Máster Oficial
en Sistemas Telemáticos e Informáticos

Curso Académico 2011/2012

Trabajo de Fin de Máster

Intensive Metrics for the Study of the Evolution of Open Source Projects

Author:
Santiago Juan
Gala-Pérez

Supervisor:
Dr. Jesús
González-Barahona

September 14, 2012

"There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle." (Albert Einstein)

Dedication

To Santiago Gala Velasco (1928-2011) *In memoriam*
To Elena Pérez Salamanca.

Acknowledgements

I'm grateful to my supervisor for being extra patient, to the personnel and parrochians of *El Rincón Guay* that saw me work during hours and were always helpful and supportive and to Ana, from *Aguardiente* for her gentle and discreet support.

Contents

1 Summary

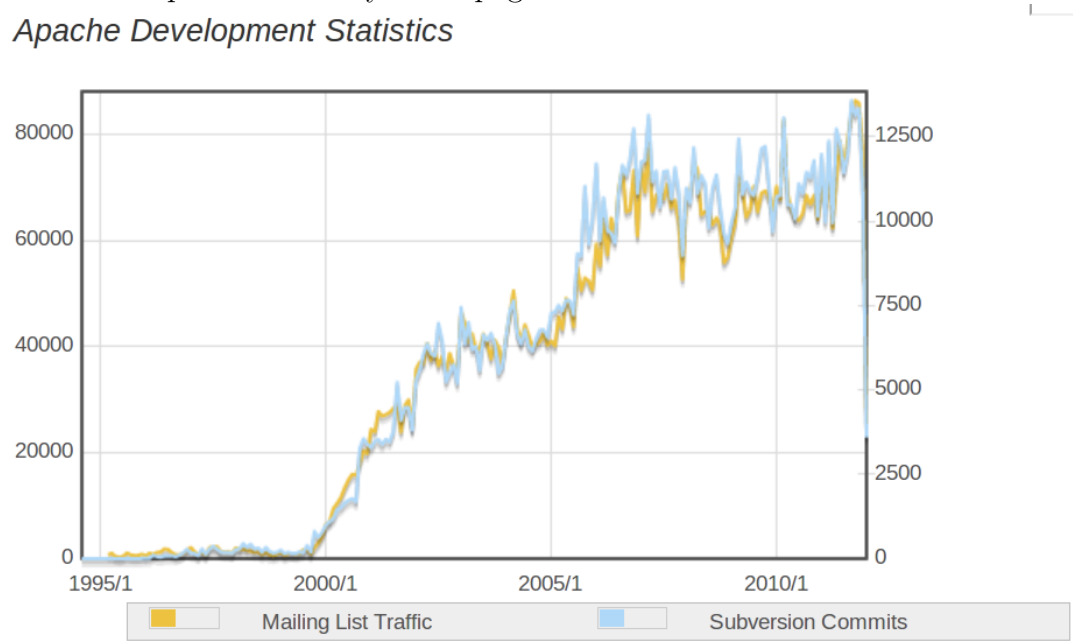
Starting with the empirical evidence that the ratio of email messages received to commits has remained relatively constant along the history of the Apache Software Foundation, we look for the use of such a metric to compare projects irrespective of their size or growth. The behaviour of this ratio, and also of similar metrics like the ratio of developer messages or issue tracker related messages to commits, is studied for several projects. The ratios are independent of the size of the project, in terms of rate of messages, commits, active developer number, etc. and remains relatively independent of the technology or functional area of the project. It looks related to the technical effervescence or popularity of the projects.

2 Introduction

On Nov 29, 2008, just a couple of months since the author started studying the *Master de Sistemas Informáticos y Telemáticos*, Joe Schaeffer, ASF member and contractor for infrastructure work sent an email to the private discussion list members@apache.org , which said, under the provocative Subject "*Community over Code debunked*": [1]

It's a bit premature to start throwing this url around,
since Paul is still working out what it should actually
do, (and it looks like IE isn't supported yet). But
it's already revealed something IMO interesting about
the ASF: the fact that the number of commits and the number
of mailing list posts have grown in linear relationship
to one another over the years. [...]
Here's the url for firefox fans: http://www.apache.org/dev/stats/
Note how well the charts for total commits and total emails align
even though they're graphed against 2 different scales.

A recent snapshot of this dynamic page looks like this:



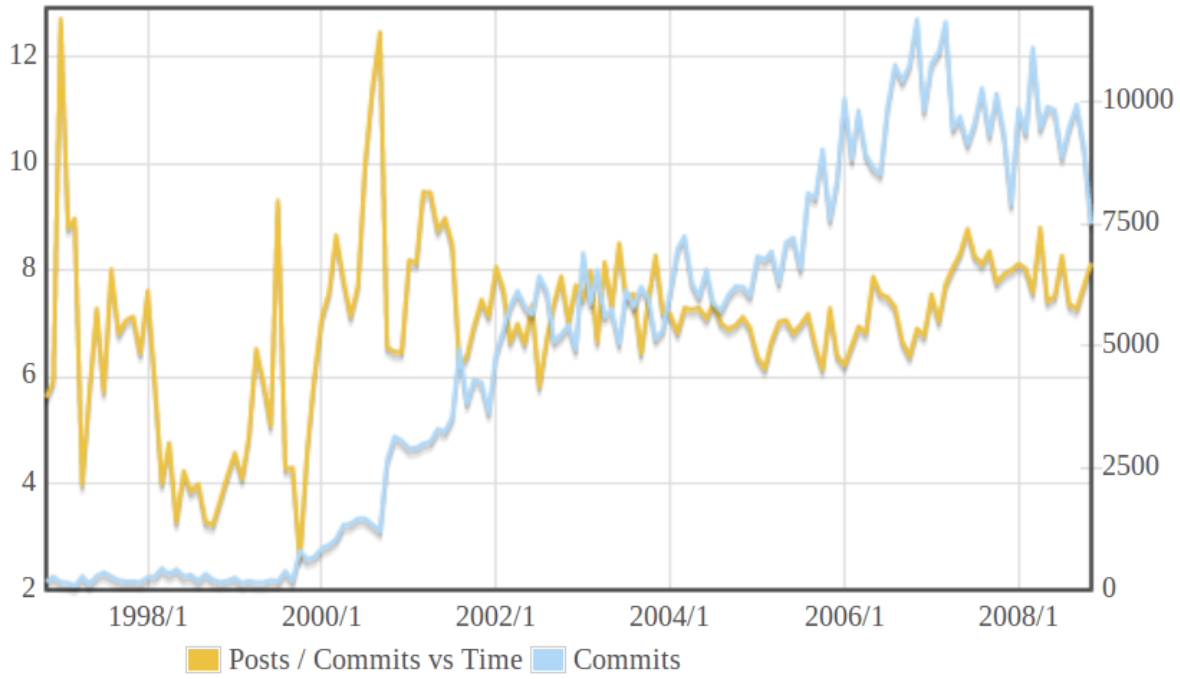
In the subsequent thread there was discussion on the importance and meaning of the result. Quantifying the results, Joe said:

you can get an idea of how significant those emails are to the total by comparing the two scales. Looks like about 1 commit email for every 7-8 non-commit emails.

The author sent an email, contributing to this thread, which essentially said:

I would only expect mail traffic growing faster than commits during "crisis", i.e. moments where the code base needs to fork after a tough technical decisions, and those are the points where our mantra would be most useful as a decision guide. In "stationary" state, not much list traffic is needed while bugs are fixed or featured finished.

Joe published a different graphic illustrating how scale invariant, was this metric globally: http://people.apache.org/ joes/invariant.html . On it one can see that globally, as an organization, the ASF produces a relatively constant ratio of emails versus code changes, even while the number of commits has grown more than one order of magnitude from 2000 to 2008.



The ratio fluctuates somewhat between the beginning of the Apache project, back in 1996, and 2002. There is missing data, on one side: most of jakarta email traffic was hosted outside of the ASF at the time, and a hard disk crash lost quite a few messages. Also, the ASF was much smaller, thus more sensitive to noise.

The aim of this report is to explore this ratio at the project or related group of projects level. As we split the flows into projects, we will found that the result is very noisy below a size/activity threshold. It is also interesting to try to get an idea of how the interplay of a growing mixture young, mature and stagnant projects happens to produce such a remarkably constant ratio.

3 Objectives

3.1 Definitions

Dissipative System:

"In summary, we have found that the distance from equilibrium and the nonlinearity may both be sources of order capable of driving the system to an ordered configuration. A highly nontrivial connection between order, stability and dissipation appears. To indicate clearly this relation we call the ordered configurations that appear beyond instability of the thermodynamic branch the dissipative structures*."* [PRI2], p60
Prigogine calls dissipative structures to the open systems showing self-organization thought this connection between order, stability and dissipation that happens far from equilibrium and/or in the presence of highly non-linear coupling.

Ecosystem:

We talk about ecosystem to describe a closely coupled group of projects, that typically consist in an original project that sprout children as it grew, or projects that depend on a common platform. Examples:

- Sling, uses jackrabbit and felix. Even if jackrabbit is a reference implementation of JSR170 and Felix is a OSGI Service Platform, all three were donated by Day Software to the Apache Software Foundation and share committers.
- httpd created apr as its runtime, and they remain related.

Intensive Properties:

From wikipedia: *"In the physical sciences, an intensive property (also called a bulk property, intensive quantity, or intensive variable), is a physical property of a system that does not depend on the system size or the amount of material in the system: it is scale invariant."* The scale invariance allow them to stay constant as a project grows, and also make them useful to compare projects of different sizes. The article continues: *"By contrast, an extensive property (also extensive quantity, extensive variable, or extensive*

parameter) is one that is additive for independent, noninteracting subsystems. It is directly proportional to the amount of material in the system."

Information Exchange:

We use the term "information exchange" for any socially atomic public interaction between one actor and an open source project. It can be something like a commit, an email, an edition of a wiki page, etc. We don't take into account the information value (or entropy) of the exchange.

