

Introduction of Innovation M1

Christopher Oezbek
oezbek@inf.fu-berlin.de

2008-05-05

Abstract

This document provides an overview of the current state of my research into the introduction of innovation as of April 2008.

1 Introduction

This document does not and actually cannot begin with definitions of introduction or innovation, as part of the goal of this research is to determine what innovation and introduction in the context of Open Source projects actually are. To give the reader a workable idea of the object under investigation it seems most useful to show an example of an innovation introduction that actually occurred:

In the project gEDA (the GNU Electronic Design Automation tool) that writes a suite of Open Source programs for electronic circuit board design and associated activities, the tool CVS was used to manage the source code of the project. Over the year 2007 CVS was gradually abandoned by the project members in favor of the decentralized source code management (SCM) system git. First, at the end of 2006 one of the main developers of gEDA announced that he had set up a git system that would be mirroring the central CVS system so that people could try out whether they liked git or not.

The following months January to March saw several implications of the existence of such trial system: First, a web-application was announced based on the existing git-mirror which could be used to follow and track changes in CVS. Second, there was an unanswered request to the project to change the changelog format in CVS to be more compatible with the git test system. Third, it was suggested to use git as a public playground for experimental coding that should not go into CVS such as the work done by students during a

Google Summer of Code. Fourth, inspired by the way git identifies repository state by SHA-1 hashes, the project was considering using a similar mechanism to manage a type of project resources.

After this, mentioning of git on the mailing-list subsided until, at the end of May, the developer who had installed the git trial system sent a link to the mailing-list about a blog posting discussing various alternatives considered for a version control system at a large and well known Open Source project. In this discussion git came out as the winner and many reasons in favor of git were given. Less than one week later, the maintainer of the project announced his plan to switch over the core repository of the project to git. While this decision got executed by the project within the next two and a half weeks, a sub-project maintainer underwent the process of considering and finally deciding to move as well with the sub-project repository to git.

This swift transition then was followed by migration problems, such as the need to now actually do the change-log format change already discussed at the beginning of the year to accommodate git or to get git client software installed on the many different platforms that contributors used.

From the end of July to the middle of September the switch to git became more and more solid by the migration of another sub-project by the maintainer to git and the set-up of mirror for the main git repository.

By the middle of November usage of git had become fluently enough so that projects members for instance kept public branches with platform dependent patches and private patches in parallel to the main branch, making full use of the decentralized abilities of git. To close the circle, at the end of the year, two sub-projects that had not switched to git were given git-mirrors. It seems that these are not there for trials as the year before, but rather as an indicator that the

project members have come to the point that they prefer to use git instead of CVS at all levels.

2 Action research and introduction research

Eight months ago after the summer term 2007, it became increasingly clear that the action research methodology envisioned in our FLOSS Workshop publication at ICSE 2007 [?] would be unrealistic due to a misjudgement about the amount of effort necessary to achieve introduction into Open Source projects. While previous research of my two bachelor students Alexander Rossner [4] and Luis Quintela-Garcia [3] has demonstrated that a Zero Acquaintance approach can be feasible for an external innovator, this kind of approach also would unrealistically restrict the kinds of results about the innovation introduction that we would be able to achieve.

Realization that the manual execution of introduction in Open Source projects is too large an endeavour to execute as a PhD topic lead to a first shift in strategy which tried to query the Open Source project members for their experiences with introducing innovations into their own projects. Results of the survey conducted in September and October 2007 were disappointing and turnout remained too low to generalize any results. The two best factors we could find for explaining these results were (1) general fatigue with replying to surveys regarding Open Source, leading to request for participation even rudely rejected, and (2) too high a conceptual level of the questions posed.

3 Mailing-list analysis pilot

From this it seemed the next logical step to examine existing projects themselves for innovation introductions. To pilot this, five developer mailing-lists with more than one thousand cumulated e-mails and sufficient activity within the last 12 months were read online (one of them being the gEDA mailing-list geda-dev from which the example introduction was taken). The following insights could be gathered by this pilot:

1. A consistent interface as provided by a mailing-list hoster like *gmane.org* is necessary so that analysis is possible,

2. activities regarding novel technology, processes or rules take up only a small part of total communication in mailing-lists¹ and often hide in sub-threads with unsuspecting titles, thus finding such activity is looking for a needle in a haystack,
3. managing results requires more thought than keeping a set of notes of each interesting e-mail, and
4. innovation processes are highly variable in type of innovation, actors setting them off and outcome.

