote:

- *Offering the change as a parallel alternative rather than an either/or choice (this was CRITICALLY important)*
- *Patience and not being too pushy*
- *Doing it, not just talking about it*

This resonated with our own experience and looked quite promising. We would have liked to have a large number of such responses, would have analyzed them collectively, derived hypotheses and returned to the respondents with specific questions for evaluating these hypotheses. But with $N = 1$, this plan was in vain.

When we tried to understand the reasons underlying this failure, we found that the small number of responses overall (6), the low fraction of process-related ones (50%), and the low fraction of abstract ones within these (33%) left only three plausible explanations:

1. There is hardly any process innovation going on in OSS projects at all.
2. OSS process innovation participants are (for whatever reason) not willing to answer our survey.
3. OSS project members are hardly aware of their project's own process innovations. They are so product-focused that process abstractions are essentially tacit concepts.

We were not yet willing to assume explanation 1 and were unsure about explanation 2. Yet, the high fraction of product-focused replies suggests that explanation 3 is highly relevant. Evidence of a similar nature is reported by Krafft (2009). He studied innovation diffusion (as opposed to introduction) of packaging workflow technology and methodology in the large-scale setting of the Debian project[8] (as opposed to the small-scale settings of our medium-sized target projects) by means of a Delphi study involving three rounds of written interviews. He states that his observations confirmed the suspicion that the participants of the Delphi panel "had not previously thought much about the issues" he raised (p.68).

Summing up, the attempt to collect process innovation episodes by polling OSS project participants failed miserably, presumably because these developers do not think in terms of process innovation about what happens in their project.

4 Successful Attempt: Finding Episodes Manually

Among the emails (as opposed to web-survey responses) we received during attempt 3 were also a few pointers to projects said to have performed process innovations recently. Investigating these cases, we found that the project mailing lists were useful

[8]The Debian project collects many packages of OSS software, performs package management (in particular dependency modeling) on them, and develops a package management software. Krafft (2009, p.xiii) calls Debian "arguably the largest OSS project with over 1000 developers".

sources of information for understanding historic OSS process innovation episodes without having direct access to any of the participants: If there is direct discussion of an innovation introduction process in an OSS project, it will almost surely be reflected on the mailing list. Even much of the actual process phenomena (such as people taking up or ignoring the change) will often leave at least some traces in emails on the list. In particular, the mailing list is where decision-making happens or is made visible in OSS projects (Li et al. 2008).

We decided to search for past innovation episodes on many OSS mailing lists and analyze the messages comprising the episodes qualitatively. We picked Grounded Theory Methodology (Strauss and Corbin 1998) for the analysis because we felt that neither did the existing innovation and organization literature provide any promising starting point for a theory-driven, top-down research approach (see Section 5.4 for a validation of this impression) nor did we know from any qualitative research a coding scheme for the basic phenomena occuring in innovation episodes that could have served as a promising starting point for some other type of inductive, bottom-up approach.

This turned out to be the research method for which we finally settled and which generated most of our process innovation results. Our particular episode search method (described just below) and our particular analysis method (described in Section 5) thus mark the end (so far) of our quest to find a suitable research method. So where and how did we find the episodes?

4.1 Mass Data Source: Gmane

After some searching, we eventually decided to use Gmane[9] as our main data source. Gmane is a public mailing list archive that stores, in a uniform way, all emails from about 15,000 mailing lists, most of them belonging to OSS projects of all sorts. Gmane offers a common interface for browsing mailing lists and allows to bulk-download emails for offline reading, which made it convenient for our purpose.

There is nothing special about choosing Gmane; any other archive of OSS mailing lists could have done if it was large enough, similarly handy, and not biased towards any particular kind of OSS project.

4.2 Project Sample and Email Sample

Despite the seemingly simple approach, finding innovation episodes in mailing lists is still a major effort. A useful balance needs to be found between the effort expended on obtaining episodes and the effort expended on analyzing them: Find too few episodes and there is still not enough data for good insights; find too many and the analysis and results will become more shallow, which misses the purpose of qualitative research.

We made the following decisions for where to look for episodes:

1. The projects selected had to be real OSS projects. It was not sufficient that they used an OSS licence, they also had to have a truly open-participation software

[9] http://gmane.org

process. This ruled out a few commercial OSS projects (West and O'Mahony 2005).
2. Project size had to be between 5 and 50 participants. This ruled out many small (in particular: most academic) projects, which we deemed to be less interesting from an innovation introduction dynamics point of view. It also ruled out the very large OSS flagship projects such as Linux and Mozilla, which we deemed to be less interesting for two reasons. First, each such project has a rather unique organizational superstructure; see for instance Berdou (2007). Generalizing results obtained in such a setting to the vast majority of smaller projects would be dubious. Second, any one large project would have absorbed enough of our analysis capacity to severely reduce the breadth of the overall set of projects.
3. We wanted projects from multiple different application domains in order to provide a broad basis for insights and capture possibly different innovation dynamics.
4. We also wanted sets of similar projects in order to check whether some behaviors are perhaps strongly provoked by a certain project type.

In an iterative process towards a sufficiently large but not too-large set of episodes, we eventually settled for thirteen projects from seven different domains. Included are three workflow applications (Bugzilla, Flyspray, Request Tracker), two desktop environments (Rox, Xfce), two bootloaders (Grub, U-Boot), two design tools (ArgoUML, a UML CASE tool; gEDA, a set of electronic design automation tools), two emulators (Bochs, an x86 hardware emulator; KVM, a machine virtualization environment), one operating system (FreeDOS), one database management system (MonetDB).

A myriad of other projects could have been in their place; other than the above criteria, the choice we made was arbitrary, yet following pointers to interesting projects.