Open Source

One of the ways to call Free Software. See "The Open Source Initiative (OSI, http://opensource.org/)

Self-organization:

In the framework of the dissipative systems theory Prigogine shows how a system shows organized behavior at the macroscopic level. In [PRI2], p4, one can read:
"The essential point is that beyond the instability of the thermodynamic branch we may have a new type of organization relating the coherent space-time behavior to the dynamical process (e.g. chemical kinetics and convection) inside the system. Only if appropriate feedback conditions are satisfied can the thermodynamic branch become unstable at a sufficient distance from equilibrium. The new structures that appear in this way are radically different from the 'equilibrium structures' studied in classical thermodynamics, such as crystals or liquids. They can be maintained in far-from-equilibrium conditions only through a sufficient flow of energy and matter. An appropriate illustration would be a town that can only survive as long as it is a center of inflow of food, fuel, and other commodities and sends out products and wastes."

FLOSS

Free/Libre/Open Source Software. The wikipedia defines FLOSS as: *"Free and open-source software (F/OSS, FOSS) or free/libre/open-source software (FLOSS) is software that is both free software and open source."* In this work we will mostly use Open Source, as it is the common idiom internally in the Apache Software Foundation.

Software Entropy:

From wikipedia :

The second law of thermodynamics, in principle, states that a closed system's disorder cannot be reduced, it can only remain unchanged or increase. A measure of this disorder is entropy. This law also seems plausible for software systems; as a system is modified, its disorder, or entropy, always increases. This is known as software entropy
Within software development, there are similar theories; see Lehman (1985)[2], who suggested a number of laws, of which two were, basically, as follows:

- A computer program that is used will be modified
- When a program is modified, its complexity will increase, provided that one does not actively work against this.

From [HAR1]: [A]n entire alphabet of symbols: $s_1, ..., s_q$ would on the average provide
$H = -\sum_{i=1}^{q} p_y \log_2 p_i$
bits of information per symbol. This quantity is called the *entropy of the information source*, or *language entropy.*

Stigmergy:

A collective process is stigmergic if the work (ergon) done by one agent provides a stimulus (stigma) that entices other agents to continue the job. The concept appeared originally in the context of social insect behavior. Note that, as Joichi Ito used to say in his conferences around 2003: *"One of the things I've found is that people don't like to be compared to ants."* Still, a big number of the non linear characteristics of the Open Source Projects come from stigmergic processes:

- The cost of distributing an email to thousands of developers is constant
- search engines bring people looking for solutions to problems to the archived discussions about those very problems
- asking a question *the right way* will entice readers to help solve it.

3.2 Problem Description

This work aims to study how to find potential metrics that describe the state and the evolution of Software Development projects , having the property that the metrics should not depend on the project size, i.e. be an intensive property. We aim specifically towards projects that are developed with the Open Source typical approach, for which most if not all of the information exchanges take place over channels that can be audited, logged and indexed for search and reference, like email, bug trackers, code revision tools, etc.

In particular we will concentrate, for the purpose of the current work, in the study of the ratio between total email traffic and number of source code commits for a selection of projects in the Apache Software Foundation.
The projects range in size from the 11500 emails/month that Apache **Hadoop** generated in Aug 2012, to a project like **Turbine**, which was relatively big back in 2000, but has generated a maximum of 175 messages per month during the last 5 years. **httpd**, spanning 17 years (1995-2012) but still healthy and being actively developed, has generated a maximum of 5200 messages in Apr 2002, but it is still around 1000 messages/month currently. **Beehive**, the first project that got retired into the Attic, where Apache projects get archived when stale, is a good example of what happens to a project that gets abandoned by developers.
This metric allows us to compare easily the evolution of the named projects, spanning two orders of magnitude sizes. It is also suitable to assess how exciting the project is for the outside community of users or developers, and also to see how the community matures into a stable development site.

3.3 Assumptions

There are a number of assumptions that guide the experimental approach we have taken in this work.

- the information content of each *information exchange* (defined as a socially atomic interaction between one actor and an open source project) is roughly comparable.
- the source code repository activity (number of commits) measures the self organisation of the project,
- all *information exchanges* are reflected by an email sent to one of the project lists

We started working looking for the validity of this thesis: *The ratio between information exchanges and progress of an Open Source project, i.e. between total emails and number of commits, is a relatively stable property, that remains constant for a stable ecosystem and varies slowly with time for the components of the ecosystem.*

3.4 Alternative Approaches

There are a number of alternate approaches to measuring progress of a project using metrics that are **intensive**, i.e. independent of project size. We discuss briefly a number of approaches that are similar to our concept in the following sections, to finish with our approach, that we call *Information Exchanges*.

3.4.1 Information Entropy

[RON2] characterizes "The web community" as a dissipative structure, and advocates the use of information entropy for the study of fluctuations as some contributors have a disproportionate contribution, thus using software entropy as a measurement of how far from equilibrium is a community. Like [RON1], while the approach seems to be oriented to the quantification and measure of the entropy of exchanges between parts of a system, and between the system and the environment, no effort is done to give quantitative data. The authors practical conclusions are limited to advocating the use of the software entropy formula:
$H_{max} = H(\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n}) = \log_2 n$
which expresses the maximum information entropy in a situation where n participants contribute each with a symmetrical 1/n fraction of the total contributions. or, in terms of agent "*score*" M_i (measuring the percentage of contribution in a given community), they use a *normalized* entropy:
$S = \frac{-\sum_{i=1}^{n} \frac{M_i}{\sum_{j=1}^{n} M_j} \log_2 \frac{M_i}{\sum_{j=1}^{n} M_j}}{\log_2 n}$
The main conclusion in both papers is that in order to minimize risk or maximize stability, it is important to maximize the entropy production, i.e., to balance the number and relevance of core contributors, avoiding situations of excessive dependence in one developer.
If we compare with the ratio information exchanges/commits, we find that, in the limit situation where a isolated developer develops a project, this ratio would be one: a commit per commit and no extra traffic. In a situation where two developers collaborate to develop a project, a number of *extra* messages will be exchanged, making this ratio greater than one... In the limit, a number of coordination messages will be exchanged per commit. In situations where the number of coordinating parties is high (for instance new users or developers approaching a project or one developer proposing a novel architecture for the next release) the ratio will go up.

3.4.2 Stigmergy

From studies on social insects Grassé (1959) introduced the term stigmergy. Stigmergy is derived from two Greek words: stigma (meaning sign) and ergon (meaning work). A process is stigmergic if the work (ergon) done by one agent provides a stimulus (stigma) that entices other agents to continue the job. Stigmergy has become popular for Artificial Intelligence research (Theraulaz and Bonabeau 1999). Heylighen (2006b) studies the stigmergic aspect of information system, like Wikipedia and FOSS development. [KIE1]
We acknowledge the importance of stigmergy for Open Source. At the end of the XXth Century, say 1998, it was apparent that trying to get technical information about Mi-

crosoft tools and Operating System was difficult and clumsy, as The MS Knowledge Base was expensive, a big set of CDs, and it required a license server. At the time, both the Apache **httpd** and the Sun Java Web Server (that later became Apache **Tomcat**) were serving the whole documentation with the default install. So, for every developer that started looking into it and using *Tomcat* or *Apache*, it was easier to find information on it.
There are also stigmergic aspects with the big public mail archives that get indexed and appear frequently when users ask for common questions. So, the auditability of all the design and engineering discussion is positive to encourage future collaboration, and this collaboration makes the documentation of the project more visible, in a strong feedback loop.
Stigmergy brings positive feedback, and thus non-linear behavior, something that Prigogine quotes as something needed for out of equilibrium dissipative systems: *"More precisely autocatalytic steps are necessary (but not sufficient) conditions for the breakdown of the stability of the thermodynamical branch"*. [PRI1, page 271])
We are not studying stigmergic aspects in the scope of this work; we focus instead in the raw information exchanges.

3.5 Our approach: Information Exchanges

We call our approach Information Exchanges. We are measuring "communication acts" between developers, users, maintainers, i.e. all roles in a project, and try to relate them with the self organisation of a project via changes in its code base. When we speak about code base we typically include documentation, as the documentation is usually under the same source repository as the code.
One could think that, to measure communication acts, it has to be looked for at a lot of separate places. But, in general, most places where information is exchanged send an email to public lists for each such exchange.
First, one has to take into account the fact that all decision making in Apache should happen through mail lists. This is a strong cultural trait of the ASF. See, for instance how Justin Erenkrantz presentation in OSBC [ERE1]: *"All technical decisions about a project are made in public (mailing lists)"*. There is also a famous Roy Fielding's mantra, quoted later, in the same presentation, in the slide "Where decisions happen": *"If it didn't happen on-list, it didn't happen."*
The fact that mail lists are used as central coordination points of the projects has another advantage for our purpose: enforce that all interactions with the system appear on list:

- Issue trackers will send an email to some configured lists for every issue opened, commented, modified or closed

- Ditto for code review tools, automated builds, specially when the build gets broken, or wiki page edition

- We are, in fact, studying commits via the email that all SCM systems send to the project lists with the changes that got pushed. This works well, because the typical way to discuss a commit or object to it is to reply to the email reflecting the code change.

The study of emails sent to the project looks appropriate if one wants to check the information flows in an open source project, at least for an Apache one.
Still, one might miss a number of different information flows in a number of cases:

- In projects dominated by a company with internal work-flows, which *dumps* code periodically to an open source projects,lots of information gets missed; the android open source (AOSP) comes to mind

- In projects dominated by a company, even when their committers interact through open lists, they could be interacting a lot privately. Say, they can process bugs on the proprietary derivative project, to end up committing a solution to the open source project with most of the information remaining invisible to our analysis.

- In projects making extensive use of videoconferencing or IRC channels. A good example of this behavior is Cyanogenmod. They mostly hang on IRC channels, and tend to meet weekly by web video conference. They also use forums, which don't send email. In those projects information flows are either not archived or not present in email. This is not the case for the ASF. In order to study such projects, different analysis techniques will have to be developed, for instance parsing IRC log activity or Forum contents.

- In the press, in documentation deployed in downloaded copies, in downstream projects redistributing it, for instance linux distributions that package Apache httpd server and have support forums, etc.

