Software Engineering Research Group
Prof. Dr. Lutz Prechelt

Tracking Innovation in Open Source Software Projects

Master Thesis in Computer Science

written by

Moritz Neeb (TU Berlin)

Matrikelnummer:

359526

supervised by

Prof. Dr. Lutz Prechelt (FU Berlin)
Prof. Dr.-Ing. Sebastian Möller (TU Berlin)

27. Januar 2017

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 27.01.2017

(Moritz Neeb)

Abstract

This thesis is a long term study on the management and communication behavior of Open Source Communities. In particular the strategies and techniques towards changing software engineering processes were investigated. This thesis is based on previous research on this topic, in which one year (2007) of email communication of 13 mid-sized Open Source projects was analyzed. This thesis presents a long term analysis of the same projects, but over the course of the years 2007 to 2016. The theory that was grounded on the data collected in one year was extended and enriched with more data and thus more insights.

A qualitative research approach, inspired by Grounded Theory Methodology was used. It was shown that it is possible to perform a full scale search for the change events that were continuing across the ten year observation period while being started in 2007. Also the theory for such process changes was adapted to fit new insights, such as the influence of an outside event on the motivation of the developer who attempts a change.

Zusammenfassung

Diese Masterarbeit ist eine Langzeitstudie über die Organisation und das Kommunikationsverhalten von Open Source Communities. Im Speziellen wurden die Strategien und Techniken für die Veränderung von softwaretechnischen Prozessen untersucht. Dabei basiert diese Arbeit auf einer vorangegangenen Forschung zu diesem Thema, in der ein Jahr E-Mail-Kommunikation (des Jahres 2007) von 13 mittelgroßen Open Source Projekten analysiert wurden. Diese Masterarbeit ist eine Langzeitanalyse derselben Projekte über den Zeitraum 2006 bis 2016. the vorherige Theorie, die auf den Daten von einem Jahr beruht, wurde erweitert und mit weiteren Daten und Einsichten in die Abläufe von Open Source Projekten und deren Innovationen vervollständigt.

Die Herangehensweise war die der qualitativen Forschung, inspiriert von Grounded Theory Methodology. Es wurde gezeigt, dass es möglich ist eine vollumfängliche Suche nach Veränderungs-Vorgängen, die sich über die Untersuchungsperiode von zehn Jahren erstrecken, nachdem sie in 2007 gestartet wurden. Außerdem wurde die Theorie solcher Veränderungs-Vorgänge angepasst, um den neuen Erkenntnissen, wie zum Beispiel dem Einfluss von äußeren Ereignissen auf die Motivation des Entwicklers, der die Veränderung anstrebt, gerecht zu werden.

Acknowledgements

Thanks to procrastination.
And to everyone who convinced me not giving up.

Contents

1 Introduction

The goal of this thesis is to investigate the long term effect of innovation episodes in Open Source Software projects.

1.1 Innovation in Open Source Software projects

Open Source Software plays an important role not only in computer science but also in the global economy. This statement is valid for more than two decades [6], and even more so as of today: With the DAO[1], a completely new company concept was created solely basing on Open Source Software [8].

Open Source Software (OSS) can be characterized by its distribution model based open source licenses, and the open development process. The development of a specific OS software is typically organized as a project, which is formed by a group of globally distributed participants. [6, 12]

This research focuses on scenarios in which developers are participating in their leisure time in small projects. Apart from large, established projects, most projects lack strictly defined software development processes and practices [10]. Despite this lack, there are OSS projects that have been running successful for more than two decades[2]. Thus the objective of this thesis is to help finding out how, by describing the decision and innovation processes for a selection of projects.

This thesis is conducting follow-up research on the dissertation on the topic of “Innovation in Open Source Software” by Christopher Özbek [9].

For the year 2007 Özbek collected and analyzed different innovation episodes in the mailing lists of 13 OSS projects. Based on this data set, he created a theory that describes the factors that influence the outcome of innovation episodes. For example, an actor may propose to switch the system used for reporting bugs. This proposal starts an innovation episode. However, the episode may result in the introduction of a proposed replacement, but it may also end with no change of the bug tracking process of the project. The research question is whether or not one can identify certain factors that influence the outcome of the innovation episode. Özbek was able to come up with a theory that is based on eight core concepts. These concepts indicate how innovation introduction processes work and which strategies can be used by the innovator to achieve his innovation goal.

[1]Decentralized Autonomous Organization
[2]cf. Section 2.3 and Appendix C

1.2 Motivation

Open Source Software is getting more and more important for businesses and industry [11], but still there is no full understanding and especially no valid theoretical model that explains why and how OSS development works [4]. This constitutes an adoption barrier as many companies are afraid to change to software or tools, having no guarantee on or knowledge of the success of the project. They are not willing to back the continuation of the projects and thus stick with commercial software, in many cases an extremely costly decision. Thus, increasing the understanding of the mechanisms of OSS projects will support the adoption of OSS, ultimately strengthening especially small and medium-size enterprises by freeing up capital for other investments.

1.3 Research Question & Goal of this thesis

This thesis investigates the following questions:

1. Are there any episodes already identified by Özbek that have continued after 2007, maybe even until today? If so, how have these continued?
2. Are the eight concepts identified by Özbek still valid for such long-term innovation episodes?
3. Given that only a fraction of the mailing list data can be read, are they a valid and valuable source of information for the purpose of studying innovation episodes in this timescale?

This thesis is the first to revisit Özbek's theory, with the goal of further publishing relevant results. Moreover his observations are set into the frame of a large scale study with email data from almost ten years (2007-2016), thus providing a significant further evaluation of the identified concepts.

1.4 Outline

Following this introduction, chapter 2 explains the theoretical and practical background. The collection of data about innovation changes or discussions about them, can be found in chapter 3. The analysis of this data then takes place in chapter 4. The thesis is concluded by chapter 5.

2 Background

This chapter provides some relevant background information. First in section 2.1 the definition of Open Source Software, more precisely their defining aspects are discussed. Afterwards, in section 2.2 the work and theory of Özbek is presented, followed by an overview of the OSS projects under investigation in section 2.3. The methodological background, mainly Grounded Theory, is discussed in section 2.4. Finally, in section 2.5 the tools and data sources employed in this thesis are introduced.

2.1 Open Source Software

As mentioned in the Introduction, OSS is software distributed below an open source license, produced with an open development process by a group of developers, who are globally distributed working together. [6, 12]

Open Source Licensing enforces everyone who distributes software below such a license, to make sure that the source code is distributed together with this software. Today this is often done in such a way, that the both the software and its source code are available through the internet, for example on project websites or through central hosting services such as GitHub[1].

The communication of OSS projects is typically mainly done via mailing lists that interested participants have to subscribe to. [7] There are other forms of communication possible, for example real time communication through chats, or more specialized communication via the processes described below.

Processes that are relevant as parts of the software development process:

- Source Code Management Systems are used to organize the work between the developers. Its purpose it to track changes and compare different versions.
- Release Processes are relevant for distributing the software. Certain snapshots of the software are tagged with a version number and then distributed. There are often different types of releases, such as stable releases, unstable releases or development versions, differing in their stability and their set of features.
- Bug Tracking processes are used to organize and collect the problems and defects of a software. For developers fixing bugs is one of the main activities and important for ensuring the product quality.
- Documentation is another form of communicating information with both users, developers and those that want to help out in the project. Often done in the form of wikis.

[1] `https://github.com`

2.2 Innovation Introduction

In his PhD thesis “Introducing Innovations into Open Source Projects” of 2010 Christopher Özbek created a theory about influencing factors of innovation introduction [9]. A short overview of his work will be given in the following sections.

Özbek’s main research question was, how software engineering processes are changed in OSS projects. This is, according to him, done by innovation episodes (2.2.1) which can be influenced by several concepts (2.2.3) on their way to success (2.2.2).

2.2.1 Innovation Episodes

To fully understand the term *innovation episode*, the overarching term of an *innovation* has to be defined first.

Defining what an innovation is, though, is controversial, cf. Özbek [9, section 2.1, pp. 23]. Özbek defines innovation in two ways. The first definition is

> An innovation is a means for changing the development process of a software development project. [9, p. 23]

Later on he provides the following disambiguation:

> One common alternative interpretation of innovation is to associate the term with the delta of change. For instance when switching from technology A to technology B, the distance between B and A according to some dimension could be seen as the innovation [[2][as cited by 9]]. In this thesis though, A and B are innovations. [9, p. 233]

In this thesis the term is used in both ways accordingly.

With this we define innovation episode anything that enables the switch between two innovations. An episode can be started with an innovation introduction email, for example by someone proposing to change the development process in a certain way.

2.2.2 Innovation Lifecycle

The Innovation Lifecycle is one major outcome of Özbeks research, cf. Figure 2.1. It pictures the fact that several barriers need to be overcome to end with a successful innovation introduction:

Discussion Barrier To overcome this a decision has to be taken by the project. To take a decision, oftentimes a discussion has to take place. Sometimes it works without a discussion and the innovator acts on its own or is motivated to act by other project members to go on without talking about the implications.

Execution Barrier For some innovations, work has to be invested to realize them. Even if in the previous step a decision was made, it can be that the person that is responsible for executing it does not find time to do so or was not involved in the discussion in the first place.

Usage Barrier For some innovations it can happen that no one uses them because it is too cumbersome or takes too much time to do so and people do not see a point to do so.

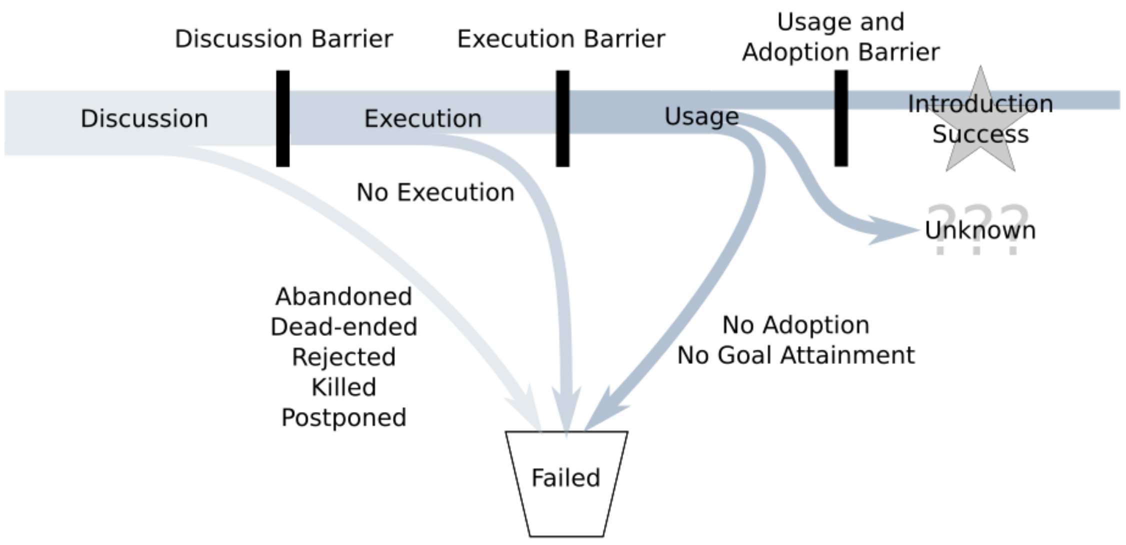


Figure 2.1: Innovation Lifecycle according to Özbek [9]

2.2.3 Concepts

The outcome of innovations depends on several factors, e.g. how the innovator proposes his innovation. These factors were pointed out and grouped into the *concepts*, which then can be used to formulate strategies for a successful innovation introduction.

For an overview of the concepts defined by Özbek, see Figure 2.2.

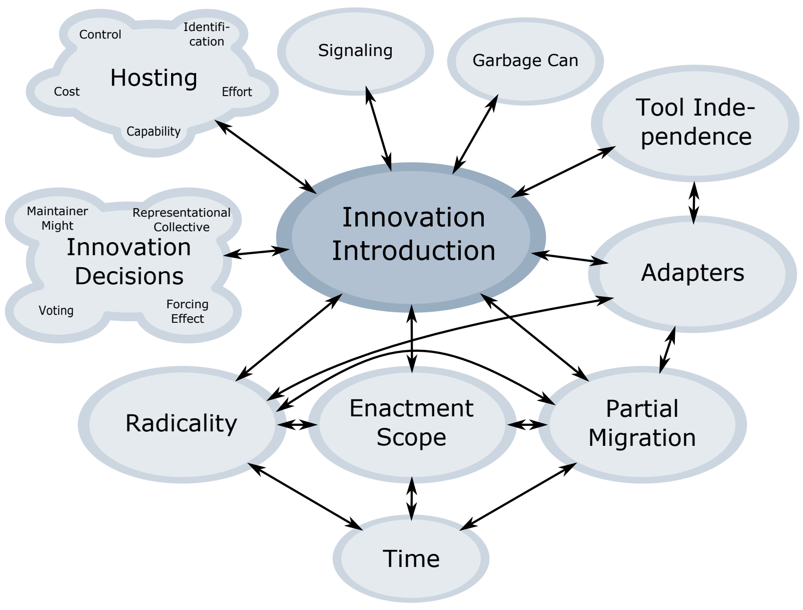


Figure 2.2: Overview of concepts by Özbek [9]

In the following subsections (2.2.3.1 – 2.2.3.8) all concept will be explained shortly. Definitions and quotes are taken from Özbek [9], if not stated otherwise.