During this process we also decided to assess corresponding subsets of messages for each project, namely all messages sent to the respective list during 2007. This gave each project a weight proportional to its average daily activity (rather than also its age) and allowed us to see a larger number of episodes that related to process innovation issues that were topical at that time. In our case, this turned out to be the oft-discussed switch to a different versioning system (such as Git), which happened to be a particularly fruitful kind of innovation episode.

4.3 Innovation Episode Detection Method

The year 2007 samples from these thirteen projects yielded a total of 33,027 emails in 9,419 threads.

We began searching this substantial database for innovation episodes by reading threads with an interesting title[10] or an unusually large number of messages, while trying to keep a good sense of the activities and larger themes in the project such as upcoming releases or feature discussions. After finding the first episodes, we also

[10]For instance, the threads titled *"Changes to U-Boot Development Process"* and *"Making it Easier to Contribute to Bugzilla (2007 Edition)"* turned out to represent innovation episodes, while *"Talking about regressions..."* and *"Quietly promoting ArgoUML"* did not.

used salient terms extracted from these episodes, such as the name of tools to be introduced (say *"Git"* or *"Google Summer of Code"*), to perform full-text searches to find similar episodes in our data.

We later also attempted to extend the resulting body of episodes by using an information extraction algorithm (Siefkes 2007) to find emails similar to those we had already identified, but the results were not helpful.

Overall, the procedure resulted in 1,387 messages from 788 threads, representing 134 innovation episodes to be analyzed in detail.

This set may be incomplete. Due to the large number of messages overall, the screening may well have overlooked a few innovation episodes. However, since our investigation does not claim representativeness of the episodes studied, this is not a problem. All we need is a sufficient number of episodes, and 134 is quite satisfactory.

5 Results: Elements of the Episode Analysis Method

5.1 Grounded Theory Methodology (GTM)

As mentioned before, GTM is at the heart of our analysis approach. It is impossible to give a complete introduction into GTM here, but we will touch upon the basic ideas.

The goal of GTM is developing a *theory* of a certain *phenomenon* under study (here: introduction of process innovations into OSS projects) directly from raw data, which can take any form (in our case: email messages). Any step of the theory construction process is *grounded* in the data, that is, it explicitly refers to one or more specific observations as its justification.

More specifically, the researcher abstracts from the data by describing it in terms of *concepts* that are invented[11] by means of *abduction*, a reasoning method that involves a creative step and is at the heart of all scientific progress (Peirce 1883). This invention step is called *open coding*; subsequent *axial coding* determines relationships between the concept occurrences.

During both of these activities, the set of concepts is gradually augmented, refined, and cleaned up by means of *constant comparison*: comparing, again and again, several observations and the corresponding concepts to find similarities, dissimilarities, and inconsistencies.

The concepts describe events, characteristics, relationships, strategies, etc. and eventually the network of concepts contains the target theory: an explanation of the mechanisms underlying the phenomenon of interest. *Selective coding* focuses on the relevant pieces and develops a theory narrative from them: the *Grounded Theory (GT)*.

Such a GT can not claim to be generalizable to any specific domain, because data selection during analysis is driven by the needs of the analysis (*theoretical sampling*) rather than by representativeness (*random sampling*). It is also not reproducible in

[11] In order to avoid distorting the observations, it is important that exactly appropriate concepts be used. This is easiest to achieve when new concepts are being invented specifically for this analysis. When previously existing concepts are used, they have to be "re-invented".

the strong sense that a different researcher would necessarily have derived the same theory—this is even quite unlikely.

However, a GT is strongly *valid* in the sense that a different researcher can scrutinize its derivation and will agree that it is correct. A GT is also *understandable* in the sense that examples of each and every concept are immediately accessible in the raw data; the GTM process thus provides a kind of requirements traceability.

5.2 Tool Support: GmanDA

GTM was originally developed to understand the strategies used by people for coping with a certain kind of problem (Glaser and Strauss 1967). The raw data was originally interview notes, a modest number of fairly small and mostly independent documents.

In contrast, in our setting, while our documents (the individual email messages) are usually not large either, they are quite numerous (1,387), 134 subsets of them have strong relationships (episode), each such subset is ordered (chronologically), and often a subset needs to be partitioned in various ways (e.g. by author) during the analysis.

When starting our analysis, we found that the existing tools that support GTM-style data analysis such as Atlas.TI (Muhr 1991) have inadequate support for handling such large numbers of related documents in this manner. Eventually, we created our own tool (using Java and Swing) tailored to the email analysis job. It is called GmanDA (*Gmane Data Analyzer*,[12] see Fig. 2 for a screenshot) and is available under GNU GPL. GmanDA is optimized for browsing and annotating large sets of small documents, specifically email messages, and provides built-in support for data import directly from Gmane or mailboxes, searching (both text and concepts), annotating, time-line visualizations, and analysis data export.

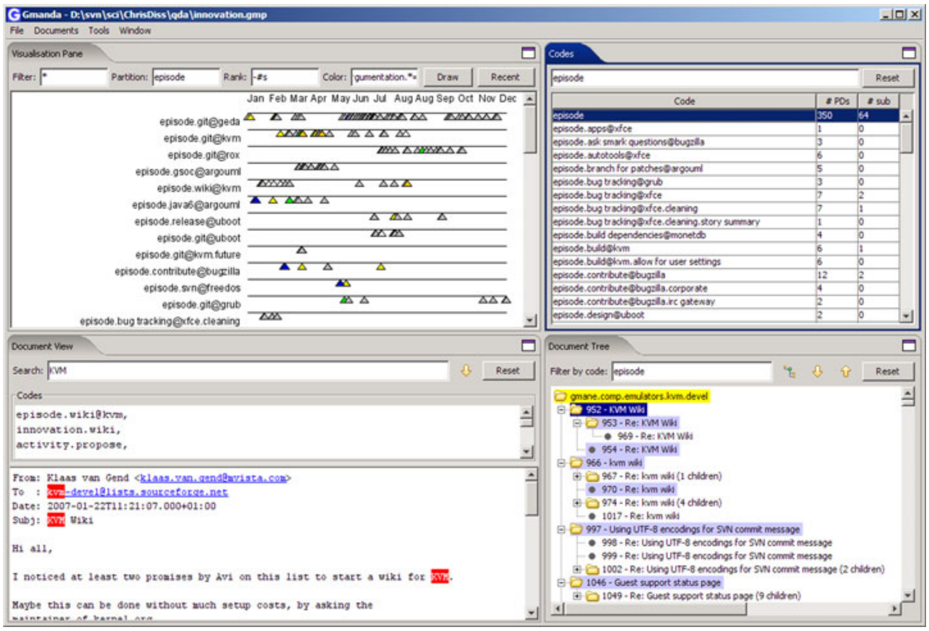
Designing and building GmanDA was a substantial effort, but the handling support offered by it was crucial for the research. We would probably never have completed the GTM analysis without it.