3.5.1 Measuring Information Exchange more precisely

Even if we agree that analysing the relation between email flows and code changes during a project evolution is a good way to characterise the properties and evolution of the project, there are a lot of ways to measure such information flows:

- One could try to find the information delivered by the message by, say, compressing the message body to remove redundancy.

- The uncompressed size of the email could be used.

Those two approaches would try to differentiate from, say, one line changes that fix a typo and long, meaningful features pushed into the code. The same for emails: it is not the same to send an email saying "+1", and thus signifying support for a previous position, that to elaborate a long response...
A third approach was suggested by Stefano Mazzocchi during the mail thread that was described in the introduction of this report. He said that we should not compare the productivity (in commits) with the number of emails processed, but rather with the number of emails that had at least one reply. his thesis is that doing so will screen very effectively those machine generated events from human ones. [MAZ2]
Also, Walter Harrison in [HAR1] proposes to measure the complexity or a program by measure its distribution of *operators*, i.e. special symbois, reserved words or function calls. His metrics, called *Average Information Content Classification*, shows the property that it is independent of program size, and can be used to assess the complexity of fragments of a program.
It would be tempting, for our study, to compute the complexity of each contribution to a software project, and thus compute our ratios in terms of total bits contributed/bits committed to the repository. This approach will be further investigated, but we felt that the contributions will have a similar average information content, and thus it would have complicated our computations without a big change in results.

3.6 Methodology

Studying the communication flows in a project is tricky. There is a big number of different channels used to communicate both between the different people, with different roles, that work in a project on any given moments. Some decision were taken, both to simplify the analysis and because the looked natural in the context of open source projects:

- Limit our study to projects in the Apache Software Foundation. The author has intimate knowledge of the Foundation, having been a committer there since 2000, a member since 2004 and VP Apache Portals from 2004-2006. Also, the ASF has cultural traits that help in our purpose (next bullet).

- Use email as information source. While Open Source projects have lots of information flows, and some are difficult to measure, the Apache Projects have some strong rules [ASF1] that facilitate our work. Those rules refer to the email lists as places where decisions happen. Typically all information flows to the project

are documented. Discussions are, obviously, but also wiki editions, commits, automated build failures or comments in bug trackers. Even IRC discussions have logs typically pasted to the developer list

  - As Justin Erenkrantz says in a Presentation about "The Apache Way" at OSCON [ERE1]:
    - there are many forms of contributions
      - evangelism, bug reports, testing, documentation, code, design feedback
  - On the other side, he continues:
    - "First time meet face-to-face is at events
      - Mailing lists are the pulse of the project
      - IRC, AIM, Jabber, etc. not for decisions
      - Roy's mantra: "If it doesn't happen on-list, it didn't happen."

- The unit we will consider initially as "communication act" or "information exchange" is the email. An email correspond to a decision to interact with the project, be it by modifying a wiki page, by asking a question, reporting a bug, making a single commit, voting, etc. While the weight of those acts in terms of social impact or information entropy varies a lot, they still look as reasonable starting points for our analysis.

- We look for intensive properties of the projects. Most metrics of projects are difficult to compare since they are not adimensional and thus they depend on the size of the project. This is additionally complicated by the fact that successful projects commonly undergo transitions in size of two orders of magnitude during their developmentt. We are looking for properties that vary with the maturity, technology or community composition of the project, but not, or at least not much, with its size.

A caveat about our chosen methodology is due: For projects implementing standards, such as **Jackrabbit**, **Felix**, **Tomcat**, even **httpd**, it might happen that the relevant standards lists has activity, such as proposals for the revision of the standard, feed-back, errata, etc. that influence directly the development of the given project. We are, thus, losing some of the information exchanges related with the project.

4 Case Studies

We will to a summary study of a number of projects in the Apache Software Foundation, comparing how they stand in terms of their ratio of messages to commits.

4.1 Selection of ASF Projects

We take an exploratory look to a group of projects in the Apache that are relatively different, in terms of technology, age, size, etc. In particular, we want to explore the set of ASF projects without being forced to study all of them. In the following list, we explain the reasons that lead to selection of each project. We select:

- Apache **httpd** server, the project that originated the ASF as such. External CVS was used until 1996, so commits data starts then. It has a 15 years history, and has passed through one major release 1.3 -> 2.0 and two minor ones: 2.0 -> 2.2 and 2.2 -> 2.4. Also, we plot one of its subprojects: the **apr** (Apache portable runtime) library used to ease cross platform development

- Apache **turbine**, an early web framework in java, part of Apache Jakarta. it is a project that was very active around 2000, but got stagnant due to endless refactoring that broke compatibility and focus of the community in alternate approaches. It is a good example of the scale invariance of our ratio

- Apache **tomcat**. Reference implementation of the it was donated by Sun Microsystems after some meetings in the Jakarta room, which started the Jakarta project. Later it migrated to its own PMC.

- Apache **lucene**, the full text indexing and search solution. It is showing a lot of activity in the last years, because of the trend to so called *Big Data* processing. A number of the main

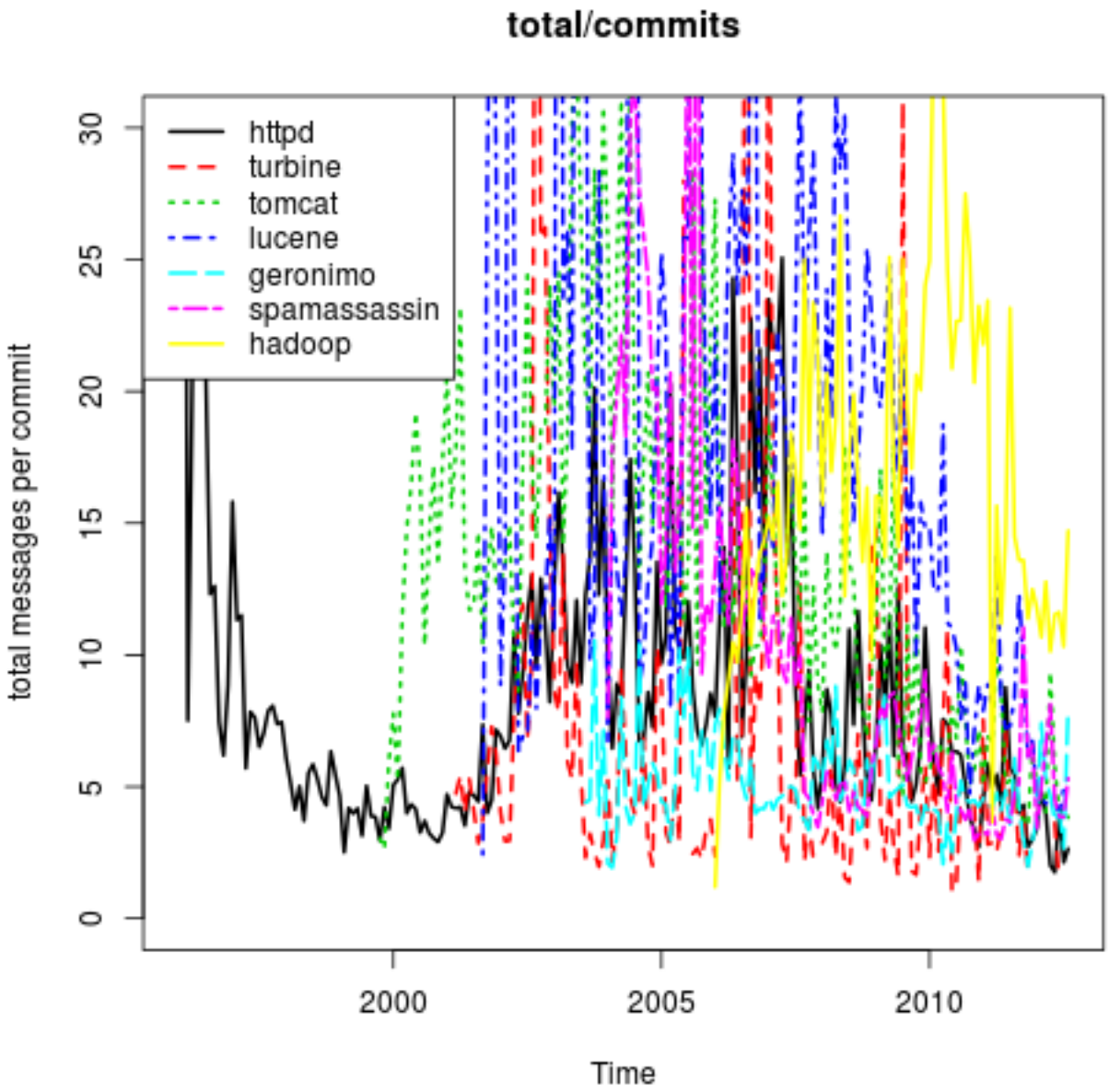
- Apache **geronimo**, a certified JavaEE Application Server, that integrates a number of external and internal projects

- Apache **spamassassin**. Written in perl, it is a preexisting community that joined the ASF in 2004. While most java developers are involved in several projects, and