2.2.3.1 Partial Migration

Migration is defined as

> A subtype of the introduction of an innovation, where an existing innovation is replaced by a newly introduced one. [9, p. 102]

If the migration happens in steps, for example "moving certain parts of the code base to the novel system while retaining others in the existing one" [9, p. 102], then this is called *partial migration*.

In 2007 partial migration is observed in two innovation episodes, where projects with a more complex code base decide to switch from Subversion to Git. A motivation for this is given by the fact that

> Migrating only partially can be a suitable effort management strategy and give the project members time to experiment and learn about a novel innovation without loosing all productivity based on the existing solution. [9, p. 107]

Nevertheless, a problem of partial migration can be the effort of maintaining both innovations at the same time.

2.2.3.2 Enactment Scope

Enactment can be defined as "[t]he execution of activities mandated by a process by one or several agents" [[5][as cited by 9, p. 108]]

Based on this, Enactment Scope is defined as "[t]he set of situations in which a process should be enacted" [9, p. 108].

Also Özbek relates enactment scope with partial migration: The latter is reducing the scope when introducing a tool or service, the former is about the scope of an enactment (e.g. a process).

2.2.3.3 Hosting

Hosting is defined as "[p]rovision of computing resources such as bandwidth, storage space, and processing capacity to the use by an innovation" [9, p. 111].

Özbek observed various types of hosting, for example private PCs, Forges or Federated hosting. He then identifies five relevant concepts to distinguish them:

- Effort, which is the "amount of time, mental concentration, or physical energy necessary to do something" [9, p. 113], is relevant to hosting because tasks as installation, migration, configuration etc. are things somebody has to spend effort on.
- Cost: Hosting often costs money. OSS projects often do not have a budget for that, so "hosting which is 'free as beer', appears to be widely preferred" [9, p. 113].
- Capability: Can either be the supported features of a host, like a specific service, or properties of a service, such as bandwidth, storage space or quality of service.
- Control is defined as "[t]he degree to which the capabilities of an innovation and host are actually available to the project." [9, p. 114].

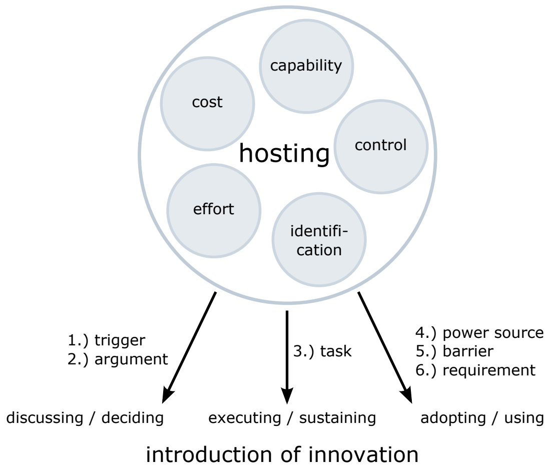


Figure 2.3: Relation between hosting and innovation introduction

- Identification: In general Özbek defined identification as "[t]he degree to which innovations [...] are in accordance with the norms and identity of the particular project and general Open Source community" [9, p. 115]. In the case of hosting this might be a proprietary hoster or hosting on private servers.

For innovation introduction hosting can be relevant in the discussion and decision process, the execution and the adoption and usage, cf. Figure 2.3.

2.2.3.4 Adapter

Adapter Innovation is defined as: "An innovation used to make an existing innovation accessible by another one innovation." [9, p. 118]

The Adapter strategy proposes three ways of making use of them during innovation introduction:

> Use an adapter to (1) bypass discussion, execution, and adoption barriers, (2) run a trial to demonstrating the usefulness of an innovation, and (3) create a homogeneous innovation landscapes for users. [9, p. 119]

2.2.3.5 Innovation Decisions

There are several definitions in this chapter, all around the question how decision are taken. First it has to be differentiated between to levels of where the decision takes place. The organizational innovation decision is a "decision of the project as a whole", for example to adopt or modify an innovation. The individual innovation decision is "the question of whether a project member does actually adopt and use and innovation" [9, p. 121] and as such typically takes place after the innovation introduction is executed.

Several forms of the organizational decision were observed by Özbek:

- Maintainer might: "Often, the maintainer's might is so strong that the maintainer never has second thoughts about making a decision in such a unilateral way." [9, p. 120]

- Representational Collective: “the subset of the project participants who are involved with the discussion assume the role of representing the project as a whole” [9, p. 120].
- Voting
- Skipped Innovation Decision, also called “just do it”-innovation decisions, can happen “when the innovator was able to execute the innovation to a large degree independently” [9, p. 121].

Influencing factors for the individual innovation decision are:

- Forcing Effects: Something that makes adoption easier/faster than normally expected. “A property or mechanism of an innovation that promotes the use of the innovation itself.” [9, p. 122]
- Compliance Enforcement: “The act of ensuring that a given norm or standard is complied with.” [9, p. 123]

2.2.3.6 Time (Participation Sprints, Time-dependent behavior)

Participation Sprints are a “period of high activity by a project member” [9, p. 126].

The first insight that Özbek presents is that “neither intense participation sprints nor prolonged absence appear to be very compatible with the asynchronous rhythm of participation and e-mail communication [[14][as cited by 9]] in Open Source” [9, p. 126]. Overloading could lead to an exceeded capacity of the project to handle all the requests. Absence could “cause an innovation introduction to be abandoned” [9, p. 126].

2.2.3.7 Radicality

A Radical Innovation is defined as “[a]n innovation which has a sudden, widespread, and effort-inducing impact on the way the project collaborates” [9, p. 132].

Özbek explains how different ideas when discussing about bug tracking in git are varying in radicality. He lists arguments, for example one that connects radicality to effort (maintaining a server). Also he says, that it is not clear, “why deviations in certain dimension are perceived as more radical than others” [9, p. 133]. Which means, there is a perceived radicality visible, but not the reason behind it.

2.2.3.8 Tool Independence

Özbek observed only a small number of episodes dealing with tool support. He explains, that this is probably, because Tool Independence is a value for Open Source projects: Every developer should be free to choose the tool they want to use.

2.3 Analyzed Open Source Projects

The data used in this thesis comes from 13 projects. Mainly their mailing list data is used to observe their communication. Often other information about the projects is considered, dependent on the innovation that is investigated, for example source code history or the project's documentation.

The projects were selected by Özbek for his research because of their diverse range of domains, to make sure the view is not biased towards a certain domain. Also a selection criteria was to be medium sized, which meant having around five active developers and more than 500 mails per month on the mailing list [9].

ArgoUML (`http://argouml.tigris.org`) is a UML CASE[2] tool written in Java. It supports the UML 1.4 standard[3]. Its latest release happened in December 2011 (version 0.34)[4].

ArgoUML can be considered as abandoned. The latest stable release was in 2011 (0.34), the latest development release was in 2014 (0.35.1). It still only supports the UML 1.4 standard, which was already released in 2001[5].

Bochs (`http://bochs.sourceforge.net/`) is an emulator and debugger for x86 and AMD64 hardware, mainly written in C++. It supports most of operating systems as a guest system and can be run on most popular platforms.

Already in 2007 it was considered as a mature project. Version 1.1 was announced in 2001[6], the latest version 2.6.8 was released on May 3, 2015.

Bugzilla (`http://bugzilla.org/`) is a web-based bug-tracking software written in Perl. It was released Open Source by Netscape in 1998 as part of the Mozilla Project.[7] It is still actively developed by today.

Flyspray (`http://www.flyspray.org/`) is a web-based bug tracking system written in PHP. It was originally developed for the Psi Jabber Client and was released Open Source in 2003. After a stalling period, the Veloz Group took over Flyspray in 2012 is sponsor and project leader.

FreeDOS (`http://www.freedos.org/`) is a Microsoft-DOS-compatible operating system written in C.

[2]Unified Modeling Language Computer-aided software engineering
[3]`http://argouml.tigris.org/features.html`
[4]`http://argouml.tigris.org/servlets/NewsItemView?newsItemID=2497`
[5]`http://www.omg.org/spec/UML/1.4/`
[6]`http://svn.code.sf.net/p/bochs/code/tags/REL_2_6_8_FINAL/bochs/CHANGES`
[7]`https://www.bugzilla.org/status/roadmap.html#history`

gEDA (`http://www.geda-project.org/`) is a CAD tool used for circuit board design. Its latest stable release is from 2013, the latest unstable release happened in 2015, but there's still activity in the repositories.

The developer mailing list was abandoned in 2013. Since then everything is handled via a unified developer and user mailing list.

GRUB (`http://savannah.gnu.org/projects/grub/`) is a popular boot-loader for the x86 Architecture. It is an official GNU project.

KVM is a virtualization software strongly tied to the Linux kernel.

MonetDB (`https://www.monetdb.org`) is a Database Management System.

RequestTracker (`https://bestpractical.com/request-tracker`) is a ticketing software.

ROX (`http://rox.sourceforge.net/desktop/`) is a window manager for UNIX-like operating systems.

U-Boot (`http://www.denx.de/wiki/U-Boot`) is another boot-loader, often used for custom circuit boards.

Xfce (`http://xfce.org`) is a desktop environment for UNIX-like operating systems. It is a minor active project although the latest release did happen in 2015. There are recent mails and recent commits.

2.3.1 Project Aliveness

To make sure the criteria are still roughly met, an email traffic analysis took place in the beginning of the research. An overview of the email traffic for each of the project's mailing lists can be found in Appendix D.

Given the tremendous changes in communication methods and the flood of new tools available, it can not be taken for granted that mailing lists still constitutes the main communication channel of the projects, see also Guzzi et al. [7]. However, as long as there is no evidence to the contrary[8], it will be assumed that the projects are still using mailing lists as their most important communication medium.

[8]e.g. by declining number of emails while development continues at the same rate

2.4 Qualitative Research

2.4.1 Grounded Theory Methodology

The research by Özbek follows the Grounded Theory Methodology (GTM) by Corbin/Strauss [13], which is a method for doing Qualitative Research in a field that has not yet been thoroughly described.

To understand the research method in this thesis, it is beneficial to have an idea about GTM, thus it will be briefly described in the following. If the reader is not familiar with this method, it is recommended to either read the more detailed explanation of Özbek's approach by himself or consult an introductory book by the creators of GTM [13].

To become aware of the theoretical sensitivity[9], it is important to note that the following description summarizes the *understanding by the author* of this methodology, which is mainly influenced by the work of Özbek and may deviate from the view of GTM by other researchers.

The goal of GTM is to create a theory that describes phenomena that are visible in the data. It then explains the processes and relationships of phenomena in the underlying data and a theory is postulated. This theory is created by three ways of approaching the data:

- Open Coding: In the first step a promising data point[10] is selected[11] and coded with keywords that are meant to describe simple phenomena which appear to be relevant for the research question. For example (a part of) an email can be coded with `proposal` if an idea is proposed.
- In Axial Coding then a specific code is chosen[12] and all data points that are relevant to this code are considered. The goal is to compare the different observations of one phenomenon to point out and extract properties that are relevant to this phenomenon, which then might lead to a concept. An example from Özbeks research would be the creation of the concept of an `innovation episode` with, for example, the parameters `innovation type` and `innovation outcome`.
- Selective Coding then is the act of creating a concept on a higher level to set the concepts into relation. Here the researcher considers the relation between concepts to note their influencing factors. This happens with the goal of answering the main research question, in Özbek this would be for example, how the outcome of an innovation episode is influenced by a decision process (cf. Section 2.2.3.5).

2.4.2 Empirical Research Process

The following three chapters are to be understood as one process in which the researcher switches between three perspectives iteratively. This is inspired by Grounded Theory Methodology as explained above, but as a theory already exists (created by Özbek) this

[9]Which is the fact that the view of a researcher is subjective, is acceptable as long as this fact is not hidden intentionally, rather pointed out where possible.

[10]here: emails.

[11]The data point is selected by using a technique called *Theoretical Sampling*, also explained in Corbin/Strauss [13].

[12]The Theoretical Sampling approach can be used here again.

is not a GTM from the ground up. The data available to the researcher is both the coded data by Özbek and the emails over the observation period of ten years (cf. Section 2.5.3).

Still there is some relation to GTM, mainly in the way of abstraction throughout the three steps:

- The first chapter is about finding and selecting data towards single innovation episodes that happened in 2007. This can be understood as some kind of Open Coding restricted to evidence for the continuation of the innovations. For example taking a successful innovation introduction episode, the research will try to find emails that show whether this innovation is still successful in 2016 or not.
- In the next chapter the found continuations will be compared, with focus to their respective outcome, which can be viewed as a subset of Axial Coding.
- The last step is to compare the new data to the theory created by Özbek, to see if the concepts that influenced the 2007 outcome are still relevant for influencing the course of innovations over the observation period.

2.5 Tool Support and Data Sources

2.5.1 Gmane

To collect and download the mailing list data for the observation period, gmane was used. Gmane[13] is a mailing list archive service. It was created in 2002 by Lars Magne Ingebritsen, who also ran the service until mid 2016.

After maintaining it for fourteen years Ingebritsen decided to take Gmane offline in July 2016 after is was facing several DDoS attacks[14]. A company then acquired gmane and re-enabled it[15]. However, by January 2017, gmane unfortunately does not provide the data used in this research anymore.