5.3 GTM in Practice

Equipped with GmanDA and plenty of advice on performing GTM (Strauss and Corbin 1998; Suddaby 2006a; Charmaz 2006; Corbin and Strauss 2008), we set out to develop our theories only to discover that qualitative methods do not come naturally to the empirical software engineer. Our instincts were itching to start *counting* occurrences of phenomena. They also told us to build a *meta-model* of the episodes using UML and turn the concepts and their relationships into elaborate class diagrams. These are noble causes in the quantitative world, but of little use for exploratory qualitative research based on GTM. This section will outline on a high level of abstraction how GTM was eventually performed in practice.

Step 1 is the task of identifying the innovation episodes as a set of related messages pertaining to a particular innovation introduction or attempt thereof. We have mentioned this above as if it was a data collection step, but it is in fact a data analysis step already: We need to identify innovation-related concepts in the messages to

[12] http://gmanda.sf.net



Filter: * Partition: episode Rank: -#s Color: gumentation.*=

episode

Code	# PDs	# sub
episode	350	64
episode..apps@xfce	1	0
episode..ask smark questions@bugzilla	3	0
episode..autotools@xfce	6	0
episode.branch for patches@argouml	5	0
episode.bug tracking@grub	3	0
episode.bug tracking@xfce	7	2
episode.bug tracking@xfce.cleaning	7	1
episode.bug tracking@xfce.cleaning.story summary	1	0
episode.build dependencies@monetdb	4	0
episode.build@kvm	6	1
episode.build@kvm.allow for user settings	6	0
episode.contribute@bugzilla	12	2
episode.contribute@bugzilla.corporate	4	0
episode.contribute@bugzilla.irc gateway	2	0
episode.design@uboot	2	0

Search: KVM

Codes

```
episode.wiki@kvm,
innovation.wiki,
activity.propose,
```

```
From: Klaas van Gend <klaas.van.gend@mvista.com>
To  : kvm-devel@lists.sourceforge.net
Date: 2007-01-22T11:21:07.000+01:00
Subj: KVM Wiki

Hi all,

I noticed at least two promises by Avi on this list to start a wiki for KVM.

Maybe this can be done without much setup costs, by asking the
```

Filter by code: episode

```
gmane.comp.emulators.kvm.devel
⊟ 952 - KVM Wiki
   ⊟ 953 - Re: KVM Wiki
      ● 969 - Re: KVM Wiki
   ● 954 - Re: KVM Wiki
⊟ 966 - kvm wiki
   ⊞ 967 - Re: kvm wiki (1 children)
   ● 970 - Re: kvm wiki
   ⊞ 974 - Re: kvm wiki (4 children)
   ● 1017 - Re: kvm wiki
⊟ 997 - Using UTF-8 encodings for SVN commit message
   ● 998 - Re: Using UTF-8 encodings for SVN commit message
   ● 999 - Re: Using UTF-8 encodings for SVN commit message
   ⊞ 1002 - Re: Using UTF-8 encodings for SVN commit message (2 children)
⊟ 1046 - Guest support status page
   ⊞ 1049 - Re: Guest support status page (9 children)
```

Fig. 2 A screenshot of GmanDA showing the four views (clockwise from *top left*) visualization, codes, document tree and document detail

determine what is an innovation episode and what is not, and we need to obtain a rough common-sensical understanding of the episode in order to determine its boundaries (what is part of it and what is not). The process is often iterative.

For instance, the researcher discovers an interesting email mentioning a new tool that could be used in the project. Some discussion of this half-proposal is easily found in the same thread. Is this the whole episode? Likely not. Related threads (e.g. regarding competing proposals) can be found by scanning for emails in the temporal vicinity and by keyword searches for salient persons, tools, or commonly used terms.

Step 2 is obtaining a somewhat deeper understanding of an episode as the basis for its conceptualization. Episodes are often complex, spread out over time, looping back to previous stages in decision processes (Li et al. 2008). Visualizing the emails of an episode based on their temporal distribution often helps to understand causal developments in an episode that are not obvious when looking only at a threaded representation. A good understanding of the overall project also makes it easier to make sense of an episode. Taking extensive notes and memos to summarize an episode is essential at this point, as a thorough understanding is the prerequisite for all further analysis (Langley 1999).

Step 3 is initial conceptualization: assigning the first few codes to the occurrences of phenomena in the emails. What should we focus on? Each email is full of interesting and rather different phenomena that tease the researcher to conceptualize them, yet not all of them will be useful for understanding the innovation process. There are uses of humor and rhetorical devices, interesting indicators of the nature of OSS development, direct or indirect statements about the participants' preferences for certain styles of communication, different implicit or explicit measures of progress, and so on and on.