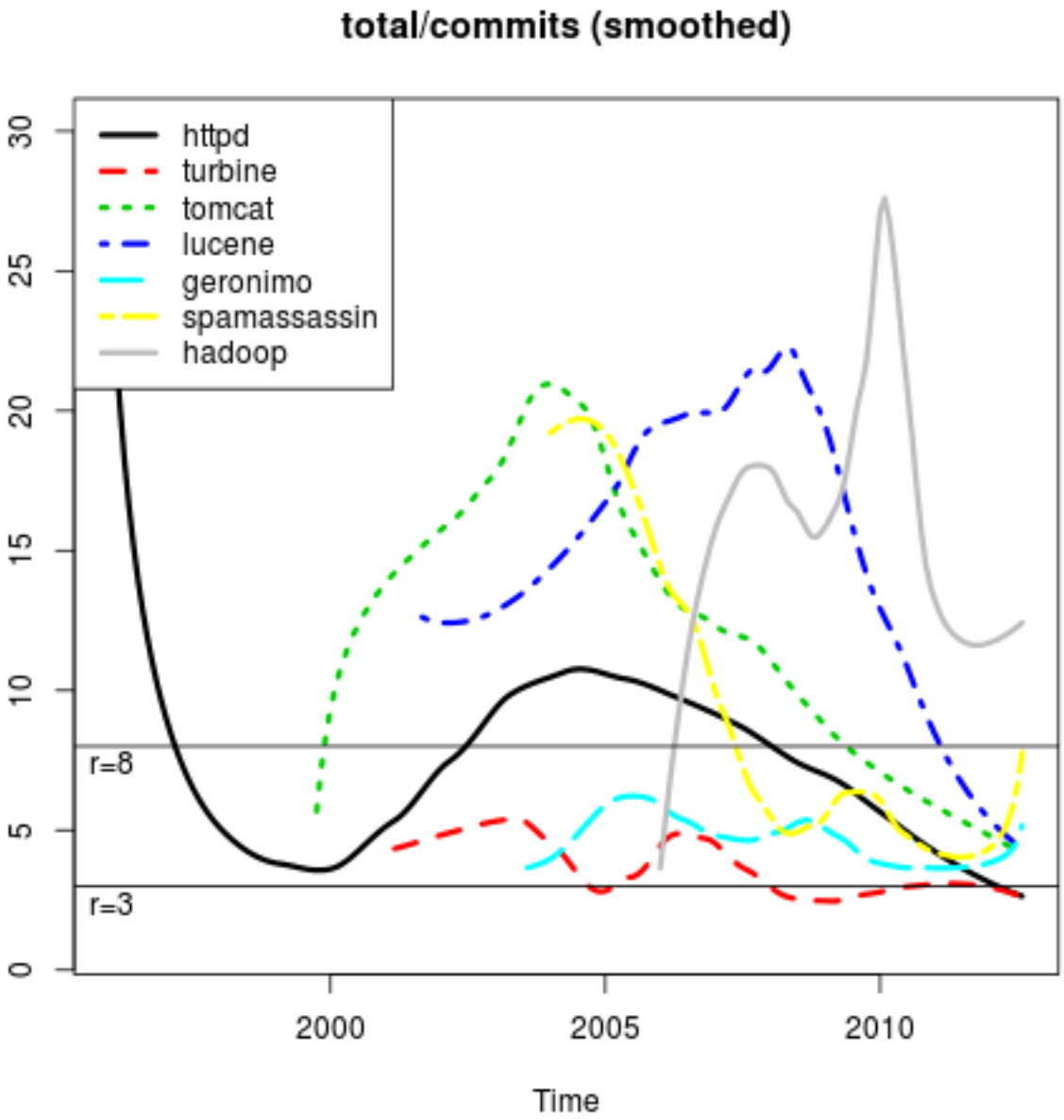
some of them even in the **httpd**, **apr** or other areas, **spamassassin** developers were not so much involved with the rest of the ASF in the initial phases of the project. Spamassassin is fairly mature and is fairly used in production.

- Apache **hadoop** ecosystem. Hadoop is a *Big Data* framework. Currently is by far the most popular Apache project, with high levels of traffic in its ecosystem. There are a lot of projects spawning off hadoop: **HBase**, **Avro**, **Hive**, **Pig**, **Zookeeper**, **Whirr**,... Also, it is very active right now, correspondingly with the trend to apply Big Data techniques everywhere.

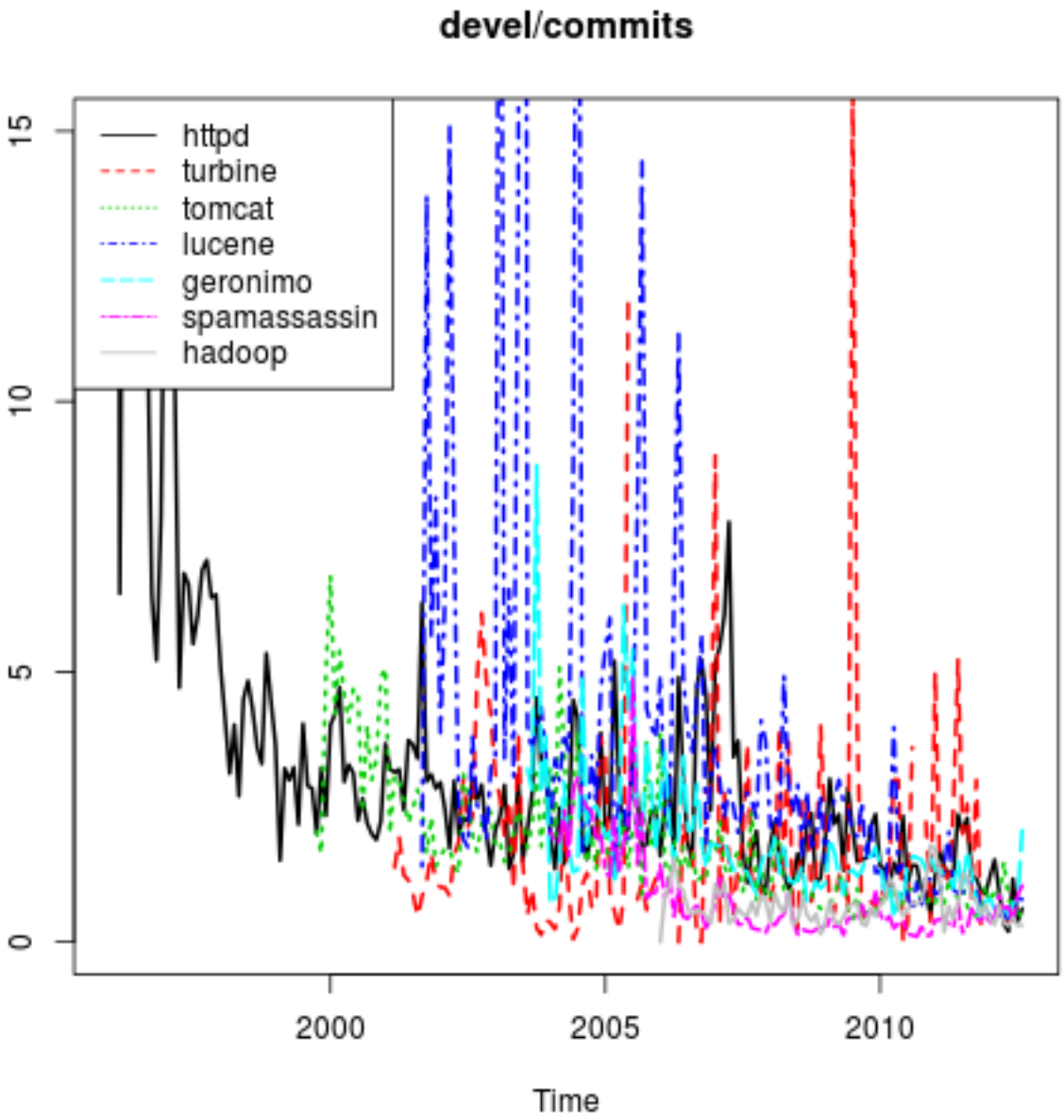
- Apache **jackrabbit**/**sling** ecosystem. **Jackrabbit** is the reference implementation of JSR-170, Java Content Repository, a standard for managing web content. This JSR was pushed forward mostly by Day Software, a company that was bought by Adobe Systems in 2011. Day Software donated also two other components to Apache, that were used in their commercial product CQ. So, Day has opensourced their OSGi service platform, **felix**, and a web framework, **sling**, that both use **jackrabbit**. The three projects are closely related, and also related to a company that started them.



The previous figure shows a line plot of the time series of a set of projects, each using a different line type and color. Noise and data spikes makes it difficult to read.
The next one shows the same data after smoothing of the total and commits series. If one examines the noisy projects, they belong to two different sets: **turbine** and **beehive** are noisy because the number of commits get relatively small and reactive to bug reports or dependency breakage as the projects are stagnating; on the other side, **lucene** is strong and growing, but had a highly fluctuating devel list traffic pattern.