The implication of this then had to be that tool support is necessary for

1. reading large number of messages in consistent format,
2. annotating messages in a way that remains accessible even when the number of annotated messages grows, and
3. aggregating and comparing annotations to form higher levels of insight.

To enable this, I began to develop the qualitative data analysis (QDA) tool *Gmane Data Analyser* (*GmanDA*) after trials with available QDA tools like ATLAS.ti and Weft.QDA revealed that they were not suitable to deal with the large amount of hierarchical-ordered primary documents of which a single mailing-list can consist.

4 Tool design and methodological questions

GmanDA was intentionally kept at a minimum in size and highly specialized for the task at hand to deal with mailing-list data in *mbox* format downloaded from *Gmane*. A screenshot can be seen in Figure 1. The central data structure is a primary document (PD) representing a single e-mail sent to a mailing-list. Each PD then can contain any number of child PDs - denoting replies - and be annotated with a single annotation string.

Hand in hand with the development of GmanDA the mailing-lists from the pilot analysis were recoded

¹We could probably put a number to this with relatively little effort.

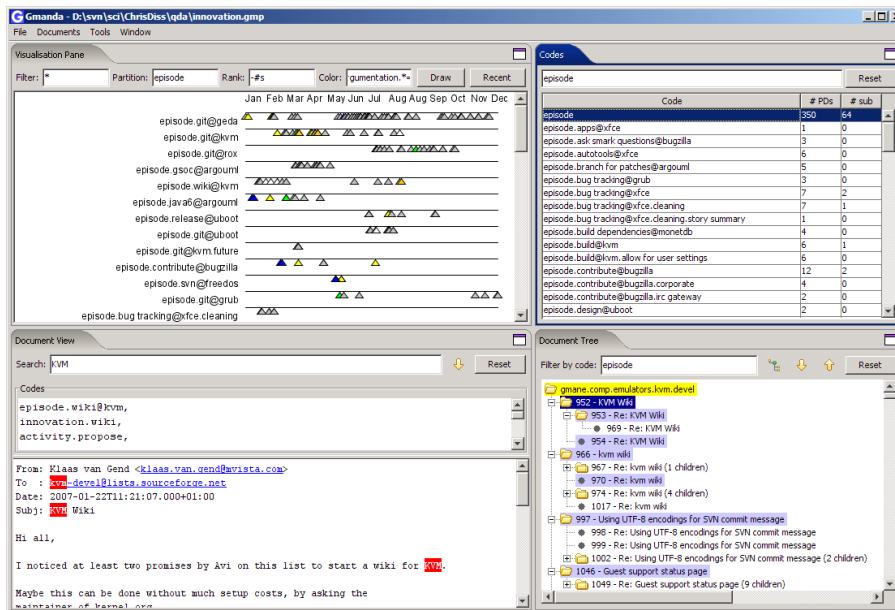


Figure 1: A screenshot of GmanDA showing the four views (clockwise from top left) Visualisation, Codes, Document Tree and Document Detail.

and several new mailing-list added. During this process a coding scheme was decided to be the best way to capture the phenomena found within the e-mails in accessible and searchable way. To this end it was chosen to hierachically describe aspects of topics discussed in the e-mail and for each aspect to to come up with a categorization that encodes this aspect from very abstract down to concrete instance.

Consider the following example from the ROX mailing-list:

I'm thinking of converting the ROX-Filer svn repository to git. Any objections? Also, do you want your e-mail address to appear in the commit messages?

This has been annotated by noting

- that the innovation under discussion is git, a source code versioning tool, thus annotating `innovation.scm.git`,
- that the author is still in the process of making a decision, thus when trying to figure out what the primary activity of this message is, he is proposing something to the project, leading to an annotation with `activity.propose`.