After a while, we developed a habit of encoding primarily two kinds of concepts:

1. A modest set of basic concepts (Salinger and Prechelt 2008), which can generically be used for most emails, for example the primary activity of the author. These provide a coarse overall orientation within the episode and create a common framework for the annotations that helps with answering most kinds of questions asked of the data later.
2. Salient phenomena that promise to provide insight for this particular episode. For instance, do not encode humor or a rhetorical device just because it was used, but do code it if other messages look as if it may have made some other participant change her mind.

Step 4 is enriching and structuring the conceptualization with properties: Open coding in the above manner will quickly lead to a bewildering array of concepts. Many researchers will feel urged to bring order into this chaos; an urge we suggest to resist: Developing a coding scheme which orthogonally captures all phenomena occurring in the data is an enormous task and should not be attempted. Rather, when moving to axial coding the researcher should focus on one concept of interest at a time and clean it up (enriching it at the same time) using a process called *developing*.

For instance, when looking at innovation proposal emails we noticed that some proposals were quite concrete (with responsibilities assigned to individuals and the scope of the proposal limited in time and focus), while others were abstract, talking about project members and tasks in general. This observation gave rise to a *property* (called Concreteness) of the Proposal concept. At the same time, it removed doubts in other places whether something was indeed a Proposal or not.

However, accumulating properties and associated values without developing these as well, can quickly lead to "static typologies" (Charmaz 2006), which contain little insight beyond stating the existence of different types of something. A lack of explanations about the reasons and implications of each type leads to insipid results.

One method for developing a concept is applying the *paradigm* (Strauss and Corbin 1998), a sort of checklist that helps understand cause-effect relationships by asking standard questions of a concept, such as "What caused the phenomenon to occur?", "What implications did it have?", "What influences have affected the phenomenon?"; see Fig. 3.

A helpful second method is often *cross-tabulation*: Build a table of frequency counts for a particular concept: Which combinations of its properties occur how often? Which correlations exist among the properties? For instance, we looked whether *concrete* or *abstract* Proposals were more often *successful* and discovered an important new property: Concrete proposals tend to be more successful if (and only if) their *enactment scope* is small: *One-time* activities ("Let us clean up the bug tracker now that the release is out the door.") were more likely accepted than *recurring* processes ("Let us clean up the bug tracker each time in the two weeks after a release.").

Step 5 is extracting a meaningful core: While richly developed concepts are the goal of GTM's *axial coding* activity, *selective coding* looks to integrate some of these concepts into a theory. We found it useful to support selective coding by drawing

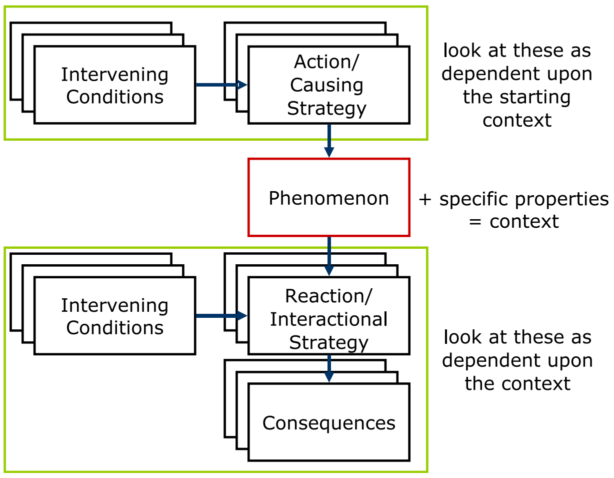

Intervening Conditions
Action/ Causing Strategy
look at these as dependent upon the starting context
Phenomenon
+ specific properties = context
Intervening Conditions
Reaction/ Interactional Strategy
look at these as dependent upon the context
Consequences

Fig. 3 The paradigm model (Strauss and Corbin 1998) extended with three more elements to emphasize that the phenomenon is caused by *actions* that occur in a *starting context* and are influenced by a separate set of *intervening conditions*. *Arrows* between elements indicate that the pointed-to element is caused or affected by the pointing-from element

diagrams of the connections a concept has with others. Again one has to start with a single concept of interest and look for relevant relationships to other concepts, using the properties previously developed. Graphically, this results in a centrally drawn concept connected to others around it; an example is shown in Fig. 4 and resembles a Mind Map (Buzan and Buzan 1993; Farrand et al. 2002) or Concept Map (Novak and Gowin 1984; Novak and Cañas 2006).

5.4 Matching with Established Theories

There are various existing theories from social science that are potentially helpful to understand OSS process innovation. However, employing the content of such theories during a GTM analysis can constrain the attention and creativity of the researcher (Suddaby 2006b), thus mislead the analysis process, and distort the resulting theory. On the other hand, we should not afford the luxury of simply ignoring such large packages of potentially relevant prior work.

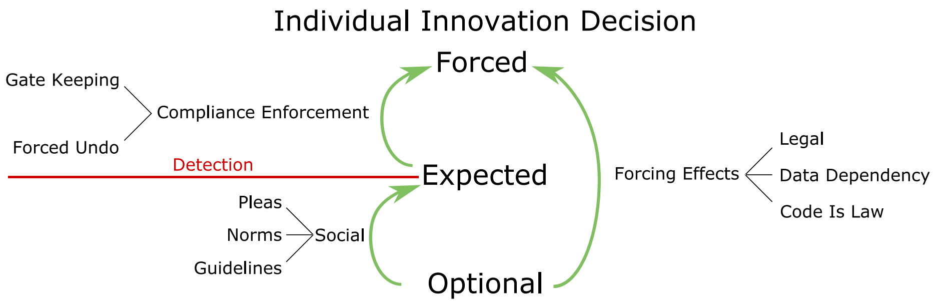


Fig. 4 The proposed model explains how an individual innovation adoption decision is shaped by the social actions (*left side*) and attributes contributed by the innovation (*right side*)

We therefore looked at such theories only *after* we had performed a substantial part of the GTM analysis. We considered a number of different theories and chose to work through each of them in the same manner: Knowing the results of our own analysis (as sketched in Section 6 below), we read the theories looking for elements that related to (or could be related to) any of our findings. When finding such an element X, we wanted to look whether the theory would make any statement S about X that was not yet present in our findings. If yes, we wanted to validate S against our data and, if successful, thus extend our findings. The result would thus still be grounded in data, but the theories would work as a creativity enhancer rather than an inhibitor.