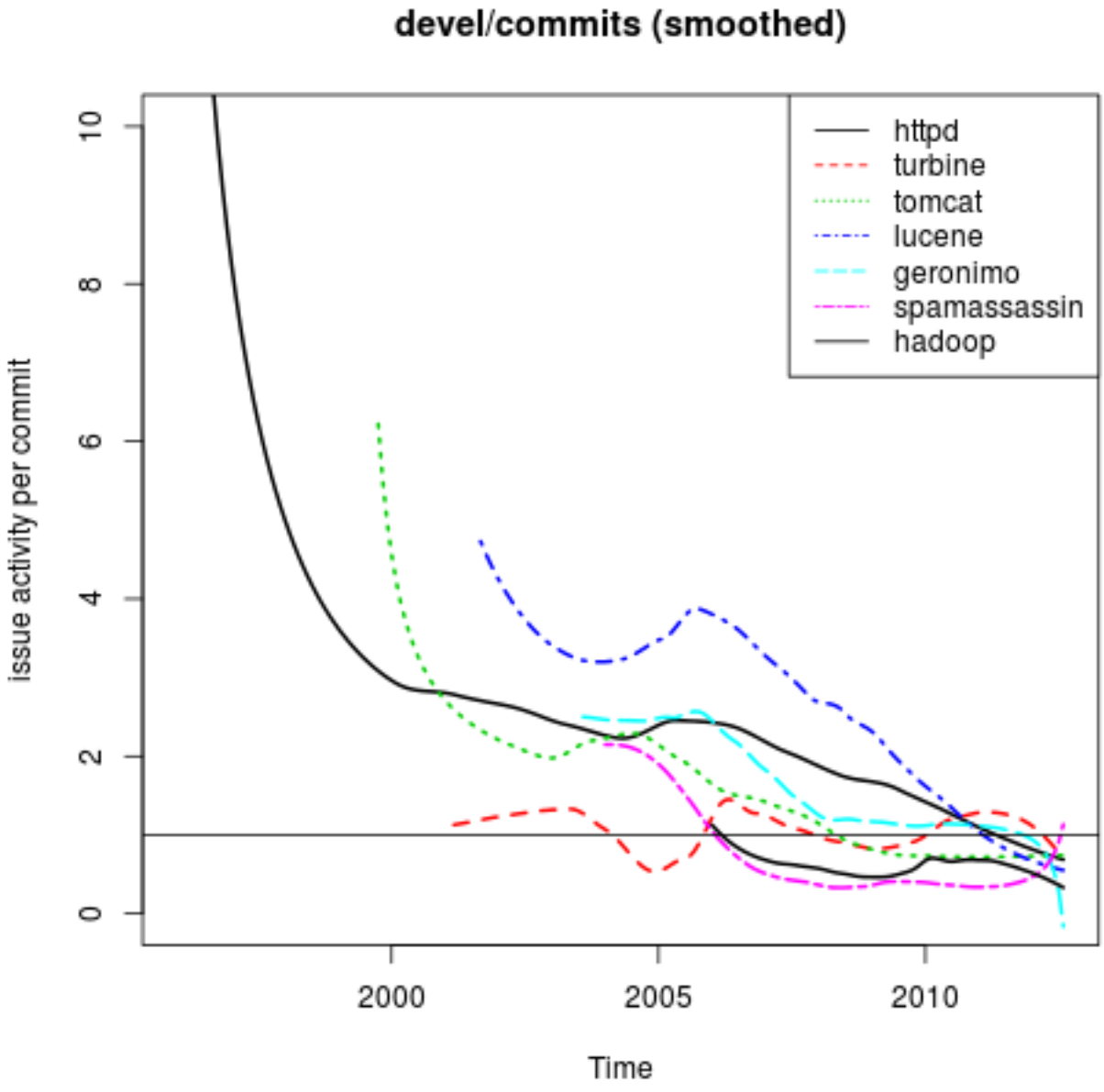


We can limit our comparison to the developer list. While the total message traffic includes bug reports, user list question, and other sources of messages that are typically not originated inside the development team, devel traffic is typically either by internal developers or people trying to join or just modifying the code for their use. Also, while total traffic typically goes up when a project is used or adopted a lot, developer traffic goes typically up before major releases or refactoring, and when there is brainstorming or discussions about design decisions.



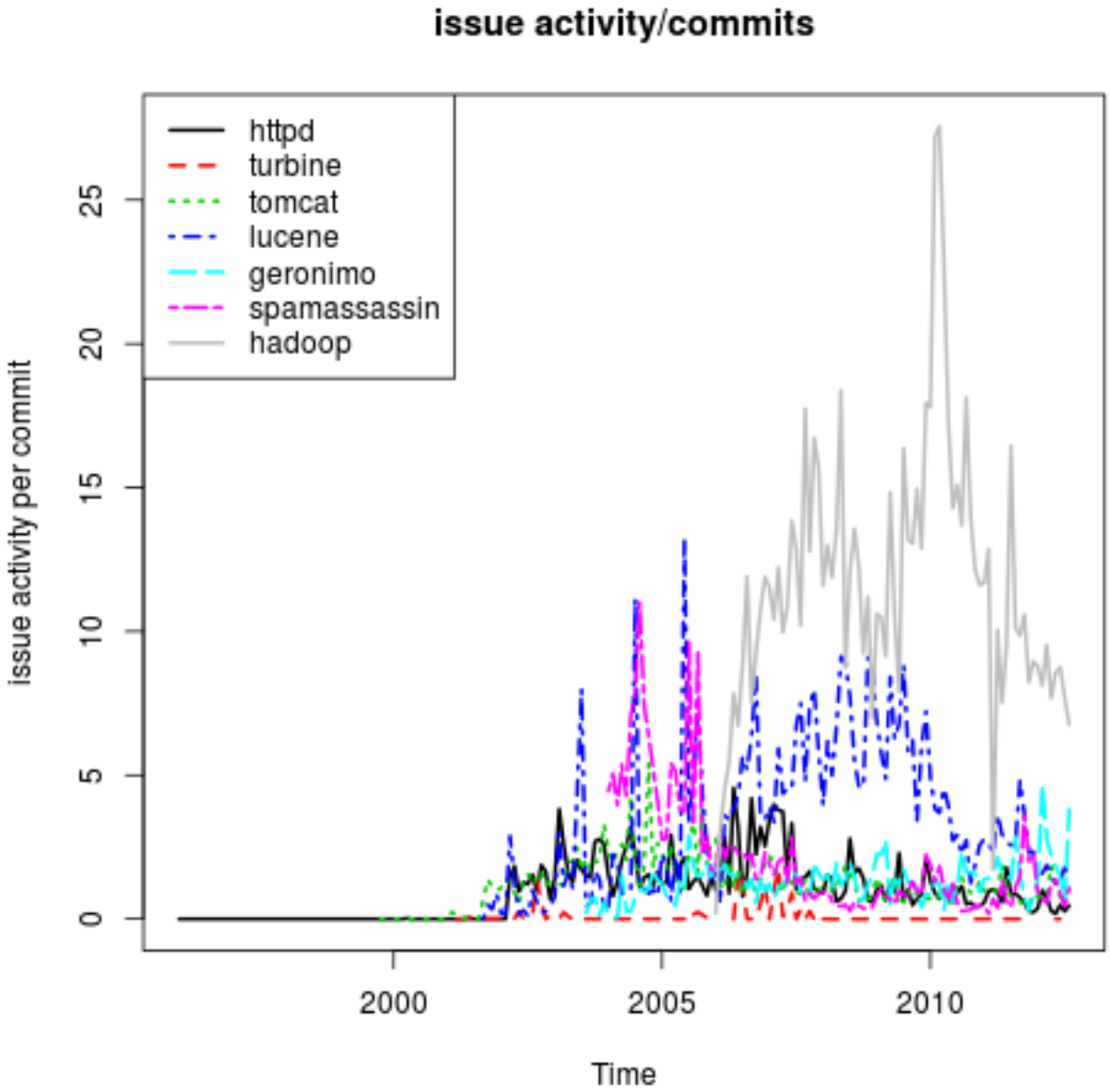
When a project gets mature and development limits to bug fixing and straightforward development, the need for coordination falls down, and one see a pattern where less messages are sent to devel list than commits are checked in. This is a recent trend for most of the projects we have been analyzing in this sample. Even when, like **hadoop**, the total/commit ratio is very high, the devel/commit ratio is below one, meaning that developers either don't need to talk or the just "*talk with patches*", as it is often said. This could be explained also by the introduction of distributed source code management tools, such as git, that make easier to work with a smaller granularity of commits, which means that the number of commits grows, but its information content per commit gets

smaller at the same time. This is a hypothesis that could be tested by computing the size of commits versus time. An alternative hypothesis, that use of code review tools again diminished the granularity of commits, is less likely. It could also be tested by the same test, plus the analysis of commits coming from code reviews of patches versus *normal* commits.

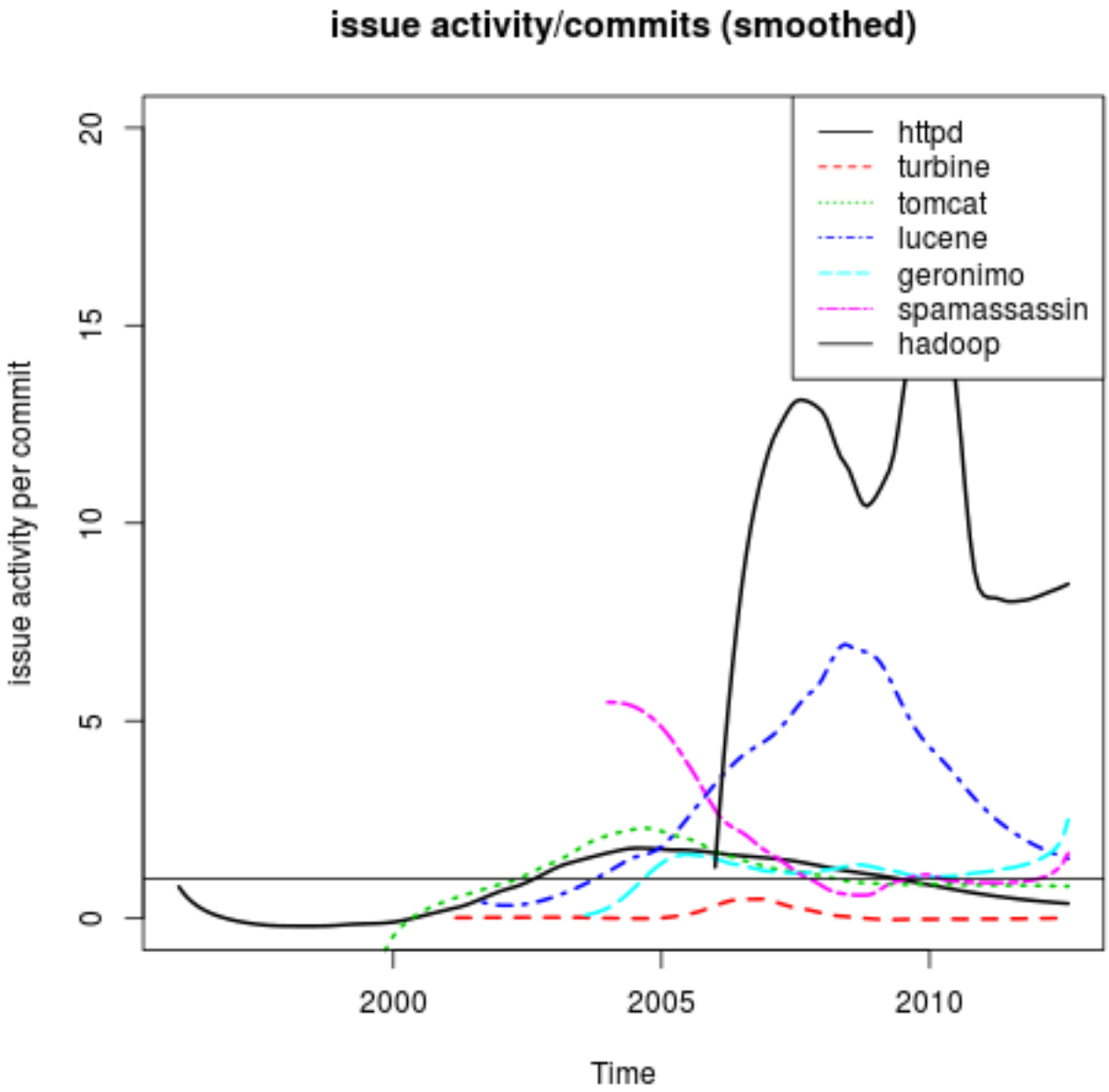


The issue related activity, in particular, does not show the same convergence with maturity than the previous metrics when compared with number of commits. It gets higher and higher as people is encouraged to report bugs and interact with the project via the issues tracker, and also when a project development is user-drive rather than developer