Despite these problems the gmane archive was still used as a source of data, the download of the analyzed mails happened before the availability problems. Until it went offline the first time, also the excellent browsing and searching features were used to support the search.

2.5.2 GmanDA

GmanDA, the Gmane Data Analyzer[16], is a tool written by Özbek to conduct his research [9]. It can be described as an email viewer for qualitative data analysis. It supports the coding process by the following features:

- Annotating emails with codes. The annotation takes place in a free form text field, but is expected to follow the coding syntax that Özbek created.
- Searching emails by keywords or codes.

[13]`http://gmane.org`
[14]`https://lars.ingebrigtsen.no/2016/07/28/the-end-of-gmane/`
[15]`https://lars.ingebrigtsen.no/2016/09/06/gmane-alive/`
[16]`http://www.inf.fu-berlin.de/w/SE/GmanDA`

- Searching the list of codes.
- Visualizing some features of the annotated emails, such as their distribution over time
- Viewing two codes in a table, to compare their set of emails.

Although it was already six years old and not further developed since then, it was not a big issue to get it running, mostly thanks to the intermediate layer provided by the Java Runtime Environment, of which the version 1.6 from back then was used.

2.5.2.1 Email Identifier

When refering to example emails it is valuable to have a system of unique identifiers. In GmanDA and in Özbeks dissertation the gmane ID was used, which is comprised of a mailing list identifier and a positive integer, for example `gmane.comp.desktop.xfce.devel.version4/13143`.

An identification scheme is of course only of any use, if the data it identifies can be accessed, which is not the case for gmane. In this thesis the identification scheme is continued to be used, because the problems with the data source only came up during the research period. To ensure that the reader is able to access the referenced emails, the author provides an alternative access at `https://moritzneeb.de/innovation-thesis/gmane/`. A gmane ID can be accessed through a URL of the format `https://moritzneeb.de/innovation-thesis/gmane/<project-name>/<numerical-id>.html`, for example `https://moritzneeb.de/innovation-thesis/gmane/xfce/13143.html`[17]. In the PDF version, the links of the format [xfce:13143] are leading to the corresponding URL.

2.5.3 Data Sources

The data used in this thesis is from several sources:

- Gmane, as explained above. The central source for the emails for this thesis.
- The coding produced by Özbek. During his research process he attached codes to ca. 1500 emails spread throughout the 13 projects and the observation period, the year 2007. This data is the grounding that led to the theory which Özbek describes in his work. In this thesis his data was available. It was imported to GmanDA and used as a basis for the research.
- The project websites and services. This is the most diverse source and often was only accessed through the browser. An exception are the code repositories, which all were collected by the author to enable browsing the history and generating statistics about the development via scripts.

[17]If the gmane service is fully restored again at one point, the email should be accessible by `http://article.gmane.org/gmane.comp.desktop.xfce.devel.version4/13143`

3 Tracking innovation changes and discussions

As already described in the previous chapters, the most of the OSS projects observed by Özbek are still active and it is assumed[1] the mailing list data can still be used as the major source of information (2.3). In this chapter, the analysis focuses on Özbeks so called innovation episodes (as described in section 2.2), moreover how to identify related emails in ten years of mailing list data.

The relevant question is, if and in what form innovation episodes are continued beyond the year 2007 as observed by Özbek. To answer this question it is necessary to identify emails that potentially contain evidence toward the continuation of an episode. Given the large amount of emails and the abstract form of the concepts defined by Özbek, one sees that this question touches two aspects:

On the one hand the aspect of efficiency. If evidence for continuation exists, is can be identified – by a human – by reading all of the emails carefully. This naive approach is not suitable, as it is too time consuming to read all of the emails one by one[2]. It is thus necessary to reduce the amount of emails to read to a minimum. It has to be noted that an approach that sets a focus to certain emails has the disadvantage that the negative predicate (i.e. there is no continuation) can not be stated with certainty. Only positive examples can be used for confident statements about a continuation.

On the other hand the nature of Özbeks concepts is not used on the same level as everyday language use. The concepts defined by Özbek can not automatically be identified in a straightforward way in the large email dataset. For example for the question how an "innovation decision" is present in an event, a question that is included is whether the decision was taken by a "Representational Collective", through executing "Maintainer Might" or, in the case of an individual decision, through "Forcing Effects". For most of them, some higher level knowledge is necessary, e.g. to know who the maintainer is and to understand that an episode failed, because this maintainer objected. This knowledge is hard to represent in a way that search could be automated, thus it is necessary to work through intermediate steps to reduce the data has has to be analyzed in detail.

3.1 The Search Process

Search Objective Before approaching the dataset, a search objective has to be defined. This is helpful to know where to continue the search process and whether enough was already found for a particular innovation episode. The search was done on the level of a particular episode that happened in a certain project. The goal was to *identify emails*

[1] even after conducting this research

[2] There are around 500.000 emails. Assuming it takes 10 seconds to decide if an email is related to an innovation episode or not, processing all emails takes around 6 month when working 8 hours a day.

that are related to this innovation episode, such that this data can be used in a later step to decide whether the innovation that was happening or attempted in 2007 can still be considered successful or failed, respectively.

3.1.1 Guiding Questions and Sources To Find Answers

It was often necessary to first read the emails and memos that were coded by Özbek for a certain innovation episode, to understand what happened in 2007. Dependent on the situation the search can start by examining the email archives or rather searching for evidence by looking at the project's infrastructure or documentation.

The process was guided by specific questions and actions were taken dependent on the situation.

An innovation (e.g. a process or a document) was successfully introduced in 2007, then evidence has to be identified whether ...

... this innovation is still in use? This could often be answered by first checking the resource that was introduced. For example whether a document or a service introduced is still existent by considering the project website.

... problems occurred related to the innovation? To see this, relevant keywords have to be identified and the email archive has to be searched for complaints or discussions. See below in this chapter how this was done.

... the innovation still be considered a success? This is often a question difficult to answer by viewing from outside and is up for interpretation. The reason for this is, that it is not common for OSS project to have a review of the situation in which someone summarizes long term developments.

If an innovation episode failed in 2007, then two things can happen: Either the problem that motivated the innovation attempt persisted or it was not considered as relevant anymore. In the second case silence is expected. In the first case, it is expected to see evidence for the problems and then it can be searched more systematically for reoccurring innovation attempts.

For episodes with unknown outcome in 2007 it might be the case that, by having a larger observation period, evidence can be found to make things more clear.

If the evidence was an email, then it was flagged to belong to the respective innovation. If it was something else then the link can be found in the results.

3.1.2 Keyword Filtering

Filtering all emails of a project by keyword turned out to be a surprisingly efficient technique in many cases. In this thesis most of the time a very basic approach was used:

Definition (Keyword Filtering) For a set of keywords (the *query*), return every email that contains all of the given keywords.

The reason why this basic approach was used is because it worked well in most of the cases, as will be reported below. Also it was already supported by GmanDA, which uses an Apache Lucene[3] backend, thus could be used off the shelve.

Alternative: Elasticsearch During the course of this research project, other approaches were attempted, for example establishing an Elasticsearch database[4] that would be aware of metadata such as author information, thread structure and the capability of more sophisticated queries. It turned out that the configuration and indexing process was more complex than expected, so after investing some time it was not considered worth the effort.

Search Quality A query should ideally not return more than 50 results. If the number of results is larger, then at least the number of threads should be low.

The quality of a query can then be judged by its Signal-to-noise ratio. This was judged subjectively on a heuristic basis by examining the threads and asking questions:

- Are there threads where the title implies a relation to innovation? This is not a necessary condition, but definitely sufficient, in other words, there are also innovation discussions in threads that have another topic (e.g. a technical discussion), but if the subject is implying an innovation discussion then it is definitely worth reading the email.
- Is the thread related to the innovation? If not, why not? Can the query be adapted to refine the search?
- Were there false positives? For example homonyms or the use of the keyword in another context. Can the query be adapted to refine the search?

If the noise was too high, the query was abandoned or adapted. If there were some hits, then it was considered valuable to visit all of the emails. If there were a larger number of emails in the result, then still some emails were visited to learn how the query can be improved.

3.1.3 Tracking Innovator Activity

For some innovation episodes it was helpful to track down the activity of a certain project participant, perhaps in combination with keyword filtering. Reasons for this are:

- Following opinion leaders: Some people have a strong opinion on a certain topic and are elaborate on it. Especially email threads where a beginner was confused about an innovation and an experienced person explains the reasons behind decisions are very helpful to judge the status of an innovation in a project.
- It could be that an innovator retries their attempt if it failed before. Or shows some other activity of solving a problem.

[3]`http://lucene.apache.org/`
[4]`https://www.elastic.co/products/elasticsearch`

outcome	#episodes
failed	33
success	22
unknown	10
umbrella	2

Table 3.1: Episodes per outcome. The umbrella episodes do have ambiguous outcome denoted

- Project members come and go. It is helpful to understand what the status of an innovator is in the community. On the other hand it can also be relevant if a project member who opposed an innovation steps down and leaves the project, which changes the situation for an innovation.

3.1.4 Strategic threads and finding new keywords – Keeping eyes open

Sometimes there are threads that are strategic, such as “Where should Bugzilla go”. These threads were observed to attract innovation discussions. So if in a query such a thread shows up, it definitely should be investigated.

It is also necessary and helpful to be sensitive how different project have different terms for their processes. By reading some emails in a random manner, starting with those that were collected by Özbek, things might be learned about the manners and terms of a project, such that the query can be adapted to better fit it.

3.2 Search Results

The results presented in this chapter are about some general insights when searching for evidence related to certain innovation episodes. The detailed findings for some selected innovation episodes of 2007 are presented in Appendix A.

3.2.1 Data Selection

In the raw data of Özbek [9] from the year 2007 there are 172 different codes for episodes. These codes can be grouped into 111 episodes with 61 sub-episodes. The 111 episode do include things that are only minor related to innovation introduction. As this is the raw data of Özbek, these can be historical artifacts, things that were first considered to be related to innovation material and now are not anymore. When dropping the episodes labeled as offtopic and the episodes that do not have an outcome themselves or in their sub-episodes, then 67 episodes are left. For those 67 episodes the results of the search for a follow-up will be presented. An overview by outcome can be found in Table 3.1.

3.2.2 Summarizing Results

Of the 67 episodes, for 42 episodes a detailed investigation of follow-ups happened during this research. For most of them, either evidence could be found on their progression over time or it could be excluded with high certainty that a continuation took place in the observation period. Descriptions of the relevant follow-ups that were detected can be found in Appendix A.

Of the 15 that were not investigated in detail most of them were dropped because it was not expected to learn something new in comparison to the effort that would be necessary to find a continuation. This means, it was difficult for some episodes to find the right keywords to have a manageable result, which leads to the first result presented in the next section.

Signal-to-noise Ratio depends a lot on the innovation In 2007 there were some license changes, for example gEDA had a license issue related to the symbols that were used in the software. This could be resolved by picking a special font-related license for the symbols.

It is difficult to investigate license issued, because most projects also post patches on their mailing list where the word “license” typically appears somewhere in the header. This leads to too many hits while searching, in other words the density of innovation-related emails was low. On the other hand it was not expected that this played a role over the observation period, because licensing is not very relevant for the development of the product, because it is a higher level, political, issue and most developer are not interested in the legal details [9].

Following up successful episodes is more difficult and less worthwhile A similar issue came up when searching for possible problems of successful episodes. For example did three project successfully introduced git in 2007. Searching haphazardly for problems is a difficult task with keywords as generic as “git” and “problem”. Again no big results were expected, because all three project are still using git as their main SCM system, so the investigation was stopped.

Another innovation type where this happened is version naming. If there was no evidence from outside, that the version naming changed during the observation period, then no investigation took place whether a successful version numbering system introduced in 2007 was causing any problems. For an overview of the versions of the investigated project, see Appendix C.

Summary Apart from the two problems described above, finding certain continuations is a feasible task for the given amount of emails. Of course, given the nature of a concept, some types of innovation episodes are not discussed on the mailing list, then the research either has to rely on other data or on other methods, such as approaching the project participants.

3.2.3 Collecting New Episodes of Common Innovations

While researching the continuation of the episodes of 2007, also information about repeating episodes in the cases of source control management systems and the Google Summer of Code program was collected. As this gives more insights about the development of the projects under observation, and general trends across the dataset, the results are presented in this thesis.

3.2.3.1 Source Control Management Systems

There were seven innovation introduction attempts in the year 2007 related to the topic of source control management (SCM) systems. This was already a lot for only one year, but more has happened since then. It was possible to track down the changes of SCM systems in all of the cases[5].

For an overview of the history of switches consider Figure 3.1. Of the 13 projects in the observation period, there are only two that did not switch the SCM. One of them was abandoned[6] and one already used the SCM that turns out to become the most popular [7]. The shortest time an innovation survived (where start and end are in the observation period) is one year[8].

There were fourteen SCM switches in total in the full observation period. Eleven in the first half, three in the second half of the observation period. Most switches were to git from some other system and also most switches were from CSV to some other system, see Table 3.2 and Figure 3.2 for an overview. The email evidence can be found in Appendix B.1.

The observation that git becomes more popular is in line with current research about git [3] and GitHub [1].