The theories to which we applied this procedure were Path Dependence (David 1985; Arthur 1989), Garbage Can Model (Cohen et al. 1972), Actor-Network Theory (Law 1992), and Structuration Theory (Giddens 1984). We also looked at other conceptual frameworks that might be applicable but would perhaps not be considered theories, namely Communities of Practice (Lave and Wenger 1991; Wenger 1999), self-organization (Luhmann 1984), social network analysis (Scott 1988, cf.) and heterarchical organizations (Hedlund 1986; Windeler 2001).

The result was disappointing: None of the concepts from the given theories applied to a concept from our findings in any interesting way. As the main reason for this lack of applicability we found different specificity: All our findings are quite concrete and talk about specific constraints typical for certain kinds of innovations in certain kinds of OSS project settings. In contrast, the conceptual frameworks listed above are all on a much higher level of abstraction. On the one hand, this makes them far more general, on the other hand it means they have little to say that is even remotely as concrete as our findings and thus can hardly add to them.

For instance in the Garbage Can Model, a central proposition is that actors, solutions, and problems should be considered as independent "streams" (Cohen et al. 1972); a notion which is absent from all concepts we found using GTM. This can either mean that OSS projects are no Garbage Cans or that these independent streams exist in OSS projects but were not salient enough (at least to us) during our analysis to appear in the results. If existing "streams" were indeed not salient enough, this can in turn either mean they were not relevant in our setting, were relevant but not visible in the emails, or we were too unperceptive to notice their presence.

In order to rule out the latter possibility, we repeated the matching on a concrete level, that is, not against our abstract findings but rather against specific episodes as such. Could any of the concepts from the theory be identified in an episode? Could the theory then be applied from that starting point? Would that lead to any new insights?

The results from this procedure were mixed. We could identify occurrences of some concepts from some of the theories in a number of our episodes, but none of these cases led to any new insight. Nothing from the theories made it into our results.

For instance with respect to the Garbage Can Model, we found instances of the Garbage Can concept of "arena of discourse" into which discussion can be concentrated. Experienced participants used such arenas skillfully to direct heated discussion about controversial topics or structure complicated topics. This observation, however, did still not connect these episodes with the rest of the Garbage Can Model. There were several similar cases.

Most of the newly identified concepts were just different terminology for something (or some subtle aspect thereof) we had already recognized as relevant. For instance, Actor-Network theory emphasizes the role of artefacts to translate the actions of human actors in the network. While this sounds quite impressive, in our case it meant for instance that control of a password file establishes power relationships in a project—a fact of which we were well aware.

Although the amount of insight gained was rather small, the matching process provided two insights: (1) Our findings are not in conflict with any of these well-established theories, which raises our confidence. (2) The matching confirmed that the GTM approach taken was a sensible one and can be considered both effective and, although it was quite time-consuming, even efficient: Any other approach would probably not have digged out all of our results.

6 Results: Nature of the Innovation Research Results Obtained

It is not the purpose of the present article to present (or even summarize) all of our results R from the actual process innovation research performed using the method described above, because the contribution of this article is the *derivation* of this particular method. In light of the methods discussion, however, it is relevant to understand the common characteristics of those results R. These can be summarized as follows.

No Grand Theory First a negative result: We have *not* found an overall theory of process innovation in OSS projects. We believe that such a theory could be created, but that it would require a far larger amount of data than the 134 episodes we had.

However, we do not think the lack of an overall theory is a problem. As we have seen in Section 5.4, general theories tend to be sufficiently unspecific in that they have hardly anything useful to say about a concrete situation in the OSS process innovation domain. Therefore, we are quite happy with narrower mini-theories as long as they are useful.

Separate Mini-theories About One Topic Each Among the 134 episodes, there were a number of recurring or very similar innovation proposals, such as a change to a different version management system, but also many unique ones. If a theory only talked about essentially the same proposal in different projects, the results would be very narrow indeed. Fortunately however, we found interesting concepts that cut across far wider classes of proposals and eventually boiled them down to five mini-theories about the following topics (or *core categories* in GTM terminology):

– Hosting: issues relevant when an innovation involves setting up a service on a server.
– Enactment scope: concepts describing when and how proposals are more easily adopted if they propose to do something once, compared to proposals of doing the same regularly.
– Forcing effects: various factors (technical or human, implicit or explicit, inherent or additional) that urge a project member to individually adopt a process innovation.

- Partial migration: when and how incomplete adoptions of an innovation can actually be *helpful* for the innovation process.
- Adapter: when and how a project can neutralize some members' unwillingness to adopt an innovation that has to be adopted on the project level, i.e. by everybody.

Surely more such topics for mini-theories could be found. For some of them, additional data will be needed, others are possibly still waiting to be discovered even within our 134 episodes.

Innovation Management Patterns Our overall goal with this work was to help a would-be innovator to be more successful. And indeed each of our mini-theories allows formulating advice for innovators that suggests for instance what to expect or be prepared for, what to do or not do, or how to propose or argue. The manner in which this advice helps is similar to that of a design pattern. The advice describes a set of situations recurring repeatedly in OSS process innovation attempts, enumerates possible actions, and discusses the consequences to expect from each action. Also just like a design pattern, a master innovator probably will get it right even without our advice, but the pattern is definitely helpful for beginners and intermediates—and masters are quite rare in this realm. Manns and Rising (2004) presents similar patterns for the corporate domain.