- that he is querying the other project members for objections (rather than opinions for instance), indicating a high level of confidence regarding his own assessment of the advantages of the technology, coding `argumentation.any objections`.
- that the last sentence is not about the question whether to introduce or not, but rather already about the configuration or actual setup of the innovation, coding `innovation.tuning.scm.include e-mail in commit message`.
- that the rationale for the second question is not revealed to us, but that there are two very likely interpretations, a concern for privacy (`forces.privacy`) and a concern for spam (`forces.spam`), that we should keep in mind for further analysis.

There are several important questions that arose while coding in such a way:

1. How much interpretation is allowed? While the first four codes assigned to the above quotation are kept close to the text, there is substantial interpretation in the two forces assigned. The question whether to include these two codes or

to merely keep the innovation tuning code, is a economical, maybe even a fundamental one. Restated, it asks: If we are seeking understanding about the underlying principles that drive occurrences of a phenomenon, instead of trying to just uncover the phenomena themselves, are we allowed to lift the interpretation level or do we have to wait until we find reflections on these phenomena by the participants themselves. This is a question of coding economics, since we see many more quotations of a certain kind than reflections on the nature of this kind.

Consider for this regard somebody trying to understand how merchants on a market strike a deal with each other by bargaining. One will see many such transactions between two merchants in which one is offering a price and the other calling or rejecting it, but little discussion about how each merchant comes up with these prices and why a certain price is called while another is rejected. Can and should the researcher try to interpret an exclamation of "oh no" to mean "oh no. This price is too high" in the context of a high offer being rejected or should she rather wait for the exclamation of "oh no, this is too high a price"?

Or recast as a question about holiday letters [1]: Can we understand why people write holiday letters² or only what kind of content is present in holiday letters using GT?

To further this question consider the following sentence by a participant from [2] "Each day I spread my activities over the morning, resting between shaving and bathing" which has been marked by the conceptual label of "pacing". The more fundamental question now is whether we can reasonably assume that the label pacing would have arisen when looking at a transcript of the activities of the respondent such as shaving and bathing or whether the respondent himself has given us much more suitable data as the base for Grounded Theory analysis than the transcript.

2. How much do we need to abstract? Strauss and Corbin argue [5] that the researcher should avoid to merely describe phenomena, but rather should conceptualize them, since theory could not be

²Given that the letters do not contain explicit reflections of participants on why they write these letters.

```
episode.apps@xfce,
memo="an interesting discussion [...]",
argumentation.closed vs open="the [...]",
rationale.popularity,
rationale.usability;
offtopic.request.documentation;
sensemaking.project goal="..."
```

Figure 2: An example coding demonstrating the analysis of a single message using codes split by semi-colons (separating topics), commas (separating aspects of a topic) and periods (creating a hierarchical system of codes).

built from "raw data" [2]. Considering "pacing" in contrast to "argumentation.any objections", we can see that the latter is much more descriptive. Yet searching for a concept that encompasses both the concrete incident and is abstract is hard: "confirming" for instance does not contain the notion that this confirmation is for something the author does not believe needs further argumentative support, but rather could only be stopped by a compelling reason against.

3. How can we achieve higher levels of insights that go beyond the mere understanding that e-mail messages in Open Source mailing-lists will contain a taxonomy of activities, innovations proposed or argumentations used?

5 Coding syntax I

A primary document is coded by a single string, which is parsed into separated codes by splitting first into semi-colon-separated sets of codes denoting topics being discussed in a message. Then these sets are comma-separated so that each topic is split into aspects encoded by a single code each and finally each code is period-separated into code levels providing on each level finer-grained abstraction of an aspect present in a topic. To provide a concrete example of an abstracting code, a double quoted description can be appended to each code using an equals sign. An example coding can be seen in figure 2.

6 Analysis part II

In the ensuing code session which took place from January 2008 until April 2008, a total of 35000 messages from 14 mailing-lists (from the projects KVM, xfce, Rox, FreeDos, Bochs, Request-Tracker, Flyspray, U-Boot, Grub, Koha, Bugzilla, Geda, ArgoUml and MonetDB) was loaded into GmanDA. I estimate that I have looked at 20% of all messages.