[5]The cases were extracted by tracing back the introduction of the available SCM systems towards 2007.
[6]ArgoUML uses SVN but is not further developed since 2012
[7]U-Boot which uses git since 2006
[8]Subversion in GRUB

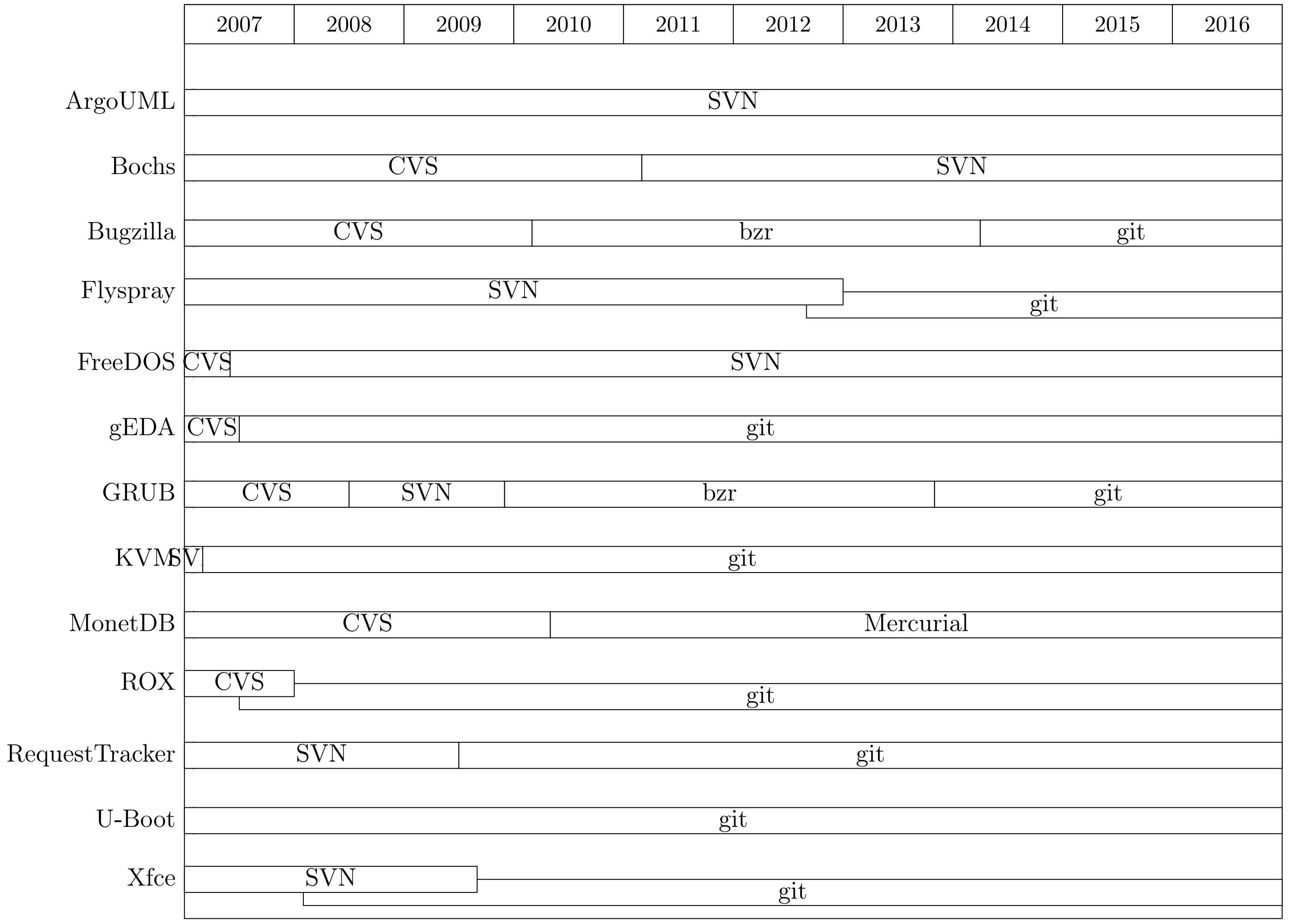


Figure 3.1: History of Source Control Management Systems

from \ to	CVS	SVN	bzr	hg	git	$\sum$
CVS	-	3	1	1	3	8
SVN	0	-	1	0	3	4
bzr	0	0	-	0	2	2
hg	0	0	0	-	0	0
git	0	0	0	0	-	0
$\sum$	0	3	2	1	8	14

Table 3.2: incoming and outgoing SCM switches

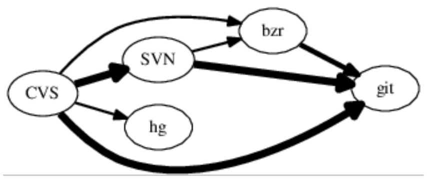


Figure 3.2: SCM evolution schematically

3.2.3.2 Google Summer of Code participations

The research about the dynamics and effects of Google Summer of Code (GSoC) could be a research project on its own[9] and thus only the first step, the data collection took place in the scope of this thesis.

From the perspective of finding evidence for discussions about this topic in mailing lists, GSoC is a keyword with up to 100% success rate[10], depending on what was the goal to find.

The first observation is that there are many projects that never considered participating, namely Flyspray, MonetDB, ROX, FreeDOS and Bochs[11]. Many projects that considered participating did this because a relevant umbrella had good chances, although this does not mean that the project succeeds.

Relevant umbrella organizations for the observed projects, participating in GSoC are:

- Mozilla (Bugzilla): 2007-2016
- GNU Project (GRUB): 2007-2016
- Enlightened Perl (RT): never[12]
- QEMU (KVM): 2010-2016, except '12
- Linux Foundation (KVM): 2008-2016

Evidence about discussion and participation can be found in Appendix B.2. This data might be helpful for continuing research on GSoC, or on innovation introduction in general, as the GSoC program likely attracts other innovations [9].

[9]The author could not find any research on GSoC, so this indeed has to be done.
[10]It could be called a SIP, cf. `https://en.wikipedia.org/wiki/Statistically_improbable_phrase`
[11]Bochs and FreeDOS each had exactly one email about GSoC
[12]see [rt:4683]), but maybe there is an even higher umbrella via the Perl Foundation.

3.3 Conclusion

In this chapter first a methodology for searching innovation related emails was presented.

Secondly, the results of applying this search methodology was described. One major result is the fact, that it is indeed possible to track down certain innovation episode continuations, as long as the keyword is not to generic. In the context of this thesis the search for continuations was not executed to its fullest degree. At a certain time, e.g. given a certain number of positive search results, the search was stopped as the trade-off between necessary effort and expected further insights was starting to become negative.

The identified innovation episodes from the basis data set analyzed in the next chapter.

4 Long-Term Observation of Innovations

In chapter 3 it was demonstrated that there is indeed data available towards the continuation of innovation episodes started in 2007. It remained unclear though, what a continuation is and how this relates to the theory by Özbek. To clarify this is the goal of this chapter.

4.1 Innovation Continuations

To further investigate the topic of what happened along several years of OSS development, the terminology of Özbek's innovation analysis has to be extended.

It was observed that innovation attempts can be repeated and fail every time. Consider for example the episode `autotools@xfce`(A.1.1), where some voices are raised towards replacing the build system of Xfce. Every innovation that was attempted was rejected relatively fast without a discussion, leaving the previous innovation in place.

As opposed to a chain of failing innovation attempts, there can also be chain of failing and successful innovation episodes. The most extreme case can be observed in `git@grub`(A.1.10) where the SCM is evolving over the years from CVS to Subversion to Bazaar to Git, with several failed innovation attempts in between.

Core Innovation From this we can infer that there can be *chain of episodes* towards the same topic. Here the term *core innovation* is proposed for and innovation that is "innovated" repeated times over a longer period of time. A core innovation is a process or a service that is used on a regular basis during the main activities performed by the developers (i.e. during development and communicating about it), for example bug tracking processes, SCM systems and processes, release processes, build tools, communication processes.

In most cases the steps to introduce those innovations are more difficult, because of its *radicality*, such that the elements of the Lifecycle (discussion, execution, usage) can be observed on their own.

Peripheral Innovation These are, opposed to the core innovations, are those that are unlikely to affect a large fraction of developers on a regular basis. Consider for example `foundation@argouml` (A.2.7), for which many developers might not even remember that ArgoUML once joined the Software Freedom Conservancy. They are unlikely to have a long term influence on their own, but they can be considered as the small steps that are necessary for progression.

The episodes can have, dependent on their group, a very different likelihood for continuation. Some episodes are just one shot-episodes and it is unlikely that they directly continue, some will pop up regularly if the underlying problem is not solved. Which leads to the next section, where such state changes as occurring problems are further explained.

4.2 Innovation Lifecycle revised

Özbek proposed the innovation lifecycle as one of his basic results. Looking at Figure 2.1, it becomes clear that this is not a cycle. Of course it should be a cycle as the term already indicates. This view becomes even more important when looking at the large scale development of projects.

A main result of this thesis is, that the lifecycle is a cycle indeed, by incorporation the found continuations, see also Figure 4.1. The extensions that fulfill the innovation lifecycle are:

- *State / Perspective Changes*, which are certain events that can transform a situation with a successfully running innovation into a problem. Or it could happen that an idea comes up, such that the successful innovation can be improved.
- If an *idea or problem* comes up, it needs a continuation to solve it or to establish the improvement.

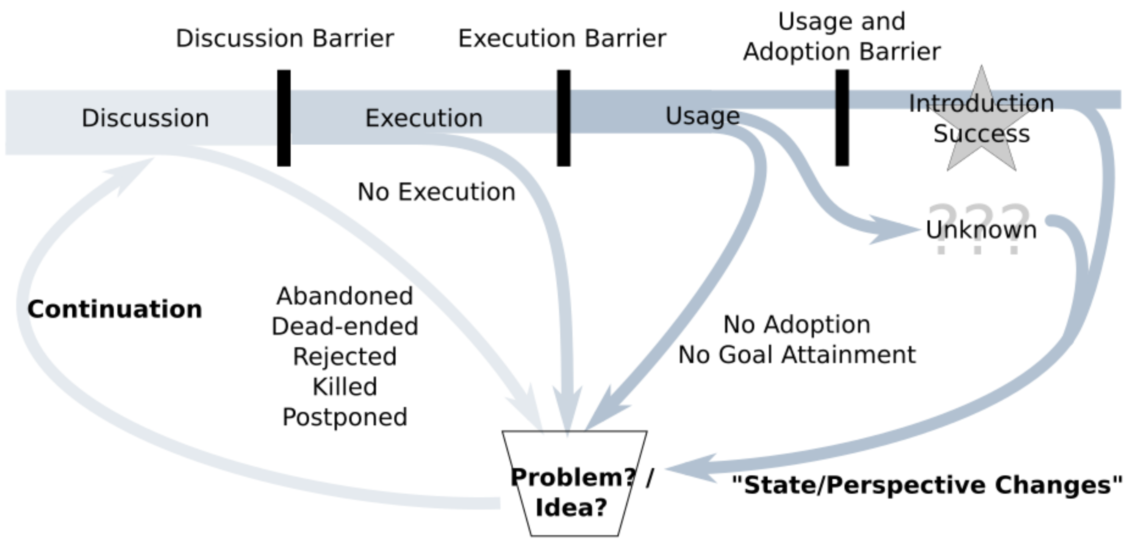


Figure 4.1: Extended Innovation Lifecycle (adapted from Özbek)

In the following, the different possible cases are supported by observed examples.

First example is `bug tracking@grub` (A.1.4), where the success of today can be questioned, because there is now three bug trackers instead of one, which is a state change. This change could lead to problems, because it is not defined for the user which bug tracker to use. There is no evidence in the project mailinglist that this was perceived as a problem though.

At `git@grub` (A.1.10) there are several iterations across different SCM systems (cf. section 3.2.3.1). There first point where the innovation changes from success to "problem" state is in June 2008 where a data loss occurs at the hosting platform. At this point an innovation introduction is attempted, with the argument that a distributed VCS wouldn't have suffered from a data loss at a central location. The innovation is rejected and the problem is solved by using a backup to restore the Subversion repository (*execution*) [grub:10959]. The usage of Subversion continuing at this point and the innovation can be considered as successful again. Here the full Lifecycle is traversed, starting with the *continuation*.

In September of 2011 the maintainer of gEDA stops being active. His last email geda-user39246. He probably did not hand over his duties, therefore the project is without a leader from there on. This is the *state change* that leads to problems later on, especially a discussion about who should the maintainer should be in 2015 which is still ongoing without a solution[geda-user:51758]. A problem that follows from a missing maintainer is, that no one is responsible for the release process, therefore the last stable version is from November 2012, which was released by a core member who does not want to be the maintainer [geda-user:49318].

An example for a transition from the *failed/problem* state, without coming from a *successful* innovation, is `autotools@xfce` (A.1.1) where there was a failed innovation attempt in 2007. As it was observed, for some developers the problem still exists. They complain about autotools and attempt follow-up innovation episodes (*continuation*) to solve the problem.

When looking at the switch from Subversion to Bazaar in `git@grub` the state change is not as clearly visible from the mails, but a fact is that the maintainer has switched in the meantime. The old maintainer was against DVCS, whereas the new maintainer seems less concerned, he starts a Bazaar-*Adapter*. This *perspective change* motivates (*idea*, *continuation*) another core developer to *discuss* using Bazaar as main VCS. After a decision this is *executed* and *used*. [grub:13962].

4.3 Concepts

In the following section, examples are presented that confirm Özbeks concepts beyond the episodes that happened in 2007.