Factors Mix In each of the five mini-theories, the mechanisms at work are a mix of factors from diverse areas:

- Technical: what is feasible? Which work modes result from it? Resource consumption issues. Compatibility issues.
- Economical: how much upfront investment is needed? How much recurring effort is needed? By whom? How high is the return? For whom? How certain am I about these estimates?
- Psychological: am I willing to learn? Am I willing to change my work style? Am I risk-adverse? Are the proposal's pros and cons presented well? Are they easy to understand?
- Sociological: do I like the proposer? Is the proposal fashionable? Do I like fashions? Does the proposal modify the distribution of power?

From a scientific point of view, this is clearly the biggest strength of the qualitative (in particular: GTM) research approach: That the mini-theories describe how factors of all these different kinds play together to create the observed innovation adoption dynamic. Any model covering just one of the areas would miss essential influences.

As an example, we provide a simplified sketch of the mini-theory about *Forcing Effects*. Its main concepts are shown in Fig. 4. The theory talks about the fact that some innovations inherently mandate adoption by all project participants *("forced")*. For instance, when my project switches to a different version management system, I need to switch too, or else I cannot check-in my future work results any more.

Other innovations do not have this character; adopting them is a separate decision for each individual project member (*"optional"*), say, the adoption of the FindBugs static code analyzer. Every member may choose whether to run this tool or not and even if it is run automatically in central builds, each member can choose whether to look at the results.

However, the project can change this status by additional actions. For instance, *social norms* can be formulated that call using FindBugs a should-do (*"expected"*). If the code acceptance process requires that FindBugs must be silent about the new code, this would elevate FindBugs use even to *forced* level (*"gate keeping"*). As a final step, the acceptance check may even be automated, thus turning the social influence into a technical constraint (*"code is law"*, Lessig (2000)). Such forcing effects have interesting psychological and sociological repercussions.

On the other hand, the *forced* use of a new version management system can be softened to *expected* level by providing a gateway from the former system to the new one, i.e. by applying an *Adapter*. Since Adapters tend to be imperfect, they are covered by a mini-theory of their own.

7 Conclusion and Further Work

The insights from our search for a research method to understand the dynamics of the introduction of process innovations in OSS projects can be summarized as follows:

1. **Participatory methods** are hardly viable for a broad investigation of these issues, because the number of cases that can be observed in this manner does not scale up sufficiently. Participatory methods may still be worthwhile if secondary motives exist, such as wanting to become a member of the project for some other purpose (Oezbek et al. 2008).
2. Participatory methods may also become useful later in the research process for evaluating specific hypotheses regarding innovation introduction by following a specific behavior, that is, by exerting some control over the unfolding of events in an episode.
3. It is likely these first two results hold for many other OSS research topics having to do with process dynamics, not just innovation introduction—at least as long as they have episodic character and the frequency (rare) and granularity (a few person-weeks) are similar.
4. Collecting process episodes by **polling** OSS developers in survey manner is problematic. It may simply fail completely (as it did in our case), because there is insufficient interest. Even if a sufficient number of answers is received, their quality may be too low, because the developers may think of their work in very different terms than the researcher would need.
5. The specific result from our attempt at polling, when compared with the results of the manual episode search described below, indicates that software process improvement is apparently not a notion that is **consciously important** to OSS developers. They do it, but they appear to be hardly aware of it and rarely think about it on a meta-level.
6. Software process improvement episodes exist in OSS projects but appear to be **infrequent**. Our data is biased towards over-estimation and contained 10 innovation-related episodes per project per year, of which two to three were successful innovation introductions.
7. **Mailing list archives** are a suitable data source for identifying such episodes and for retrieving information about them that is sufficient for a qualitative analysis.
8. **Grounded Theory Method (GTM)** is a feasible and effective analysis method for deriving narrow mini-theories regarding such episodes. Comparing process

innovation episodes with different topics and from different OSS projects allows to identify a modest number of recurring phenomena.

9. In contrast, using **existing theories** of potentially relevant type such as Path Dependence Theory, the Garbage Can Model, Actor-Network Theory, or Structuration Theory or conceptual frameworks such as Communities of Practice, self-organization, social network analysis, or heterarchical organizations did not lead to insights regarding OSS process innovation introduction. The primary reason appears to be a lack of specificity in these theories and frameworks.
10. The mini-theories found via GTM are not just of academic interest, but also allow formulating **constructive advice** that will help a would-be OSS process innovator achieve her aim.

Analysis of many further episodes beyond our 134 will be required for complementing the five innovation dynamics topics we describe by further ones, and for connecting these pieces into a coherent whole.

It will presumably also be useful to complement the analysis of mailing list messages by the analysis of other artefacts. Depending on the type of process change, these other artefacts might be web pages (in particular wiki pages, which often have a public version archive), bug tracker data, or version histories of pertinent files in the source code management system.

Finally, approaching developers for interviews *after* having identified and analyzed an episode may be a source of particularly rich information (O'Mahony 2003; West and O'Mahony 2008).

Acknowledgements The authors would like to thank Robert Schuster, Luis Quintela-Garcia, and Alexander Roßner, who conducted three of the studies as part of their theses, and the participants of the Open Source projects without whom this work would not have been possible. We thank the reviewers for insisting on some changes that make this article a less complicated read.