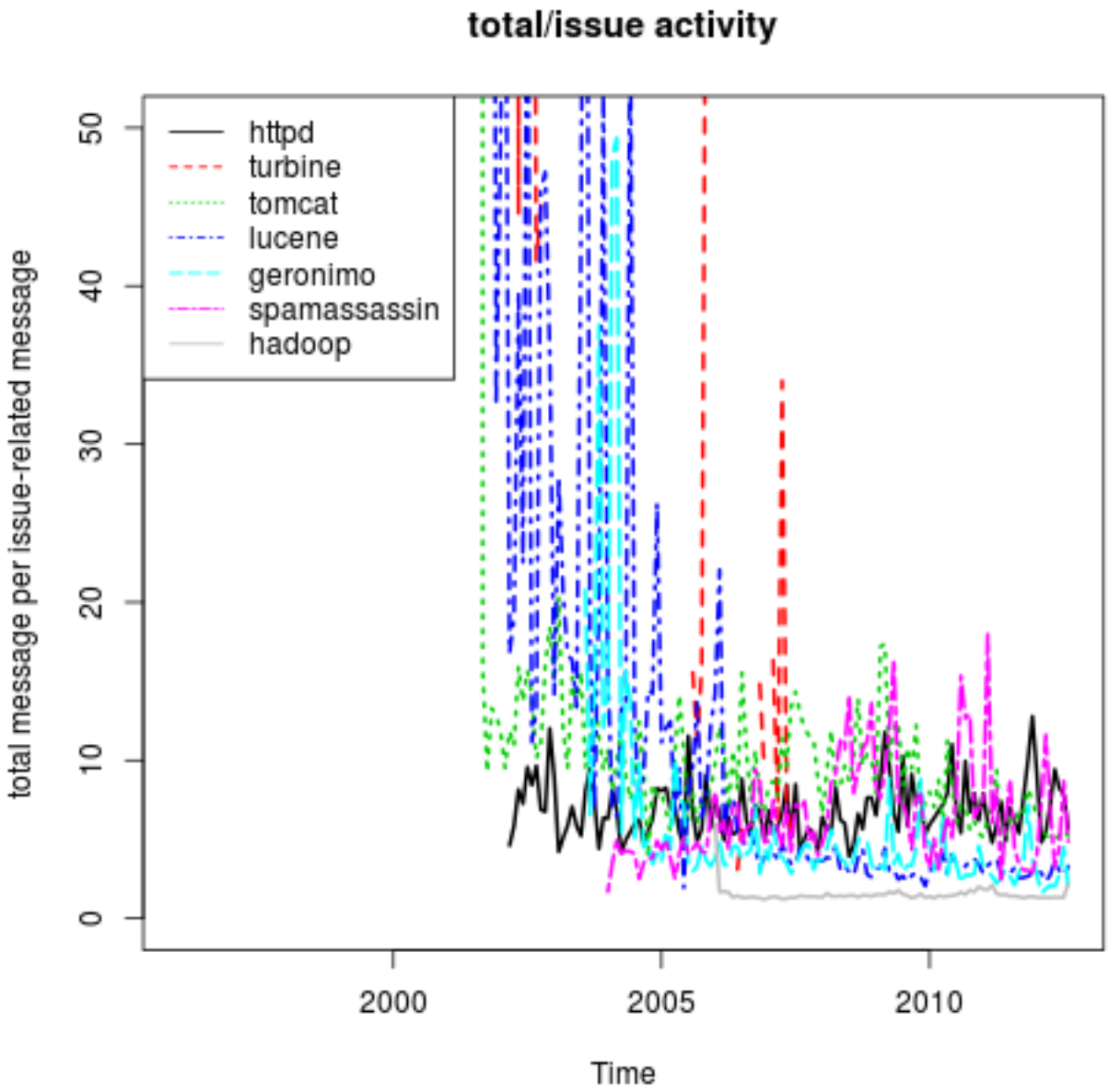
driven.



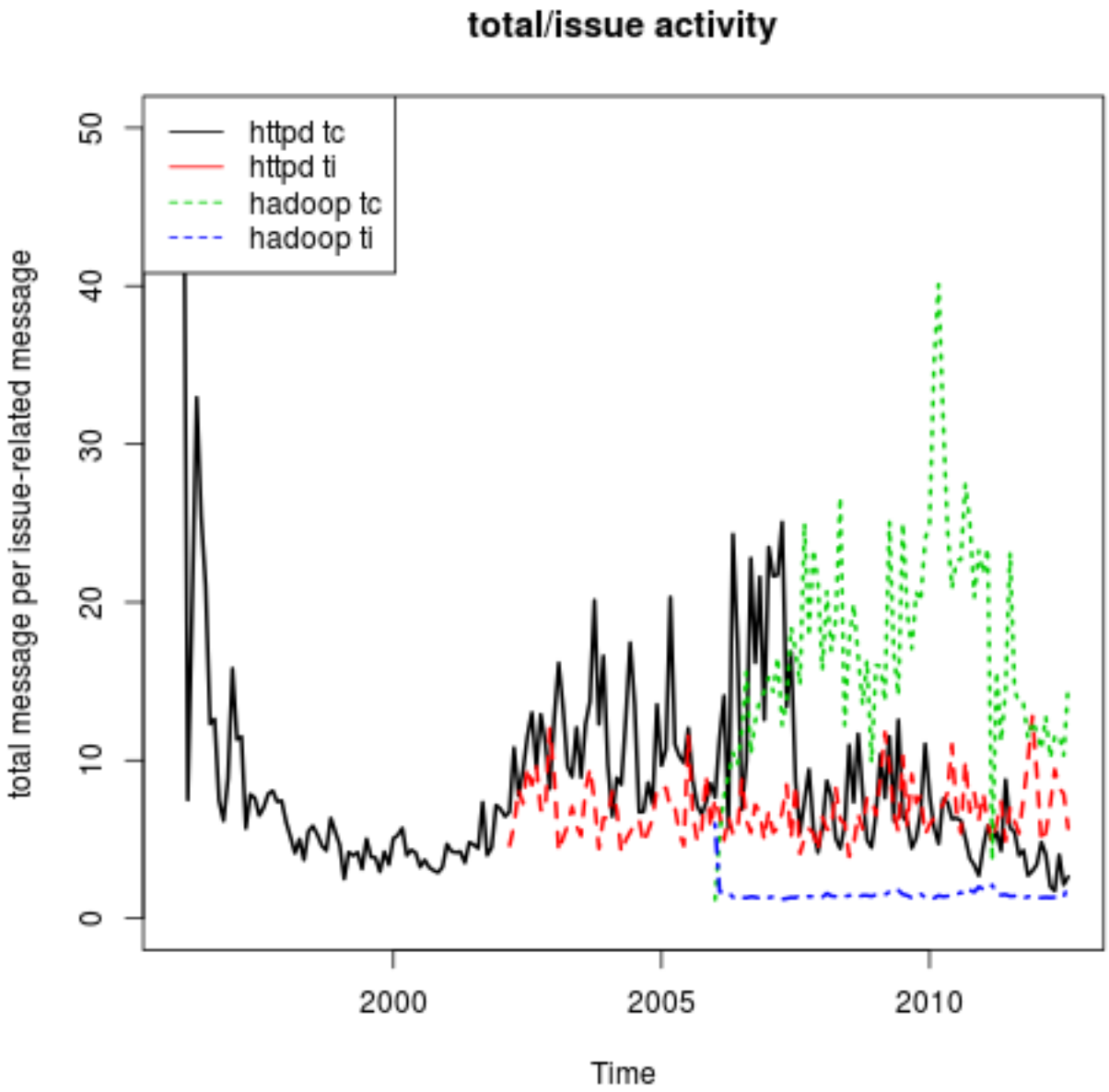
If we see it smoothed, we find that hadoop has very big peaks of activity, and other projects vary in their use of issue trackers.



What turns out to be interesting is that the issue-related activity correlates with the total project activity in a similar way as with the commits. It looks like mature projects have a similar trend to get issue-related activity 10-15% of total activity. The hadoop ecosystem, though, shows a much lower total/issue ratio, thus showing that the amount of issue activity for them is a much higher percentage of the total ratio



We plot in the next figure the total messages/ commit and total messages/issue for two projects like httpd and hadoop. We see that hadoop uses extensively the issue tracker (issue messages are more than half of their traffic), while httpd is more focused and developer oriented, with more balanced traffic distribution.