Innovation Decision

- In `bug tracking@grub` (A.1.4) the individual decision which bug tracker to use is relevant today, as no bug tracking process is defined by the project. The users can choose between three bug trackers which might, if the project gets more attention again, lead to problems. To compare this with section 4.2, a slight state change is taking place here.
- In `git@grub` (A.1.10) GRUB had to take several decisions in its follow-up episodes towards. As an example the decision towards Bazaar is taken in a much more open way by a new maintainer [grub:13706], than it was done by Okuji, the first maintainer, who killed many attempts by executing his maintainer might.
- In `autotools@xfce` (A.1.1) no organizational decision is taken, because the project seems happy with autotools.
- In `contribute@bugzilla` (A.1.8) the individual decisions of e.g. doing the self-introduction is regularly taking place such that this can be considered a success.

- In `branching@geda` (A.1.3) in the usage of this innovation (i.e. when doing a relase), the decision has to be taken to actually perform a release. Later on especially the missing maintainer might is relevant such that the release cycle seem to stall.
- In `branch for patches@argouml` (A.1.2) there are the individual decision of using branches visible.

The innovation decision is a very central concept for a lot of innovations. Özbek splits the concept into the organizational decision and the individual decision, which are indeed two very different things.

The organizational decision is important if not necessary for all core innovations. Because every innovation that is used on a regular basis has to be supported by the project. This can, as described by Özbek, either happen through a representational collective or via a uniform decision by the maintainer.

On the technique of voting only two relevant examples were found. The first is a strong opinion against it:

> gEDA is a doitocracy. If you want something done, do it. If you want it done badly, convince others to help you do it. Voting just isn't going to work, because losing a vote doesn't make a developer want to work on the winning idea. [geda-user:45918]

The second example is the introduction of Bazaar in Bugzilla in 2008 where the maintainer pulls of a vote. [bugzilla:7962]

Hosting

- In `git@grub` (A.1.10) staying with Savannah, and as such the *capability* has a huge influence on the discussion and progress of all the SCM decisions.
- Also in `bug tracking@grub` (A.1.4) (as above, with git@grub), Savannah is the central platform that GRUB wants to use. Notably, other platforms are used over time as well (e.g. the debian bug tracker). The fact that the innovator can change the website, is another Hosting aspect.
- In `branch for patches@argouml` it was found that Gerrit on a home server of Linus

Tool Independence

- The episode `autotools@xfce` (A.1.1) is related to tool indepence, because Xfce slightly enforces the build tool upon the developers, whereas it might be possible to use another one.

Adapter

- The adapter is of course relevant for the SCM episodes (such as `git@grub`), enabling the projects to learn about the Systems before using them.
- but also the project maintained IRC Adapter of `contribute@bugzilla` (A.1.8) is still alive. Interesting to note, because this is the only project maintained adapter that I know of.

Partial Migration

- Partial migration is mentioned in `autotools@xfce` (A.1.1) as an argument that the switch is not as *radical* as one might think.

Time

- `git@grub` (A.1.10) has a lot of good timing, for example the innovator that uses a data loss event to advertise his innovation (still fails, but the thread got good attention).
- `branching@geda` (A.1.3) we see how timing fails in several dimensions. Especially the time to market, but I'm not sure whether this concept is defined by Özbek.

Radicality

- `branching@geda` (A.1.3) the follow-up proposition to allow unstable versions in distribution could've been less radical maybe.

Enactment Scope

- might be relevant in the failure of some follow-up episodes of `autotools@xfce` (A.1.1).
- the switch from Subversion to bazaar in GRUB could be viewed as a very small enactment scope or as partial migration. The gist here is, that the maintainer sets up the bazaar already on Savannah, such that nothing else is left but flipping the switch.

This evidence shows that the concepts are still relevant for the outcome of innovation episodes.

4.4 Conclusion

In this chapter, first the difference between those innovations that have a longer term history of innovation episodes and those that do not was shown. Then the innovation lifecycle was revised with new insights and at last the eight core concepts of Özbek were confirmed by more evidence.

5 Closing Remark

This thesis was about three questions related to a predecessor work by Özbek [9]:

- Whether there are continuations of the innovation episodes that were observed during the year 2007.
- If the theory that was created by Özbek is still valid and can be applied to current data.
- While Özbek could handle one year of data, one of the main challenges of this thesis was to deal with ten times more data.

The approach towards these questions was from the viewpoint of qualitative research, always having in mind where more could be learned.

The main contributions of this thesis are:

- A full scale search for innovation continuations across ten years of email data for 13 projects. It was shown that it is possible to find a large fraction of innovation related emails by keyword filtering.
- A detailed analysis of the continuations of the most relevant innovations of 2007, by which understanding towards the general question about the development process of OSS projects could be increased.
- The theory that was created by Özbek could be confirmed in some points and was extended with a GTM-related approach in other points.

Özbek's theory was mostly confirmed. In no point his theory was proven wrong. It was show in this thesis that how his theory could be extended towards including long term observations.

Open to further research are the following points:

- A detailed and more rigid GTM-driven approach towards analyze the innovation episodes that were found during this research project.
- Research towards finding innovation episodes if no starting point is given. Do innovation related emails have certain properties that allows for a better automatic identification?
- An embedding of the findings into other current research about Open Source Software and Innovation introduction in general.

A Episode Continuations

A.1 Core Episodes

A.1.1 autotools@xfce

What happened in 2007? In January 2007 the Xfce package maintainer for Xubuntu starts a discussion about replacing the buildsystem of Xfce, autotools [grub:13023]. After the reaction is mostly negative, i.e. towards keeping autotools in place, the innovator accepts the answer and explains that he just wanted to make sure that a conscious decision is taking place.[1]

This was categorized as *failed.postponed*.

Follow-up Apparently there are other developers who dislike autotools which is shown in the following occasions:

- In Feb 2009 a developer fails to compile Xfce, another developer responds: "Blame autotools" [xfce:16701]
- In May 2012 a developer (telling from the email address, it is another package maintainer), who volunteers to clean the configuration files up, but is brushed off by the maintainer. This happend two years after the innovator had made his confusion about the configuration files public in Nov 2010. [xfce:20452, 19028]

Outcome 2016 By today this innovation can still be considered as *failed.postponed*, because the project sees no need to replace autotools.

A.1.2 branch for patches@argouml

What happened in 2007? In 2007 the maintainer *successfully* introduced the possibility of using SVN branches for patches.

[1]See also Özbek's section on Path Dependence [9].

Follow-up There is evidence, that the branches are still used.

- January 2008: “So, I’ve got a branch of my own now.” [argouml:5807]
- July 2008: A problem occurs with a user branch [argouml:7052]
- In May 2010 they have a discussion about how to handle fixes: “we need to work out how and why we decide a defect fix should trigger a defect fix in branch.” [argouml: 9507].
- In May 2012 Gerrit is introduced [argouml:10435]. This enables for a patch reviewing cycle. This is a related Adapter. Note that it is offline by today.

Outcome 2016 The outcome is *unknown*, it is unclear at this stage of research which branching process is mainly used. It is possible that more evidence is available, for example in the developer cookbook[2], but in the emails no clear direction was found.

A.1.3 branching@geda

What happened in 2007? In 2007 gEDA, motivated by having a product that is mature and stable enough to be version 1.0, switched from CVS to git to be able to have a development process with stable and development branches. This episode was driven by a core developer with support of another core developer and the maintainer, who executes the switch *successfully*. This episode also led to a success of `git@geda`.

Follow-up In December 2008 a core developer asks, if the project could release stable version 1.6.0 in January 2009 to meet the deadline of Ubuntu Jaunty. This fails and misses another deadline when it appears in October 2009. [geda:7965]

Outcome 2016 gEDA is having some problems, for example it does not have a lot of stable releases which leads to outdated software in different Linux repositories. By today it seems the community is quarreling. To tell whether the branching scheme is having an influence there, more research is necessary. So the outcome/state of this process can be considered *unknown*.

A.1.4 bug tracking@grub

Background Information: GRUB-Lecacy vs. GRUB2 In 2006 the GRUB project did a complete rewrite of the software, called GRUB 2. The previous software is often identified with the term GRUB Legacy. Over the course of 2007 the development is focused towards GRUB 2 only, while GRUB Legacy development is gradually phased-out. This episode can be understood as a condition for this transition.

[2]`http://argouml.tigris.org/wiki/Cookbook`

What happened in 2007? In June 2007 it comes up, that GRUB 2 is missing a functional bug tracking process. The project's BTS[3], hosted at Savannah [4] is filled (occupied) with bugs from GRUB Legacy, and as such unusable for the use with GRUB 2. The episode from 2007 is *successful*, after four attempts in six months, in cleaning up the project's BTS and thus making it usable for GRUB 2 bugs.

Follow-Up Note that this happens still in December 2007, but Özbek did not describe it.

An episode that follows [grub:4093], is initiated as well by the innovator who cleaned up the system and is discussing several aspects:

- The documentation of the bug reporting process has to be modified to advertise the new BTS. This can be seen as part of the *execution phase*, because documentation is necessary to teach users to actually use the new system. Note, that often bug reporters are not part of the developer community, but the user community [6]. The corresponding webpage is successfully updated with a pointer to the new system by the innovator himself [grub:4110]. This website still exists today[5] with a similar content, so this part can be considered successful.
- A technical discussion about the different mailing lists (bug-grub, grub-help,...)
- For the question on how to continue discussing some bugs on the mailing list without losing context, no answer was found (-> thread *abandoned*). [grub:4118]

Outcome 2017 The first observation is that this exact situation did not reoccur: There is no GRUB 3 which could have suffered from a BTS that is occupied by GRUB 2 bugs. So this particular issue can be consideres as *successful*, although the whole BTS process seems to be rather weakly defined.

A.1.5 bug tracking@argouml

What happened in 2007? In 2007 the maintainer changed the process of closing a bug and documented the new process in the cookbook, thus it was executed. This was classified as `unknown.success` because it could not be verified, whether the throughput was in fact increased (which was the stated goal).

Follow-up

- In September 2008 the process is questioned by a developer [argouml:7404]. This leads to a bug tracker cleaning session where the maintainer closes a lot of bugs and adapts the bug tracking process in the Cookbook. [argouml:7409,7440]

Outcome 2017 The process was under refactoring but there is no evidence for problems, as such this innovation can be considered *successful*.

[3]bug tracking system
[4]`http://savannah.gnu.org/bugs/?group=grub`
[5]`http://gnu.org/software/grub/grub-bugs.html`

A.1.6 bug tracking@monetdb

What happend in 2007? A core developer proposes to use the bug tracker as an extended documentation, in such a way that every issue that is fixed in the code should reference a corresponding bug in the bug tracker. This means, that if the bug was not reported before, it had to be created. This episode *fails* by not drawing enough attention to it.

Follow-Up

- In 2009 the maintainer asks the community to make sure that "each failing test is documented in a bug report" and that "each closed (fixed) bug report is backed-up by a test in CVS to monitor the status of that very query or usage scenario." [monetdb:2064].
- A related episode of 2010: The bug tracker is moved from SourceForge to a self-hosted Bugzilla instance. [monetdb:2371]

Outcome 2017 It is *unknown* whether the call of 2009 was successful, because the author did not do further research here.

A.1.7 bug tracker@xfce

What happened in 2007? Two episodes that happened in 2007 are `bug milestone@xfce` and `bug tracking cleaning@xfce`. The first one fails by drawing no responses, the second one *succeeds* in cleaning up the bugtracker.

Follow-Up In Mai 2009 a developer wants to restructure the bugtracker. [xfce:17436]

Outcome 2017 There is no problem with the bug tracker evident, so this can be considered a *success*.

A.1.8 contribute@bugzilla

What happened in 2007? In his announcement email in February 2007, the maintainer explains his motivation and goal ("to have 10 active reviewers from a growing pool of contributors, by the end of the year"). He comes up with a five point plan with ideas he collected in a thread from two years before. The plan consists of several small adaptions, like publishing and advertising a guide on how to contribute.

Follow-up

- First, some outcomes of the sub-episodes in 2007 were unclear. By today, in hindsight, it's possible to say a bit more.
- In January 2011, the maintainer describes the 2007 episode as a success and tries to continue by asking for more ideas on how to actually get developers and also asks for someone volunteering to be responsible for this community management. [bugzilla:8973]
- In April 2011 the maintainer announces that he has a new job and that he will not abandon the project [bugzilla:9055]. In February 2012 his position as a Release Manager is replaced by another developer [bugzilla:9190]. Finally, in November 2012, announces to leave the project [bugzilla:9338].

Outcome 2017 The overall outcome is hard to tell. The most obvious thing that is still *successfully* taking place is a self-introduction by newcomers which also draws responses [bugzilla:9869,9949,10016].

A.1.9 design competition@bugzilla

What happened in 2007? A developer proposes to organize a design competition for the UI of the product, which is *rejected* by a core developer.

Follow-Up Interestingly, in 2010, a design competition is announced by the maintainer. [bugzilla:8830] It seemed to attract some submissions and a winner was selected to design Bugzilla 5.0[6]

Outcome 2017 The Bugzilla 5.0 seems not to have been changed to the winner's proposal. There was also no evidence of activity by the winner on the mailing list. So the outcome of this can be considered *failed*.

A.1.10 git@grub

What happened in 2007?