The following main primary code levels were extracted during this second - more structured - analysis:

1. *Episode* Aggregate messages to introduction episodes.
2. *Innovation* Categorize the innovation appearing in the message.
3. *Activity* Categorize the activity the author of the message is performing; for instance the author might *announce* the availability of an innovation or *reject* a proposed innovation.
4. *Argumentation* Categorize *how* an innovation is argued for or against, for instance somebody might use humor or an enumeration of advantages to argue in favor of an innovation.
5. *Capabilities* A categorization of the features and implications of the innovation, for instance on which platform it runs on and whether it needs less bandwidth than the innovation it tries to replace, but also whether the innovation might cause more process overhead or alienate new users. Capabilities are important for the introduction, as people often use them in their argumentations or they are the trigger for new introduction processes.
6. *Persons* Foremost used to categorize the author of a message, these codes can also be used to identify people affected by an innovation.
7. *Forces* A categorization of things that affect an innovation or its introduction. For instance whether you are a native English speaker or not might have implications on whether an innovation is easy or difficult to use; 'daylight savings time' might have implications if you are trying to set a meeting on IRC or the bandwidth a new service uses might cause problems in another innovation running on the same server.

First I will detail how the coding was done, then provide more details about these primary code levels.

6.1 How coding was done

Coding was done using two modes explained below: Scanning and searching. Both modes were done either in a single pass, i.e. identifying potentially interesting e-mails and directly coding their contents, or using two passes, i.e. first identifying all messages containing innovation-relevant content and then as a second pass coding them. The trade-off between one and two passes is that doing two passes requires less concentration but more time.

- Primarily messages were *scanned* by date, i.e. looking at the title of threads and discarding the whole thread without reading any message bodies, if the title was clearly pointing to support enquiries, commit messages or technical issues and the thread did not contain a large number of messages. Threads not discarded were then opened and the message body of the first e-mail was read. If any association to process or tools could be made, the whole thread was read and coded where appropriate. For large threads, answers with many replies were read with preference.
- Once certain innovations had been identified, the mailing-list was then often *searched* for a term closely related to this innovation to extract all message pertaining to the innovation. For instance if a discussion was found in the project to switch the source code management system, all occurrence of the proposed replacement were searched for and then coded.

Messages are often revisited to modify existing codes, for instance if boundaries between codes have changed or additional aspects have become of interest.

6.2 Episode

Because the unit of coding is a message and not the set of all messages associated with a certain introduction, we use the code `episode.<episode name>@<project name>` to aggregate those first. Introduction episodes (and not innovations) are the primary mechanism of aggregation, because the episode

will also contain mentioning of problems with existing innovations, discussions leading up to the idea for an innovation, the retiring of an superseded innovation, problems related to the use of the new innovation, etc.

When to use: This code should be used for all messages except those containing only `offtopic` or `memo` codes.

6.3 Innovation

According to the current working hypothesis an `innovation` is something that intends to change any kind of process in a project. Process for this matter is any generalizable course of action intended to achieve a result.³ This definition has been chosen so that (1) everything that merely changes the product of software development like coding, bug fixing or writing documentation is excluded from observation, (2) it is not essential whether process is actually changed by an innovation, but rather it is sufficient that there was an intention to change the process and (3) a minimal amount of generalizability can be found, so that we exclude one time solutions to one time problems.

Stephan has at several times voiced concerns regarding the loose concept of innovation in this research. To address these concerns which point to the question how we can distinguish innovations from non-innovations, I first want to give five example innovations that have occurred in the analysis II and then describe several important aspects implicated in the definition of innovation for the use in this research.

1. Switching the source code management system from centralized Subversion to decentralized Git. A very popular innovation in 2007 for the projects regarded is to adopt the SCM used by Linus Torvalds to manage the source base of the Linux Kernel. One core advantage of the decentralization is that it becomes unnecessary to manage user accounts on a server to grant write access to new developers.
2. Switch to a release naming scheme based on year and month. Many projects in the analysis have encountered discussion relating to adopting a different naming scheme for their software releases. While no project has changed, many different propositions have been made, among

them one to use the current year and month to tag a release, the suggested advantage being that continuous development is recognized better and decision processes necessary for major/minor schemes are avoided.