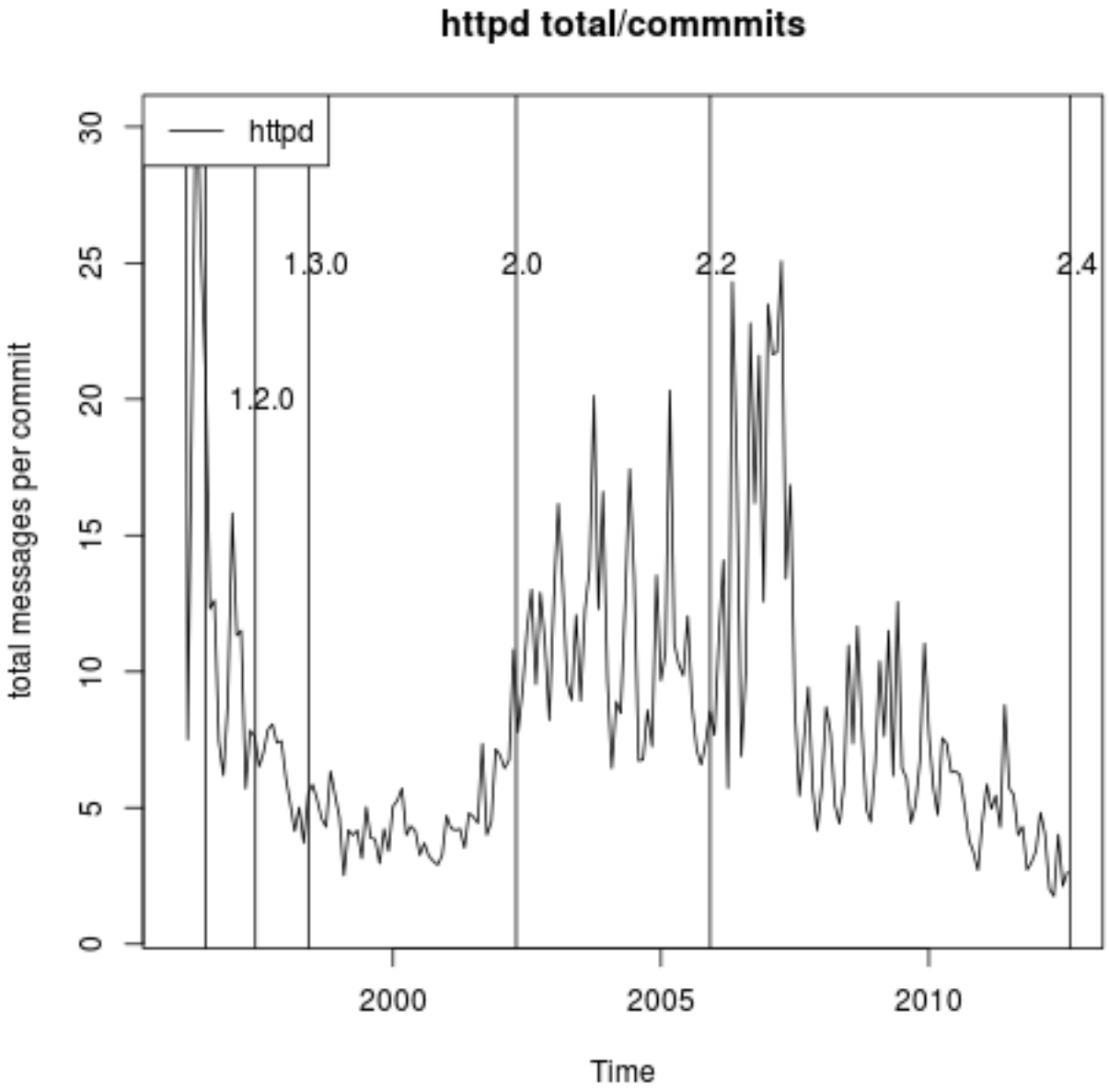


4.2 Apache Web Server

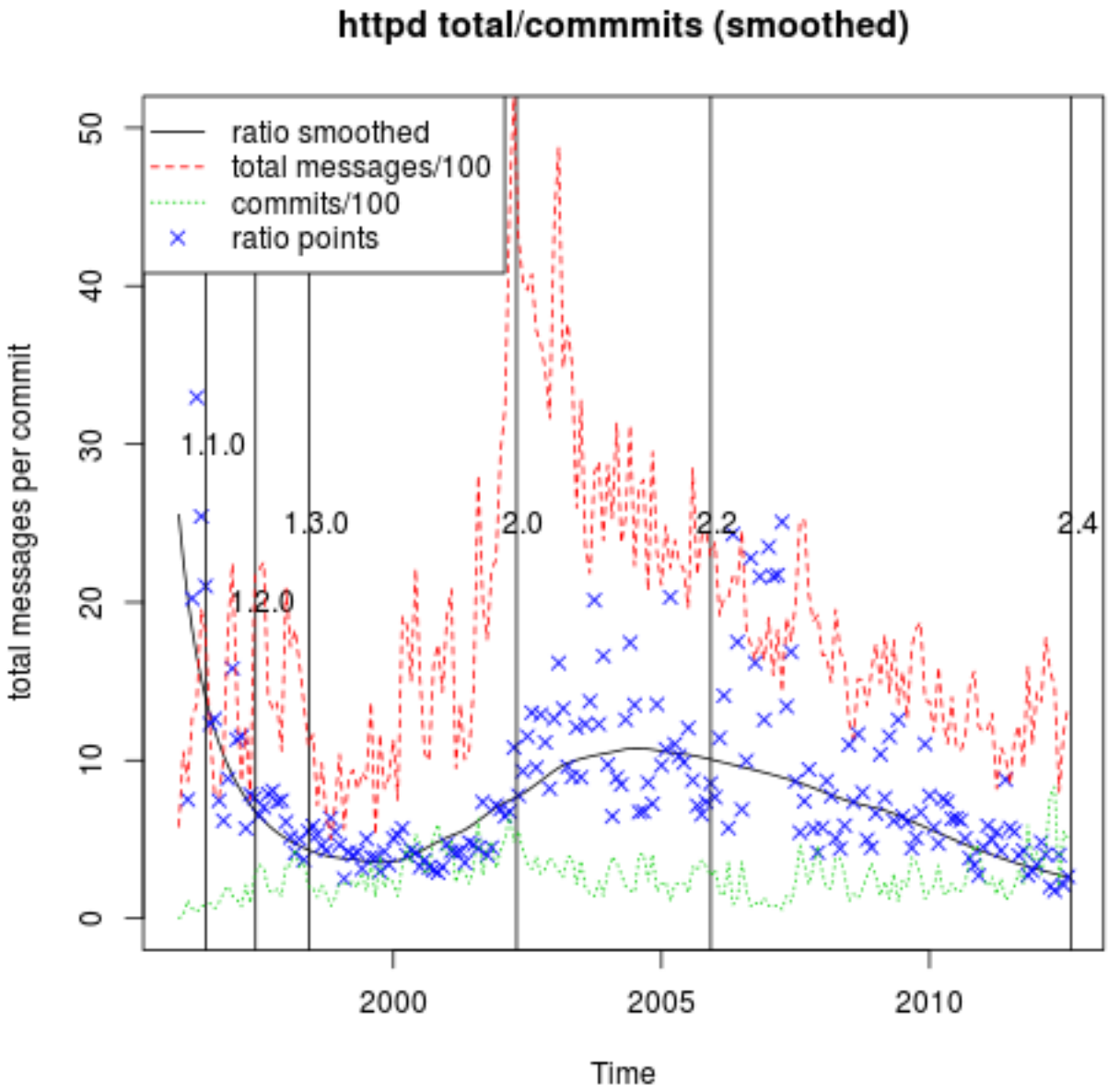
The Apache Web Server, or Apache httpd, is the project that gave rise to the Apache Group, first, and then to the Apache Software Foundation. It is the leading web server in the world.

The first recorded source code change was made on February 22, 1996. The previous version of Apache, 1.0, was released in January 1996 and had a separate CVS database. [FIE2] There is an approximate list of the early releases, compiled in 1999 by Ken Coar

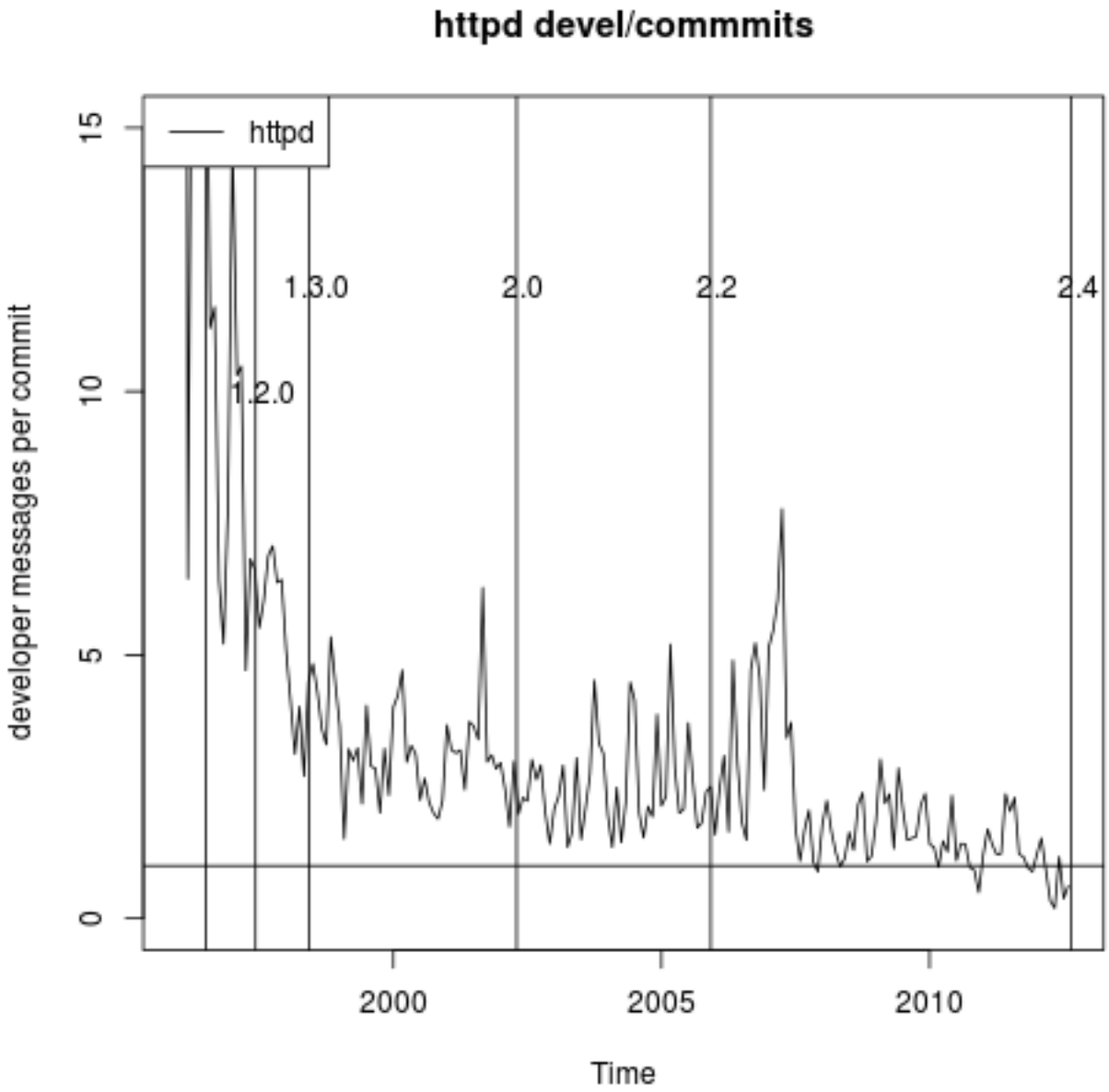
[COA1]. 2.0.35, the first release in the 2.0 series present in http://archive.apache.org/dist/httpd/ , was released in 2002-04-05.