- In May 2007 a core developer asks if "GRUB is planning to move to another SCM" and implies switching to git [grub:3022]. After some discussion (most answers positive), the thread is abandoned by the innovator.
- In December 2007 another innovator proposes to switch to git. After a *Representational Collective* finds common ground [grub:4091], the maintainer kills the discussion rejecting git. This episode does not fully fail though, because an *Adapter* is created by the innovator, which is used by some project members and is regularly advertised (*Signaling*) by the innovator. [grub:4383,5120,6080,6830]

[6] `https://bugzillaupdate.wordpress.com/2011/03/31/winner-of-the-make-bugzilla-pretty-contest/`

Follow-up

- During GSoC 2008, a student mentions at several occasions, that he works with a Bazaar repository, which is another *Adapter* to the project. His actions are no full innovation episodes, but can be considered as *Signaling* as well.
- In the beginning of 2008, a core developer asks to switch to SVN, which is postponed, because the *Hoster* Savannah does not support Subversion yet and it is argued that switching is not worth it. [grub:5120]
- In June 2008, Savannah supports Subversion (*Hosting Capability*) which is noted by a developer on the list [grub:6603]. This is a fast, successful episode, because everyone (*Representational Collective*) is at this point already convinced that switching away from CSV is a good idea. The project sends a request immediately to Savannah and the successful migration is announced shortly after. [grub:6603,6823]
- June 2009 Savannah (the *Hoster*) suffered from data loss[7]. This causes the SCM discussion to be restarted by a git proponent. Although most developers seem to be in favour, no decision is taken. The opinion of the maintainer who strongly argued against git is still present [grub:10946], but the maintainer himself is not considered as active anymore [grub:10947].
- In October 2010, on the same day (coincidence? look for IRC logs?), two things happen: The maintainer (new one) announces a Bazaar mirror and a git proponent starts an innovation episode for experimental branches (*Bedarf erzeugen*-Strategie). The latter already was a strategy to introduce git in gEDA.
- In November 2010 the Bazaar mirror is announced as main repository, after the maintainer agrees with another maintainer and no other objections are raised. [grub:13962]
- Switching to Bazaar leads to two problems. First, Bazaar seems to be susceptible to data loss, which leads to another call for git by a developer [grub:14440]. Second, the git Adapter introduced in 2008 fell out of sync, because it still followed the Subversion repository, and had to be reconfigured [grub:15526].
- Three years later, in June 2013, another innovation episode is started (who is active since beginning of 2013) to introduce git. He succeed by talking beforehand to the maintainers on IRC (exploiting *Maintainer Might*). Four months later the maintainer executes the switch.
- [grub:21036] GIT workflow discussion

Outcome 2017 git is still *successfully* in place.

[7]`http://lists.gnu.org/archive/html/savannah-users/2009-05/msg00009.html`

A.2 Peripheral Episodes

A.2.1 ask smart questions@bugzilla

What happened in 2007? A core developer proposes a document that explains “how to ask good questions”[8]. It is supposed to simplify a famous but more difficult document[9] on the same topic and is positively received by another core developer and the maintainer [bugzilla:6540]. The former adds the document in an execution step “to the text that is mailed to non-subscribers who post to the support-bugzilla list via email. (The one that tells them they need to subscribe before they can post and warns them that it’s a public forum and not a private support channel).”[bugzilla:6543]

The episode was considered a *success*.

Follow-Up Report

- The document still exists by today on the external server.
- Up until 2012 it is linked regularly on the Bugzilla support mailing list. In the beginning it was linked by others, after 2010-12 almost only by the creator[10] which shows that also the problem that this innovation was supposed to solve still exists. On the other hand the document introduced in this innovation was not established, as a comparison with the more famous question asking guide shows[11]. The latter is cited more often and up until today.
- The mailing list however does not link to this document anymore. After I notified the maintainer that this link is not existing anymore, he told me it might have gotten lost in an update of the mailing list service. By today this is still not fixed. It seems not to be a priority.
- Neither the document nor the topic of asking good questions does come up again on the developer mailing list (keywords used: "ask good questions").

Outcome 2017 The episode can be considered as *failed.no adoption*, because the document is not established in the project.

A.2.2 changelog@grub

What happened in 2007? A core developer requested to have another changelog format. He *fails* in establishing this innovation.

Follow-Up

- The maintainer changes in 2015 the way the changelog is generated. He creates a makefile for it. [grub:22849]

[8] `http://www.gerv.net/hacking/how-to-ask-good-questions/`
[9] `http://www.catb.org/~esr/faqs/smart-questions.html`
[10] `http://bit.ly/2gwCubh`
[11] `http://bit.ly/2ka1ryk`

Outcome 2017 The Changelog generation was *successfully* automated.

A.2.3 checkstyle@argouml

What happened in 2007? After ArgoUML upgraded to Java 5, a core developer discusses whether a feature of the programming language (namely autoboxing) should be prohibited for performance reasons. The innovation fails, no decision is taken.

It *failed*.

Follow-Up The keyword autoboxing does not appear again on the list

Outcome 2017 Still failed, but no one cares.

A.2.4 commit script@xfce

What happened in 2007? A small script to support the changelog process is introduced in this innovation episode. Özbek marked this as unknown adoption, because it's unclear whether this script was used.

Follow-Up Searching for the keyword xdt-commit, also there is evidence that makes usage of the script seem probable:

- [xfce:17323]
- Script still exists today and was last changed in 2011[12]

Outcome 2017 This should be considered as *success*, because the script is still in the repository and more or less kept up to date.

A.2.5 contribute@grub

What happened in 2007? A developer is asking for a change on the website to add information on "how to send in patches"[grub:3905].

Follow-Up For the keywords "website"/"contribute" nothing comes up.

Outcome 2017 Still failed.

A.2.6 design@uboot

What happened in 2007? A design document was *successfully* introduced.

[12] `https://git.xfce.org/xfce/xfce4-dev-tools/log/scripts/xdt-commit`

Follow-Up It seems still to be used. It's regularly linked on the mailing list[uboot:145556, 147027,172989] and the wiki page is kept up to date[13].

Outcome 2017 This is still *successful*.

A.2.7 foundation@argouml

What happened in 2007? ArgoUML successfully joins the SFC to be able to handle GSoC money. It was *unknown* if the goal of *more efficiency* was reached.

Follow-Up

- ArgoUML is still in SFC.[14]
- In 2008 they joined Eclipse Foundation [argouml:6662].

Outcome 2017 Still successfully part of the organization. Even one more: The Eclipse Foundation. If this now is an overall success, is still *unknown*[15].

A.2.8 http expires@monetdb

What happened in 2007? A new test version of MonetDB did not reach everyone immediately. In the case of the innovator, an older version of the test client was still cached in his browser. As a solution he proposes to adapt the HTTP Header to expire immediately, which is rejected, because the developer he's in contact with does not have access to the webserver configuration.

Follow-Up

- By today this configuration seems not to have changed, as the TestWeb still does not have and expiration time set in the header[16].
- But the test/build server seems to have changed.
- search: "http expires", "MonetDB" does not give more results about this innovation.

Outcome 2017 Although the introduction of this is still *failed*, no evidence could be found on the mailing list that this is still a problem

Outcome 2017

[13]`http://www.denx.de/wiki/rdiff/U-Boot/DesignPrinciples`
[14]`https://sfconservancy.org/projects/current/`
[15]How should we know?e.g. by interviewing the project participants. But the project is dead anyway.
[16]`curl -I http://monetdb.cwi.nl/testweb/web/status.php | grep -i expires` shows no results

A.2.9 online demo system@flyspray

What happened in 2007? A demo system[17] was provided on a private server.

Follow-Up

- After this innovation it seems that `demo.flyspray.org` was created. But both demo system do not exist anymore. [flyspray:5404]

Outcome 2017 This innovation might be considered *abandoned*.

A.2.10 observer@argouml

What happened in 2007? In February 2007 the maintainer announces the creation of a new role, the “observer”, who’s responsibility is to “monitor mailing list communication and help with answering recurring questions, identify duplicates in the bug tracker, etc.”[9]. No one takes this role, the episode is thus *failed*.

Follow-Up Looked through all 50 mails containing the keyword “observer” and did not find anything about this process.

Outcome 2017 The episode is still *failed*.

It can be said that the problem is still *successfully* solved.

[17] `http://gosdaturacatala-zucht.de/flyspray/`

B Notes

B.1 Evidence for SCM switches per project

- ArgoUML: Christopher said it was SVN and tigris still shows SVN.
- Bochs: [bochs:9125] (I found it by searching for "SVN" and scrolling through the list. It's not many mails on there)
- Bugzilla: CVS->bzr: [bugzilla:8275], bzr->git [bugzilla:9576]
- Flyspray: SVN->git: [flyspray:8038], then there is some time where SVN and git are running parallel, which creates a mess. See e.g. the thread [flyspray:8213]
- FreeDOS: [freedos:4837] (coded by Christopher)
- gEDA: [geda:4334] (coded by Christopher), the "campaign" apparently was starting before 2007.
- GRUB: CVS->SVN [grub:6823], SVN->bzr [grub:13962], bzr->git [grub:20817]
- KVM: [kvm:1399] (by Christopher)
- MonetDB: [monetdb:2365]
- ROX: Filer: [rox:9371], more: [rox:9403]ff, Lib:[rox:9647], ...? TODO. Also: is it finished/abandoned?
- Request Tracker: SVN->git [rt:5615]
- U-Boot: git since 2006, said Christopher
- Xfce: SVN->git (first as Adapter [xfce:15052]): [xfce:17320,17402,17945]

B.2 Collection of GSoC emails per project

- ArgoUML
  - 2007: students interested [argouml:4898,4931], collecting ideas [argouml:4936, 4941], participate [argouml:5092], and more (organizational. see Özbek)
  - 2008: discuss [argouml:5982], apply [argouml:6016], participate [argouml:6035], lots of student applications TODO, announce start [argouml:6375]
  - 2009: discuss [argouml:8046], org rejection [argouml:8373]
  - 2010: discussion [argouml:9224], org rejection [argouml:9247]
  - 2011: discussion [argouml:9922], org rejection [argouml:9991]
  - 2012: discussion [argouml:10382]

- Bugzilla
  - 2007: discuss participation [bugzilla:6200]
  - 2008: ideas collection [bugzilla:7048,7077], students participate [bugzilla:7173]
  - 2009: ideas collection [bugzilla:7698]
  - 2010: ideas collection [bugzilla:8450]
- gEDA
  - 2007: discuss[geda:3021,3026], org accepted[geda:3104], student apply[geda:3354], retrospective [geda:4843]
  - 2008: student interest [geda:5944,6154], org accepted [geda:6208], participants welcome [geda:6467,6474], results announced [geda:7105,7203], retrospective announcement [geda:7570]
  - 2009: discussion [geda:8226]
  - 2010: call for mentors [geda:8832], application... [geda:8841], ...rejected [geda:8894]
- GRUB
  - 2007: discuss participation [grub:2688], students are interested [grub:2731,2744], participation TODO
  - 2008: discuss participation (announce) [grub:5792], participation/slots [grub:6075], students are interested [grub:7205], work gets done [grub:8146,9134]
  - 2009: discuss participation [grub:9863]
  - 2010: discuss participation [grub:15613]
- KVM
  - 2008: communicating with KVM: because of Linux GSoC [kvm:16670] and coreboot GSoC [kvm:19161]
  - 2010: collecting ideas [kvm:48332], accepted [kvm:49005], lots of applications TODO, accepted students [kvm:51215]
  - 2011: discuss participation [kvm:66674,66971], discuss tasks (GSoc in "KVM call minutes") [kvm:67165,68261], acceptance [kvm:69413]
  - 2012: umbrella announces participation [kvm:86463], collect ideas [kvm:87182], umbrella rejected [kvm:88399]
  - 2013: collect ideas [kvm:104699], umbrella accepted [kvm:107755]
  - 2014: collect ideas [kvm:118644], umbrella accepted [kvm:119230], application information [kvm:119738]
  - 2015: collect ideas [kvm:131980], umbrella accepted [kvm:133292]
  - 2016: collect ideas [kvm:145925]
- Xfce

  - 2008: discuss participation [xfce:14850], application [xfce:14922], rejected [xfce:14962]
  - 2009: discuss (and reject) participation [xfce:16991]
  - 2011: discuss (and reject) participation [xfce:19239,19305,19379]
  - 2015: ask for participation [xfce:21733]
  - 2016: discuss participation [xfce:22213]

- U-Boot:
  - 2011: Beagleboard, had some kind of supporting project related to U-Boot[uboot:94210]
  - 2015: Minnowboard has "enabling ACPI support" as one of their GSoC projects [uboot:223965]. A student is working on it [uboot:231712,246345]
  - 2016: Minnowboard, a project that uses U-Boot (can be considered as Umbrella), requests for ideas [uboot:252049]
- Bochs:
  - 2008: connection to coreboot ideas [bochs:8353]
  - 2010: discussion [bochs:8802]
- Request Tracker: 2008: idea collection below umbrella [rt:4683]
- FreeDOS: 2010: one not really serious request [freedos:6270]
- Flyspray, MonetDB, ROX: None

C Project Release History

In this chapter all the releases of some of the observed projects are listed for reference. If there was an stable/unstable release scheme, only the latest of the unstable releases is shown. Also, if the first release is listed, the releases between first and 2007 release were left out.