3. Use a continuous integration tool. This innovation was popular with larger projects which were targetting several different platforms and consists of running dedicated build-servers which tracks the SCM and then try building the software upon change. While advantages should include greater awareness of platform issues and a more consistent code-base, analysis points to the need to carefully design notification schemes.
4. Joining a non-profit organization for money handling. This is a legal innovation aimed to simplify processes necessary to deal with money received from advertising, merchandising or cooperating with Open Source supporters like IBM or Google.
5. Use of a merge window for changes to the development trunk. A process innovation used in projects where there are many parts of the software that are affected by changes to a central component. Thus the idea is to lock down this central component some time before the release so that dependent components can adapt to changes in the core.

These five innovations all appeared in the mailing-lists analyzed and reached varying stages in the introduction process ranging from discussion, execution to adoption. While these innovations present prominent examples, they are only a small part of the set of innovation that have been seen. I hope that these examples nevertheless make the following discussion of aspects of the definition of innovations more meaningful:

1. Processes, Tools, Legal and Organizational setups. Innovation tries to encompass (1) tool innovations such as using a different build tool, (2) process innovations such as performing peer review before committing to version control, (3) legal innovation such as switching the license used, (4) organizational definitions such as assigning roles to peoples and (5) combinations of these such as the implementation of a unit testing regiment in a project, which entails the use of a unit

³<http://www.thefreedictionary.com/process>

testing tool as well as changes to the expected behavior of project members and their roles in the project. Innovation tries to encompass all these broad meanings of things that affect the way a project works.

2. Minor and Major. Innovations can be small or large, both regarding effort to execute and implications of use. Some innovations seem so small that it is unclear whether they are worth coding independently or rather as belonging to a broader innovation. The heuristic used for grouping several separate interventions as one innovation is that they are all related to the same announcement of availability to the project (see Section 6.4 for a discussion of announcement), i.e. they have been proposed before their associated innovation has been announced.
3. Innovation and Introduction. Can an innovation be regarded without considering its getting established in a project, its implications and the implications of phasing out a replaced innovation? In other words, which of the following constitutes the innovation: (1) switching to a new source code management tool, (2) using a new source code management tool, (3) the source code management tool itself or (4) any of these in the context of an actual project, and should we distinguish each of these in our analysis? This also relates to the insight that an innovation is not something 'new', because 'new' is always relative to a point in time and we want to use the term innovation to talk about the same thing using the same label 'innovation' regardless of the current point in time where we are. If this feels awkward, then one should think of software development supporting entities instead of innovations.
4. Capabilities, Problems, and Goals. There are many aspects of an innovation that become interesting in the context of its introduction into a project. For instance we notice that there are different angles from which new innovations can be approached. The primary mode is technology-centric, putting a technical or process innovation at the centre of the discussion, but sometimes the discussion is driven rather by (1) the goal to attain, and the innovation is rather a side-note to this or (2) a problem that the project is trying to solve, but is still trying to come up with a in-

novation that could be solving it. Certainly these are different facets of looking at the same issue, yet it is not yet clear how to best unify goals, capabilities (as called from the technical viewpoint) and problems or at least link them against each other.

6.4 Activity

An activity (with regards to this analysis) tries to categorize the primary intention of a message with regards to the usage of an innovation or its introduction into a project. Before the innovation is usable by the project, we find the following activities:

- The most typical way an innovation finds its way into a project is by being **proposed** to the project, i.e. a project member asking the project what they think about using a technology or process. If the innovation is ready to be used, we call this an **offer** instead.
- In some instances, the project member does not ask the rest of the project whether there is interest in using a technology or process, but rather just **lets** them **know** that he is currently making an innovation ready for usage.
- Such making ready is coded as **execute**.
- After the innovation has been readied by a project member, it can be **announced** to the project as available. While usage of an innovation might predate its announcement, the announcement is still a significant point in time.
- Once an innovation has been announced, **sustaining** activities like **instructing** people on the usage of the technology or performing upgrades to an installed tool might occur.