References

Arthur WB (1989) Competing technologies, increasing returns, and lock-in by historical events. Econ J 99(394):116–131. http://www.jstor.org/stable/2234208

Avison DE, Lau F, Myers MD, Nielsen PA (1999) Action research. Commun ACM 42(1):94–97. doi:10.1145/291469.291479

Barcellini F, Détienne F, Burkhardt JM, Sack W (2008) A socio-cognitive analysis of online design discussions in an Open Source Software community. Interact Comput 20(1):141–165. doi:10.1016/j.intcom.2007.10.004

Berdou E (2007) Managing the bazaar: commercialization and peripheral participation in mature, community-led F/OS software projects. Doctoral dissertation, London School of Economics and Political Science, Department of Media and Communications

Bergquist M, Ljungberg J (2001) The power of gifts: organizing social relationships in Open Source communities. Inf Syst J 11(4):305–320. doi:10.1046/j.1365-2575.2001.00111.x

Buzan T, Buzan B (1993) The Mind Map book. BBC Books, London

Charmaz K (2006) Constructing grounded theory: a practical guide through qualitative analysis, 1st edn. Sage Publications Ltd. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&amp;path=ASIN/0761973532

CMMI Product Team (2006) CMMI for development, version 1.2. Tech. Rep. CMU/SEI-2006-TR-008, Software Engineering Institute

Cohen MD, March JG, Olsen JP (1972) A garbage can model of organizational choice. Adm Sci Q 17(1):1–25. http://www.jstor.org/stable/2392088

Corbin JM, Strauss AL (2008) Basics of qualitative research: techniques and procedures for developing grounded theory, 3rd edn. SAGE

David PA (1985) Clio and the economics of QWERTY. Am Econ Rev 75(2):332–337. http://www.jstor.org/stable/1805621

Davison R, Martinsons MG, Kock N (2004) Principles of canonical action research. Inf Syst J 14(1):65–86. doi:10.1111/j.1365-2575.2004.00162.x

Denning PJ, Dunham R (2006) Innovation as language action. Commun ACM 49(5):47–52. doi:10.1145/1125944.1125974

Dietze S (2004) Modell und optimierungsansatz für Open Source Softwareentwicklungsprozesse. Doktorarbeit, Universität Potsdam. http://opus.kobv.de/ubp/volltexte/2005/168/pdf/DIETZE.PDF

Farrand P, Hussain F, Hennessy E (2002) The efficacy of the 'mind map' study technique. Med Educ 36(5):426–431. doi:10.1046/j.1365-2923.2002.01205.x

Giddens A (1984) The constitution of society: outline of the theory of structuration. University of California Press, Berkeley

Glaser BG, Strauss AL (1967) The discovery of grounded theory: strategies for qualitative research. Aldine de Gruyter, New York

Hedlund G (1986) The hypermodern MNC—a heterarchy? Hum Resour Manag 25(1):9–35. doi:10.1002/hrm.3930250103

Hertel G, Niedner S, Herrmann S (2003) Motivation of software developers in Open Source projects: an internet-based survey of contributors to the Linux kernel. Research Policy 32(7):1159–1177. doi:10.1016/S0048-7333(03)00047-7, http://www.sciencedirect.com/science/article/B6V77-48BV04S-2/%2/5f8c87b7acd7f8541a97820b22805248 (Open Source Software development)

Highsmith J, Cockburn A (2001) Agile software development: the business of innovation. IEEE Softw 18(5):120–122

Jeffries R, Melnik G (2007) TDD—the art of fearless programming. IEEE Softw 24(3):24–30. doi:10.1109/MS.2007.75

Jorgensen DL (1989) Participant observation: a methodology for human studies, applied social research methods series, vol 15. Sage, Newbury Park, CA

Krafft MF (2009) A Delphi study of the influences on innovation adoption and process evolution in a large open-source project—the case of Debian. PhD thesis, University of Limerick, Ireland, 2009.10.01—version submitted to examiners

Langley A (1999) Strategies for theorizing from process data. Acad Manage Rev 24(4):691–710. http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=%2553248&site=ehost-live

Lave J, Wenger E (1991) Situated learning: legitimate peripheral participation. Cambridge University Press. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&amp;path=ASIN/0521423740

Law J (1992) Notes on the theory of the actor-network: ordering, strategy and heterogeneity. Syst Pract 5(4):379–393

Lessig L (2000) Code and other laws of cyberspace. Basic Books, New York

Li Q, Heckman R, Crowston K, Howison J, Allen E, Eseryel UY (2008) Decision making paths in self-organizing technology-mediated distributed teams. In: Proccedings of the international conference on information systems (ICIS) 2008, Association for Information Systems

Luhmann N (1984) Soziale Systeme. Grundriß einer allgemeinen Theorie. Suhrkamp, Frankfurt am Main, http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&amp;path=ASIN/3518282662

Manns ML, Rising L (2004) Fearless change: patterns for introducing new ideas. Addison-Wesley. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&amp;path=ASIN/0201741571

Mayring P (2002) Einführung in die qualitative Sozialforschung. Beltz, Weinheim. http://hdl.handle.net/2003/23120

Muhr T (1991) Atlas/ti — a prototype for the support of text interpretation. Qual Sociol 14(4):349–371. doi:10.1007/BF00989645

Novak JD, Cañas AJ (2006) The theory underlying concept maps and how to construct them. IHMC CmapTools 2006-01, Florida Institute for Human and Machine Cognition

Novak JD, Gowin DB (1984) Learning how to learn. Cambridge University Press, New York

Oezbek C (2010) Introducing automated regression testing in Open Source projects. Tech. Rep. TR-B-10-01, Freie Universität Berlin, Institut für Informatik, Berlin, Germany. http://www.inf.fu-berlin.de/inst/ag-se/pubs/OSSmediateTR-2008%.pdf

Oezbek C (2010) Introducing innovations into Open Source projects. Doctoral thesis, Freie Universität Berlin (to appear)

Oezbek C, Prechelt L (2007) On understanding how to introduce an innovation to an open source project. In: Proceedings of the 29th international conference on software engineering workshops

(ICSEW '07), IEEE computer society, Washington, DC, USA, reprinted in UPGRADE. The European Journal for the Informatics Professional vol 8(6), pp 40–44

Oezbek C, Schuster R, Prechelt L (2008) Information management as an explicit role in OSS projects: a case study. Tech. Rep. TR-B-08-05, Freie Universität Berlin, Institut für Informatik, Berlin, Germany. http://www.inf.fu-berlin.de/inst/ag-se/pubs/OSSmediateTR-2008%.pdf