Date	Release
1999 February	released as Open Source
2007 February	0.24
2008 September	0.26
2009 March	0.28
2010 March	0.3
2011 February	0.32
2011 December	0.34
2014 August	0.35.1

Table C.1: ArgoUML Releases

Date	Release
April 2001	1.0
August 2006	2.3
May 2009	2.4
Dec 2011	2.5
Sep 2012	2.6
May 2015	2.6.8

Table C.2: Bochs Releases

Date	Release
Sep 1998	2.0 (first OS release)
April 2006	2.22
May 2007	3.0
Nov 2008	3.2
Jul 2009	3.4(rc1)
April 2010	3.6
Feb 2011	4.0
Feb 2012	4.2
May 2013	4.4
Jul 2015	5.0
2016 May 16	Release of Bugzilla 4.4.12, 5.0.3, and 5.1.

Table C.3: Bugzilla Releases

Date	Release
Feb 2007	0.9.9
May 2007	0.9.9.2
August 2007	0.9.9.3
Dec 2007	0.9.9.4
Feb 2008	0.9.9.5
May 2009	0.9.9.6
May 2012	0.9.9.7
March 2015	1.0 Alpha
Nov 2016	1.0-RC4

Table C.4: Flyspray Releases

Date	Release
May 2007	1.0-20070526
Sept 2007	1.2.0-20070902
Jan 2008	1.4.0-20080127
Oct 2009	1.6.0-20091004
Nov 2012	1.8.0-20121118
Sept 2015	1.9.2-20150930

Table C.5: gEDA Releases

Date	Release
Nov 2007	1.3.0
Dec 2007	1.3.1
Mar 2008	1.3.2
May 2008	1.3.3
Aug 2008	1.3.4
	2008.10
	2009.01
	2009.03
	2009.06
	. . .
	2016.09
	2016.11

Table C.6: U-Boot Releases

Date	Release
Jan 2007	4.4
Jul 2009	4.6
Jan 2011	4.8
Apr 2012	4.10
Feb 2015	4.12

Table C.7: Xfce Releases

Date	Release
June 1994	First Announcement
Sep 2006	1.0
Jan 2012	1.1
Dec 2016	1.2

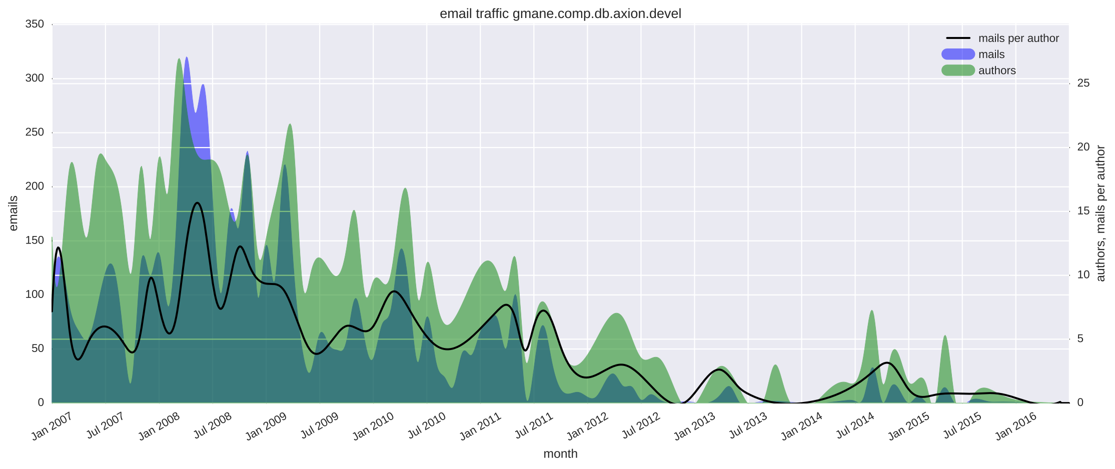
Table C.8: FreeDOS Releases

Date	Release
June 2006	3.6.0
July 2008	3.8.0
Apr 2011	4.0.0
Oct 2013	4.2.0
Feb 2016	4.4.0

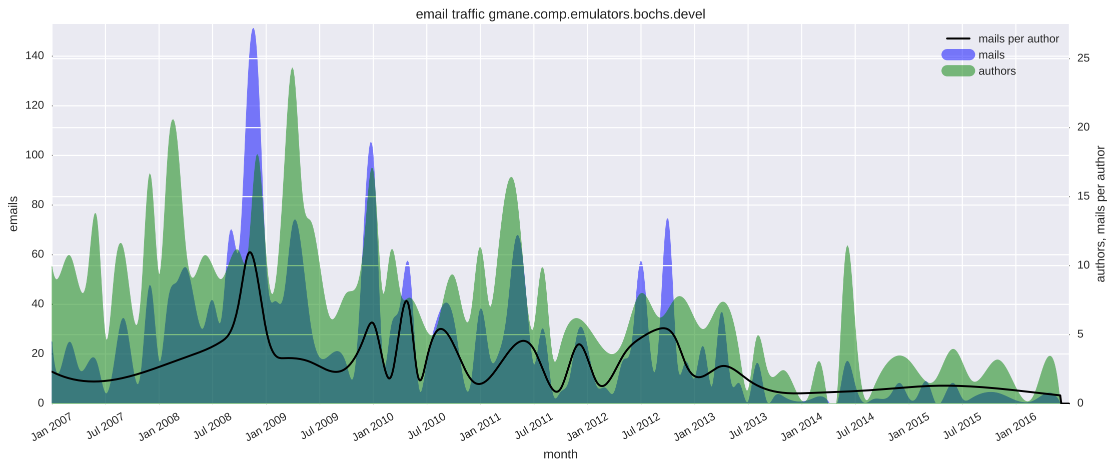
Table C.9: Request Tracker Releases

D Email Traffic

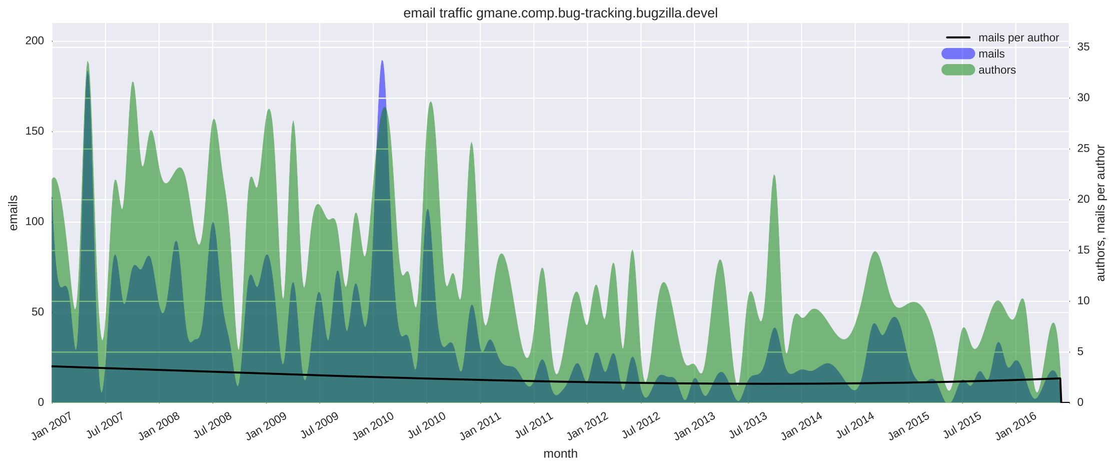
To get an overview of the mails sent throughout the observation period for each project, consider Figure D.1a to D.1m. Note that the mailinglists for gEDA and MonetDB were continued at another location which are not included in the visualization here.