6.5 Argumentation

Before a project decides to use an innovation, we will usually see a discussion between project members about this innovation. In this discussion the use of arguments to sway the opinion of the group is central. Argumentation codes try to capture the *how* an innovation is proposed, offered or announced to the group. So far only a large number of sub-codes without further categorization exist. Example codes include the

use of humor in arguing against a proposed innovation or to finish a message with a question to invite discussion.

6.6 Memos

Grounded theory methodology encourages researchers to use memos as “written records of analysis” [5] along the way to formulate theory from coding. The following list is an overview of the most interesting insights gathered using memo codes in this first analysis.

- Partial migrations. As with new features, it seems that there is a tendency to space out the introduction of an innovation in more manageable chunks, especially with version control tools. Some projects have for instance first moved one or two modules to a new version tool before switching larger portions or some projects experimented with using Git as an scm by using it to track their existing system, to then switch to their existing system tracking git. In the Open Source world we can very likely say: refactor, not rewrite is key.
- Abstraction level for process innovations. The occurrence of some one-time process improvement initiatives has prompted the idea that instead of introducing a new process innovation such as a formalized process as a general prescription to the project, one can ask for permission to perform an actual process step from this formalized process once. For instance instead of stating a requirement to clean the bug-tracker after each release, one can ask for permission for the current release, alleviating fears that this might cause overhead for the project in the future. This point also makes sense when considering the difficulty to enforce compliance in Open Source projects.
- Capacity for change is limited. Even though this was mentioned by a maintainer when planning the next release time and mostly referred to code changes, it seems equally true for the amount of innovation that a project can digest. Each change needs time to be discussed, executed, adopted and used, bringing with it side effects and additional overhead.
- Maintainer might is substantial. In most cases maintainers are those who accept and execute innovations, only in a few cases do they lose arguments with the rest of the project, and there are even some cases where they unilaterally decide things against the will of the rest of the project.
- Technology-oriented thinking is central. While there are no numbers to it yet, I can only recall the instance of a single maintainer with explicit goal-oriented thinking. All other innovation instances were clearly driven by problems or technology.
- Innovation terminology needs more precision. This analysis of innovation introduction in Open Source started with terminology derived from Denning and Dunham which proposed seven practices for the innovator. Using this terminology has been difficult, not only because we do not understand the terms fully as detailed in the introduction, but also since in reality they often do not exist as clear-cut distinctions, for instance it is hard to distinguish offering and sustaining an innovation by an innovator and adoption by non-innovators.
- Innovations can be really small. This is down to the point that the importance of the innovation seems negligible, yet discussion is large and lively (which could be interpreted as pointing to a lack of efficiency thinking in OSS projects, which is not unreasonable to assume). For instance the discussion about reply-headers in mailing-lists even dwarfed one about switching SCMs.
- Tool independence is tricky. While several areas exist where tool independence can work, I have seen many examples where tools are too closely intertwined, or making a tool work with a project is too high a barrier for general adoption. For instance one project only provided support for one IDE as they found it a substantial burden, effectively blocking out other IDEs.
- Democracy of access. Wikis and decentralized SCMs have occurred time and again as enabling and prominent examples of innovations in projects in 2007. Making discussions about access unnecessary, has not once caused a problem in all the e-mails looked at.

- Hosting augments introductions. Most tool innovations require some sort of central hosting on project servers and this affects the introduction process in many ways. Not only is access to these servers often difficult, but also questions about maintenance and upgrade, downtimes and responsibilities need to be managed by the project. Often projects need to choose between free hosting and being able to make their own tool choices.

7 Ponderings

The following items have been pondered but not given thorough thought:

7.1 Episodicity

At the moment, episodes are grouped by attaching the same episode codes to all e-mails associated with a certain episode (see Section 6.2). It is then possible to filter the set of all threads affiliated with such codes and thereby have an overview of the e-mails belonging to an episode.