O'Mahony S (2003) Guarding the commons: how community managed software projects protect their work. Res Policy 32(7):1179–1198. doi:10.1016/S0048-7333(03)00048-9, http://www.sciencedirect.com/science/article/B6V77-48PM412-1/%2/06fb5992706ae69b561640f2a93a6d4e (Open Source Software development)

O'Mahony S (2005) Nonprofit foundations and their role in community-firm software collaboration. In: Feller J, Fitzgerald B, Hissam SA, Lakhani KR (eds) Perspectives on free and open source software. The MIT Press Ltd., Cambridge, MA, chap 20, pp 393–414

Peirce CS (1883) Studies in Logic, Little, Brown, and Company, Boston, MA, chap A Theory of Probable Inference, pp 126–181

Quintela García L (2006) Die Kontaktaufnahme mit Open Source Software-Projekten. Eine Fallstudie. Bachelor thesis, Freie Universität Berlin. http://projects.mi.fu-berlin.de/w/bin/view/SE/ThesisCommandLi%ne

Raymond ES (1998) Homesteading the Noosphere. First Monday 3(10):n/a. http://www.firstmonday.org/issues/issue3_10/raymond/index.htm%l

Rogers EM (2003) Diffusion of innovations, 5th edn. Free Press, New York

Salinger S, Prechelt L (2008) What happens during pair programming? In: Proceedings of the 20th annual workshop of the psychology of programming interest group (PPIG '08), Lancaster, England. http://www.ppig.org/workshops/20th-programme.html

Sarma A, Maccherone L, Wagstrom P, Herbsleb J (2009) Tesseract: interactive visual exploration of socio-technical relationships in software development. In: ICSE '09: Proceedings of the 2009 IEEE 31st international conference on software engineering. IEEE Computer Society, Washington, DC, USA, pp 23–33. doi:10.1109/ICSE.2009.5070505

Schuster R (2005) Effizienzsteigerung freier softwareprojekte durch informationsmanagement. Studienarbeit, Freie Universität Berlin. https://www.inf.fu-berlin.de/w/SE/ThesisFOSSIM

Scott J (1988) Social network analysis. Sociology 22(1):109–127. doi:10.1177/0038038588022001007, http://soc.sagepub.com/cgi/content/abstract/22/1/109, http://soc.sagepub.com/cgi/reprint/22/1/109.pdf

Scotto M, Succi G (eds) (2005) The first international conference on Open Source Systems, Genova

Shaikh M, Cornford T (2003) Version management tools: CVS to BK in the linux kernel. In: Feller J, Fitzgerald B, Hissam S, Lakhani K (eds) Taking stock of the bazaar: the 3rd Workshop on open source software engineering. IEEE Computer Society, Portland, Oregon, pp 127–132

Siefkes C (2007) An incrementally trainable statistical approach to information extraction based on token classification and rich context models. PhD thesis, Freie Universität Berlin, Berlin

Strauss AL, Corbin JM (1998) Basics of qualitative research: techniques and procedures for developing grounded theory, 2nd edn. SAGE. http://www.amazon.co.uk/exec/obidos/ASIN/0803959400/citeulike%-21

Suddaby R (2006a) From the editors: what grounded theory is not. Acad Manage J 49(4):633–642

Suddaby R (2006B) From the editors: what grounded theory is not. Acad Manage J 49(4):633–642

Torvalds L, Diamond D (2001) Just for fun: the story of an accidental revolutionary. HarperCollins

Wenger E (1999) Communities of practice: learning, meaning, and identity. Cambridge University Press. http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20%&amp;path=ASIN/0521663636

West J, O'Mahony S (2005) Contrasting community building in sponsored and community founded open source projects. In: 38th annual Hawaii international conference on system sciences. IEEE Computer Society, Los Alamitos, CA, USA, vol 7, p 196c. doi:10.1109/HICSS.2005.166

West J, O'Mahony S (2008) The role of participation architecture in growing sponsored open source communities. Ind Innov 15(2):145–168, 10.1080/13662710801970142

Whittaker JA (2000) What is software testing? and why is it so hard? IEEE Softw 17(1):70–79. doi:10.1109/52.819971

Windeler A (2001) Unternehmungsnetzwerke: konstitution und strukturation. VS Verlag

Yamauchi Y, Yokozawa M, Shinohara T, Ishida T (2000) Collaboration with lean media: how open-source software succeeds. In: CSCW '00: proceedings of the 2000 ACM conference on computer supported cooperative work, ACM, New York, NY, USA, pp 329–338. doi:10.1145/358916.359004

Yin RK (2003) Case study research: design and methods. Sage. http://www.amazon.de/Case-Study-Research-Design-Methods/dp/07%61925538/

Lutz Prechelt is full professor of Informatics at Freie Universität Berlin since 2003. Until 2000, he worked as senior researcher at the School of Informatics, University of Karlsruhe, where he also received his Ph.D. in Informatics in 1995. In between, he was with abaXX Technology, Stuttgart, first as the head of various departments, then as Chief Technology Officer. His research interests include empirical software engineering (using both qualitative and quantitative methods), measurement and benchmarking issues, and research methodology. Current research topics revolve around open source software development, agile methods, and web development platforms. Prechelt is a member of IEEE CS, ACM, and GI and is the editor of the Forum for Negative Results (FNR) within the Journal of Universal Computer Science (J.UCS).

Christopher Oezbek is head of software development at medical technology company Scopis since March 2011. Before, he pursued and completed his Ph.D. in computer science on the topic introducing software engineering innovations into Open Source projects at Freie Universität Berlin. Christopher Oezbek holds a M.S. in computer science from Georgia Institute of Technology and a Vordiplom in informatics from Universität Karlsruhe (TH). His research interest focuses on the development processes of individuals and small teams.