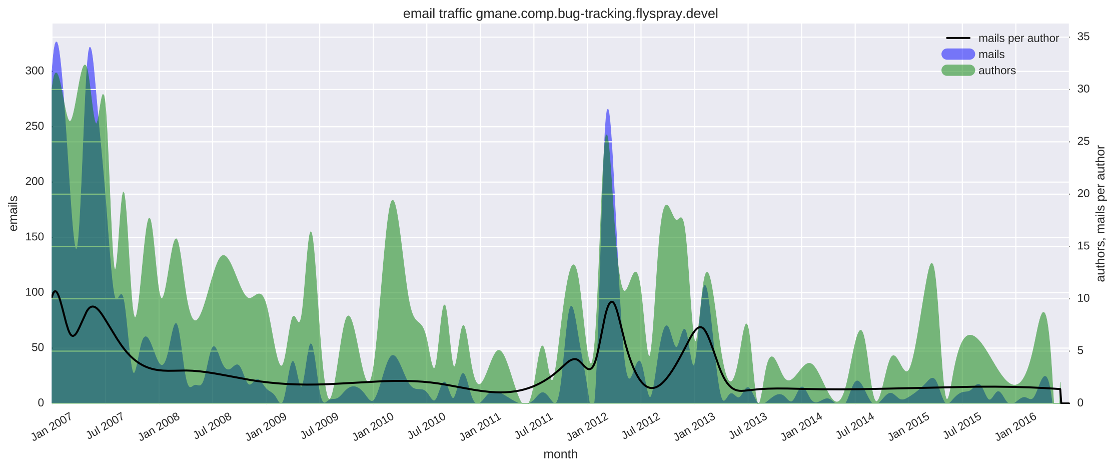
(a) ArgoUML



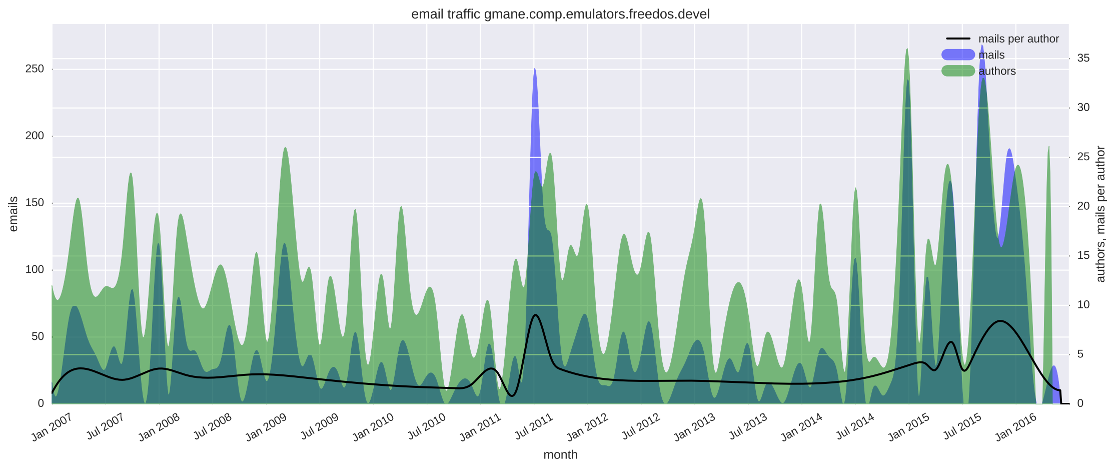
(b) Bochs



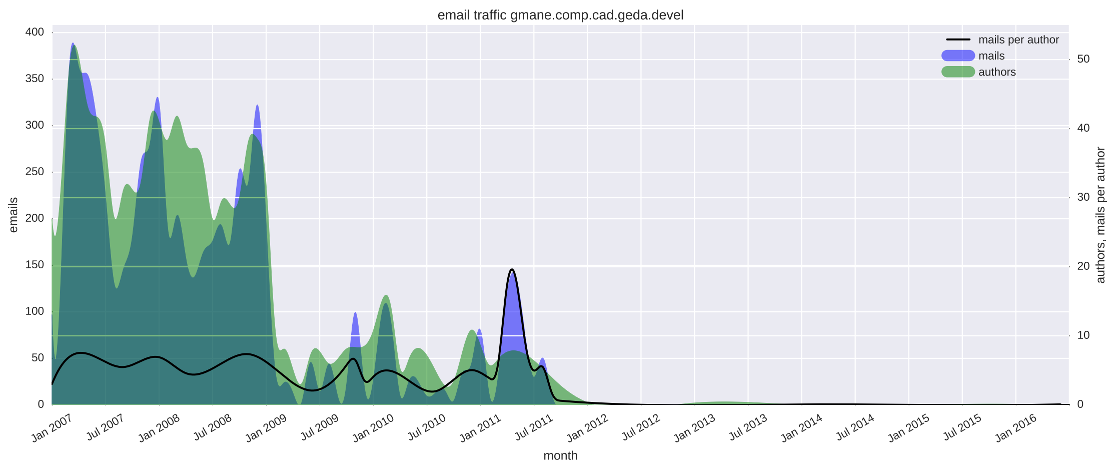
(c) Bugzilla



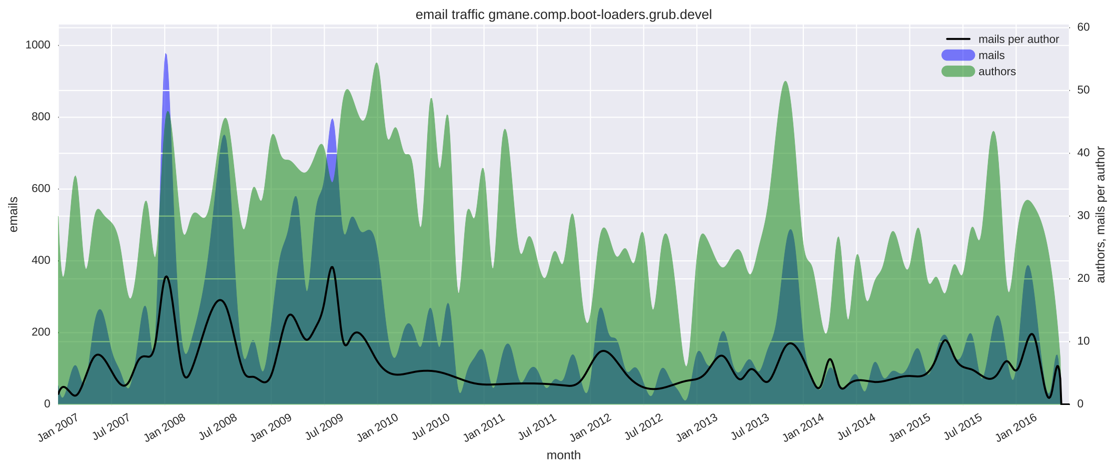
(d) Flyspray



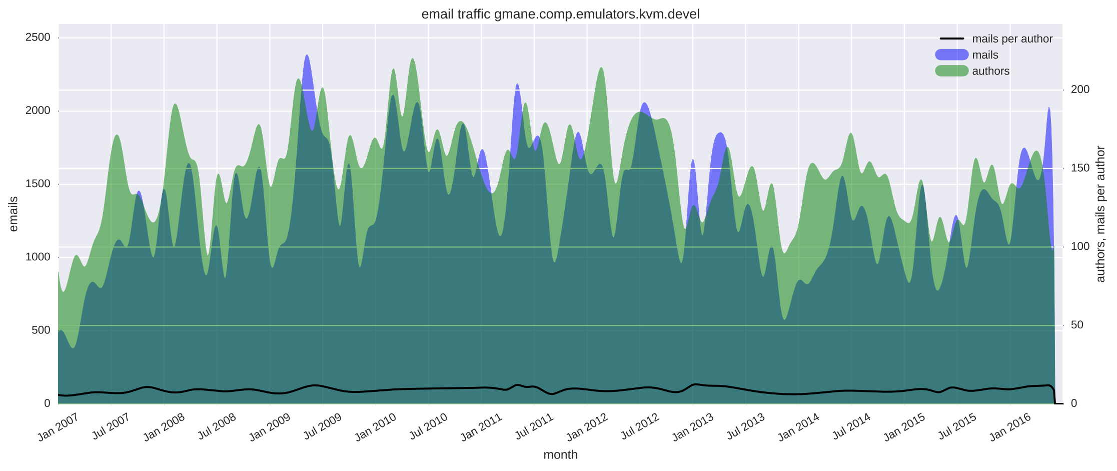
(e) FreeDOS



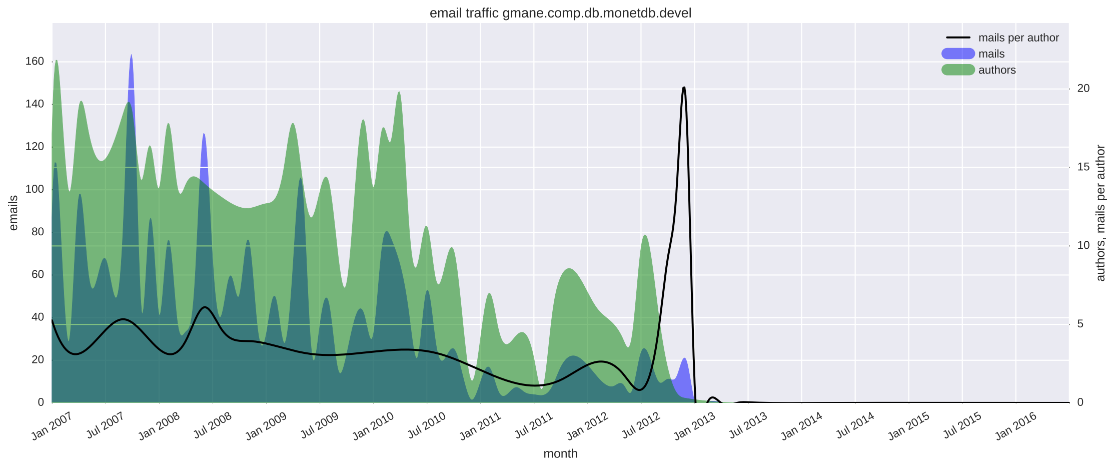
(f) gEDA



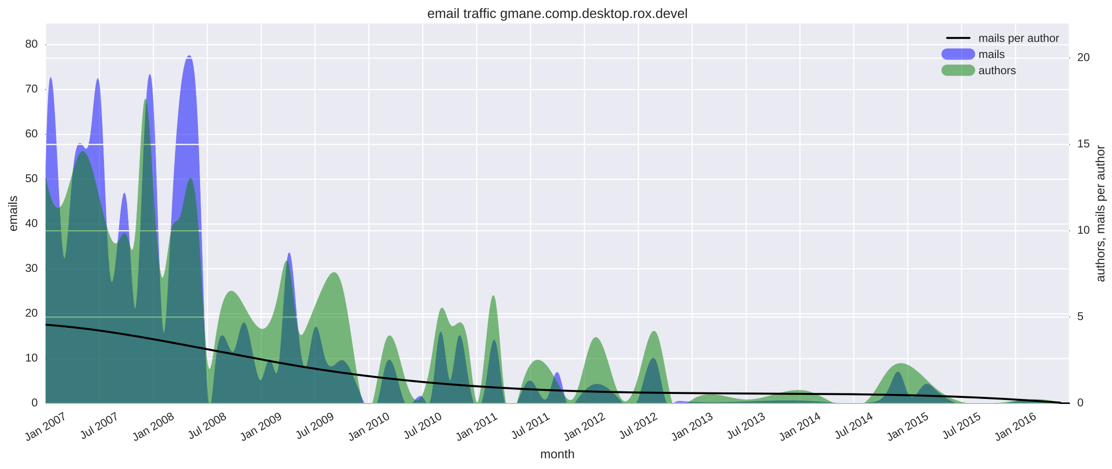
(g) GRUB



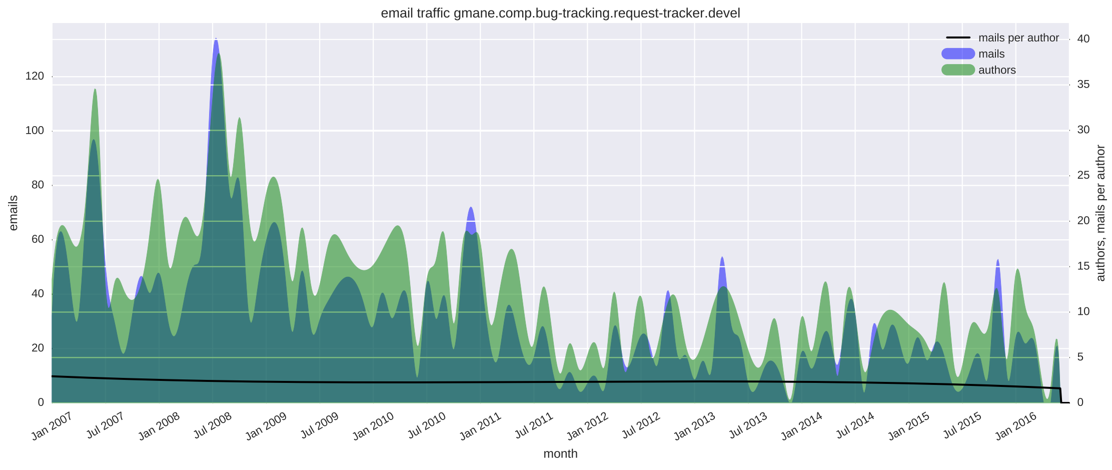
(h) KVM



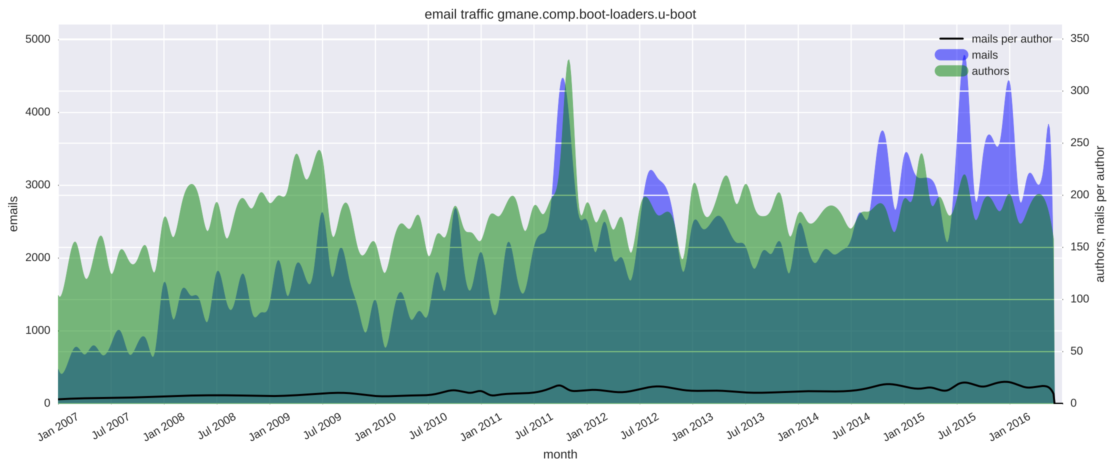
(i) MonetDB



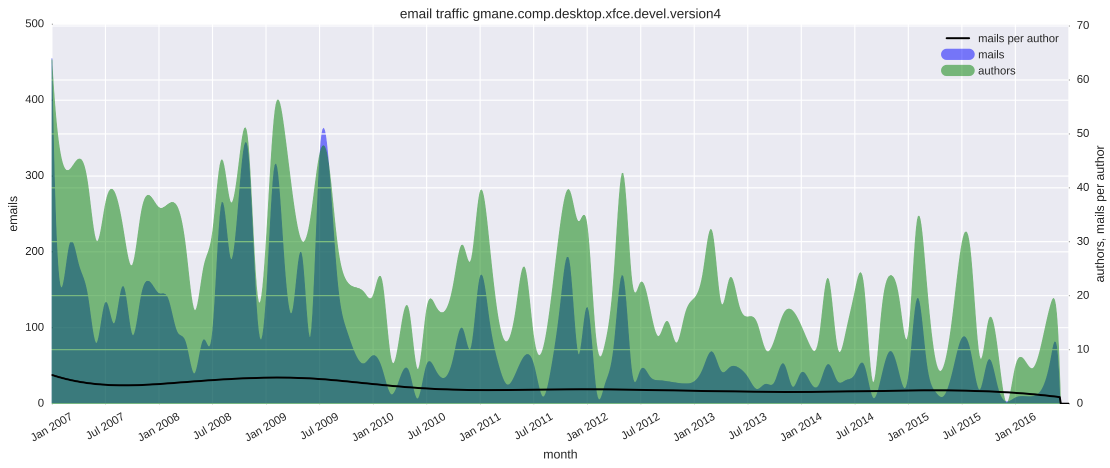
(j) ROX



(k) Request Tracker



(l) U-Boot



(m) Xfce

Figure D.1: This collection of figures shows the email traffic for the 13 project along the observation period. On the left y-axis, the number of emails is denoted, which can be observed over time along the blue plot. The green plot show the number of authors which is denoted on the right y-axis. The ratio of emails per author is also denoted on the right y-axis.

Bibliography

[1] Hudson Borges et al. “On the popularity of GitHub applications: A preliminary note”. In: *arXiv preprint arXiv:1507.00604* (2015).

[2] Clayton Christensen. *The innovator’s dilemma: when new technologies cause great firms to fail*. Harvard Business Review Press, 2013.

[3] Bird Christian et al. “The Promises and Perils of Mining Git”. In: *Mining Software Repositories*. 2009.

[4] Kevin Crowston et al. “Free/Libre Open-Source Software Development: What We Know and What We Do Not Know”. In: *ACM Computing Surveys* 44.2 (2012), pp. 1–35.

[5] Peter H Feiler and Watts S Humphrey. “Software process development and enactment: Concepts and definitions”. In: *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*. IEEE. 1993, pp. 28–40.

[6] Karl Fogel. *Producing open source software: How to run a successful free software project*. 2nd ed. O’Reilly Media, 2017.

[7] Anja Guzzi et al. “Communication in open source software development mailing lists”. In: *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE. 2013, pp. 277–286.

[8] Christoph Jentzsch. *Decentralized Autonomous Organization To Automate Governance*. `https://download.slock.it/public/DAO/WhitePaper.pdf`. Last visited 2017-01-27. Apr. 2016.

[9] Christopher Özbek. “Introducing Innovations into Open Source projects”. PhD thesis. 2010.

[10] Param Vir Singh. “The Small-world Effect: The Influence of Macro-level Properties of Developer Collaboration Networks on Open-source Project Success”. In: *ACM Trans. Softw. Eng. Methodol.* 20.2 (Sept. 2010), 6:1–6:27.

[11] Diomidis Spinellis and Vaggelis Giannikas. “Organizational adoption of open source software”. In: *Journal of Systems and Software* 85.3 (2012), pp. 666–682.

[12] Katherine J. Stewart and Sanjay Gosain. “The impact of ideology on effectiveness in Open Source software development teams”. In: *MIS Quarterly* 30.2 (June 2006), pp. 291–314.

[13] Strauss, Anselm and Corbin, Juliet. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 2008.

[14] Joshua R Tyler and John C Tang. “When can I expect an email response? A study of rhythms in email usage”. In: *ECSCW 2003*. Springer. 2003, pp. 239–258.