Since it is reasonable to assume that there are many insights about innovations and inventions that we want to associate with whole episodes instead of individual messages, for instance the question whether the outcome of this episode has been a successful introduction or a failure, it is an interesting question whether some means for summarizing episodes becomes necessary. The following ideas have surfaced for this in addition to just providing unstructured summaries using plain text:

- Structured Episode Recaps. Provide a standard format for summarizing episodes, with a set of required fields for innovation, innovation type, introduction outcome, participating project participants, episode duration, number of e-mails, an overview of occurring codes, etc. In essence using such a structured format ensures that all relevant content of an episode is coded and present and that episodes are increasingly comparable.
- Episodes as Shakespearean Plays. The format is semi-structured here using quotations from e-mails or paraphrased quotations summarizing individual posts, but trying to give a sense of the discussion as a sequential narrative with a stage (the mailing-list), acts (sub-episodes) and a cast

(the participants in the discussion) upon which an introduction drama is performed.

Once episodes have been summarized in such a way, an interesting question that arises is whether it should be possible to perform a Grounded Theory analysis upon such summaries. By their nature the summaries try to lift each episode to a higher level, thus making it an interesting source for understanding introduction on a more abstract level. The danger of this approach is that the researcher is categorizing interpretations and no longer actual data, which could lead to distorted results, unless one is making sure that the novel high-level phenomena can be found in the underlying discussion upon inspection.

7.2 Quantitative analysis

The primary focus of the analysis is to gather qualitative insights from the mailing-list under study. Yet, once coding has been done, some quantitative questions about episodes might be possible, if questions about validity are not forgotten. Especially number of e-mails and time between certain events within episodes lend themselves to quantitative analysis. For instance, it might be interesting to compare time from first proposal to announcement for successful introduction between different innovation types, such as tools and processes.

8 Next Steps

Looking into the close future of the next three months there are four different directions that the research could be steered towards. These four directions are not exclusive to each other and often the implicate and further each other, yet they do provide the focal point for which I will aim:

- Hypothesis driven coding. Using the existing memos (Section 6.6), I can formulate explicit hypotheses to be investigated in the data. Such a course is not part of doing Grounded Theory (which aims at building theory, not validating hypothesis), but feels sufficiently natural to be proposed here. The advantage of this approach is, that results are in a scientifically presentable form, given an interesting hypothesis. The disadvantage is, that sufficient data regarding the hypothesis must be found and that our abilities

to abstract from data and understand it, must be on par with the abstraction level at which the hypothesis is formulated.

- Innovation driven coding. In this course of action, a single innovation is put at the center of the analysis. Comparing and contrasting several occurrences of the same innovation in several introductions across projects, especially if both failed and successful introduction can be found, should best increase our understanding regarding the question of how to introduce an innovation into an Open Source projects and might be even practically useful. The disadvantage of this course is, that generalizability might suffer, since only a single innovation will be regarded.
- Project driven coding. Similarly, one could focus also on a single project and compare all innovation introductions that have occurred in this single project. This course is most suitable, if we aim to gain a deep understanding of the innovation introduction behavior.
- Taxonomy driven coding. If we rather want to aim for a broad understanding of all the phenomena related to innovation introduction it might make more sense to continue with Open Coding and focus on establishing clean taxonomies for argumentations, activities, innovations, etc. Doing this will help to answer more comprehensively what innovation introduction actually means.

Independently of the direction chosen, coding quality needs to be improved. First, existing coding of messages needs to be redone to adapt to the evolution in the coding scheme. Second, the codes themselves must be checked, whether new sub-codes should be introduced or whether existing codes should be merged.

References

- [1] Stephen P. Banks, Esther Louie, and Martha Einerson. Constructing personal identities in holiday letters. *Journal of Social and Personal Relationships*, 17(3):299–327, 2000.
- [2] Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, March 1990.
- [3] Luis Quintela García. Die Kontaktaufnahme mit Open Source Software-Projekten. Eine Fallstudie. Bachelor thesis, Freie Universität Berlin, 2006.
- [4] Alexander Roßner. Empirisch-qualitative Exploration verschiedener Kontaktstrategien am Beispiel der Einführung von Informationsmanagement in OSS-Projekten. Bachelor thesis, Freie Universität Berlin, May 2007.
- [5] Anselm L. Strauss and Juliet M. Corbin. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE, 1990.