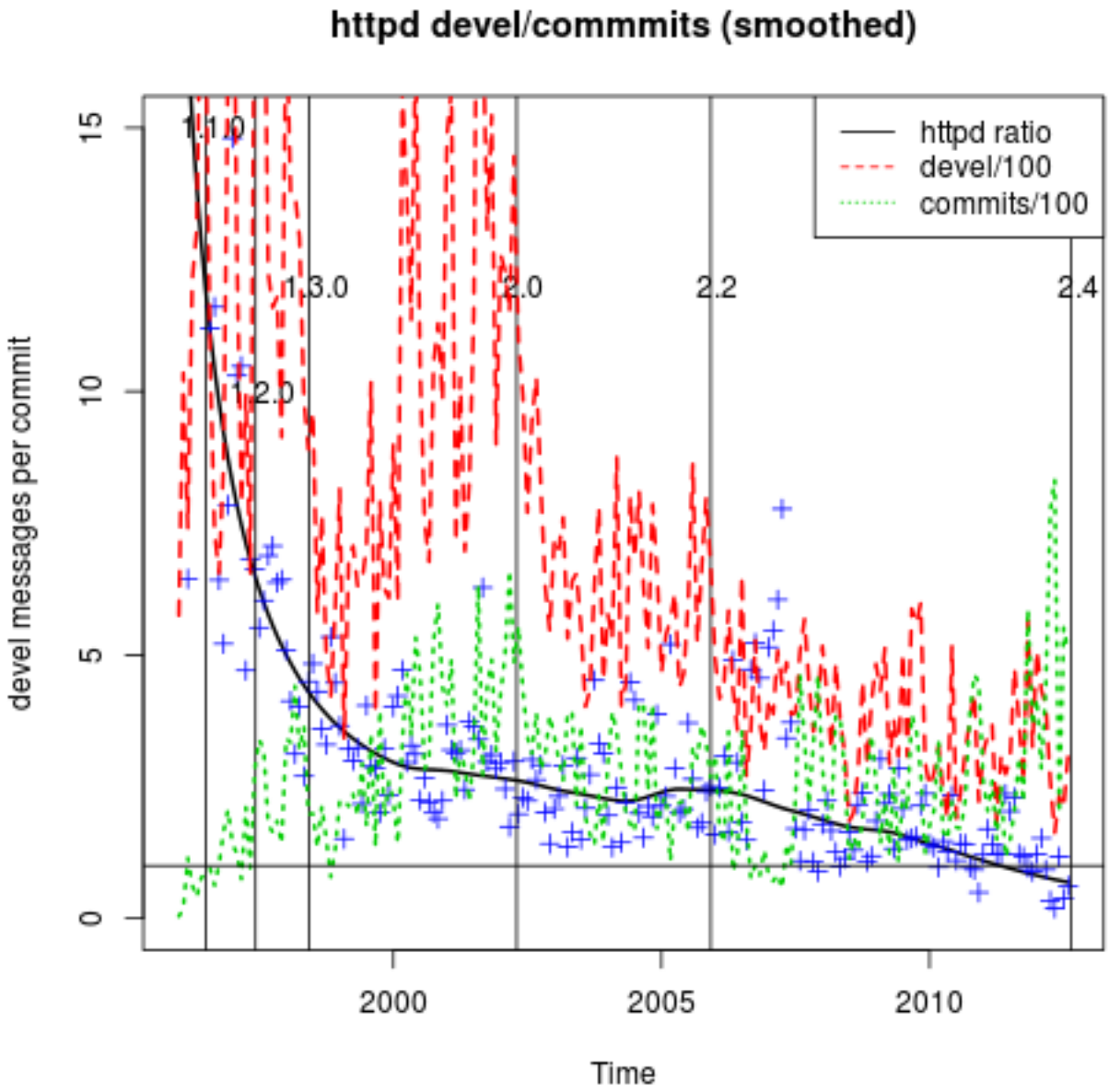
The total traffic was relatively high, with spikes of 2000 messages/month, and went down to around 1000 by 1999. Commits started growing, to around 100 at the beginning of 1997, and up to 300 around 1999-2000. This was reflecting the maduration of Apache 1.X. From then on there was a big growth in total traffic, followed with a certain delay by commits, which made the ratio go up to 10 by 2004. This growth corresponds to Apache 2.0 and then 2.2. After this the ratio of commits to total messages has been going down steadily, and even the recent release of 2.4 didn't make any change.



The increase in the ratio around 2001-2002 comes from a sharp increase in traffic, related with the release of 2.0. Apache HTTPD 2.0 had a new API and features. The extra traffic took some time to stabilise an then decrease. The commit flow kept reasonably constant in time, going slightly down after the release and keeping flat, but only the traffic decrease as the excitement faded and the knowledge about HTTP 2.0 spread returned the ratio to stationary levels.



The increase in total traffic is not seen in the developer list: here the traffic grows fairly symmetrically with the commits. Actually the commits outpace the traffic, as there is a decreasing coordination cost as the project matures and the organisational roles and design and implementation knowledge is spread and better documented.

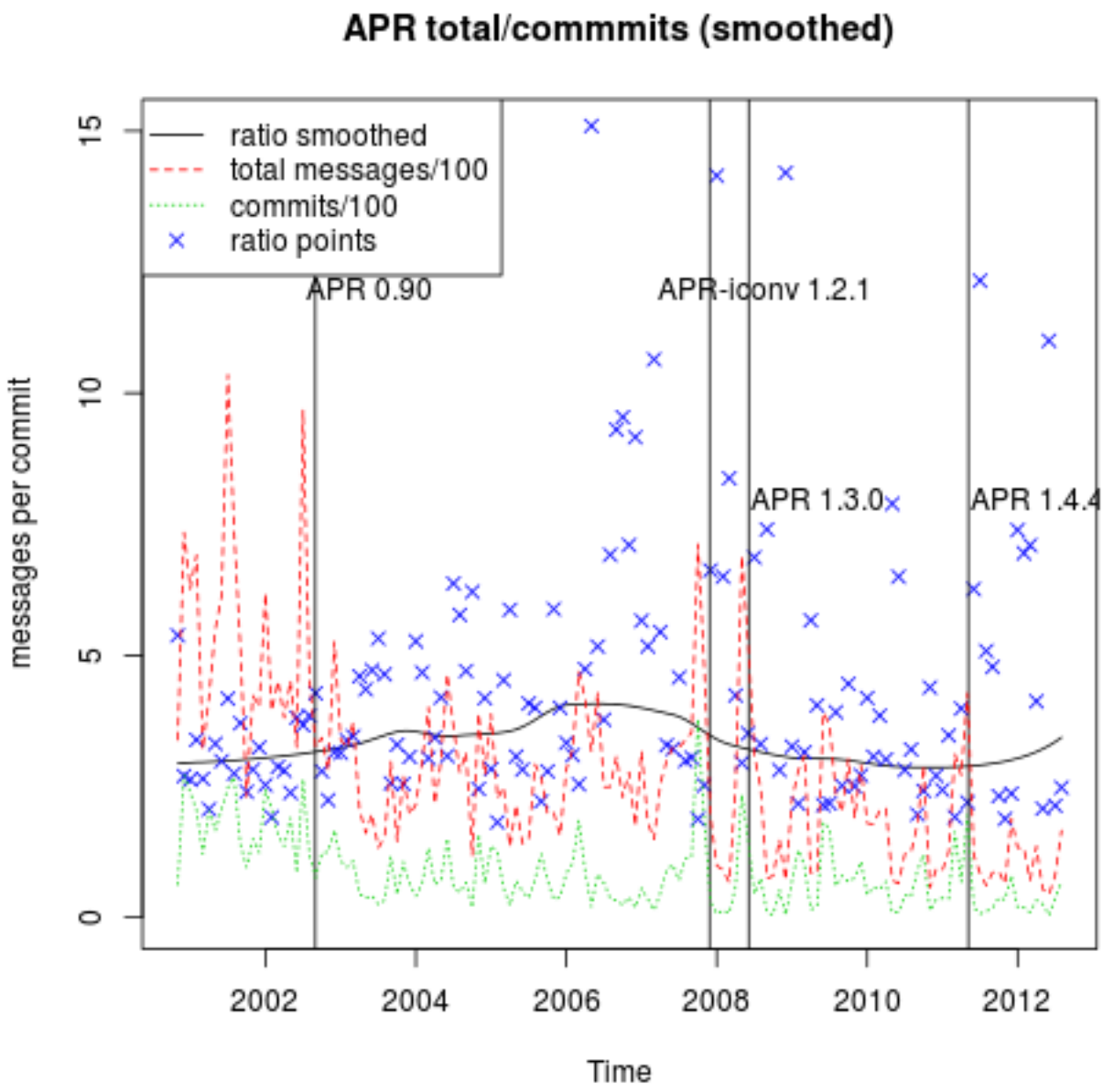


Apache is the first project that shows us this effect, that is happening in more projects, and we don't understand completely: how the developer traffic is going down, or at least grows less than the number of commits. We advance some hypothesis further on.

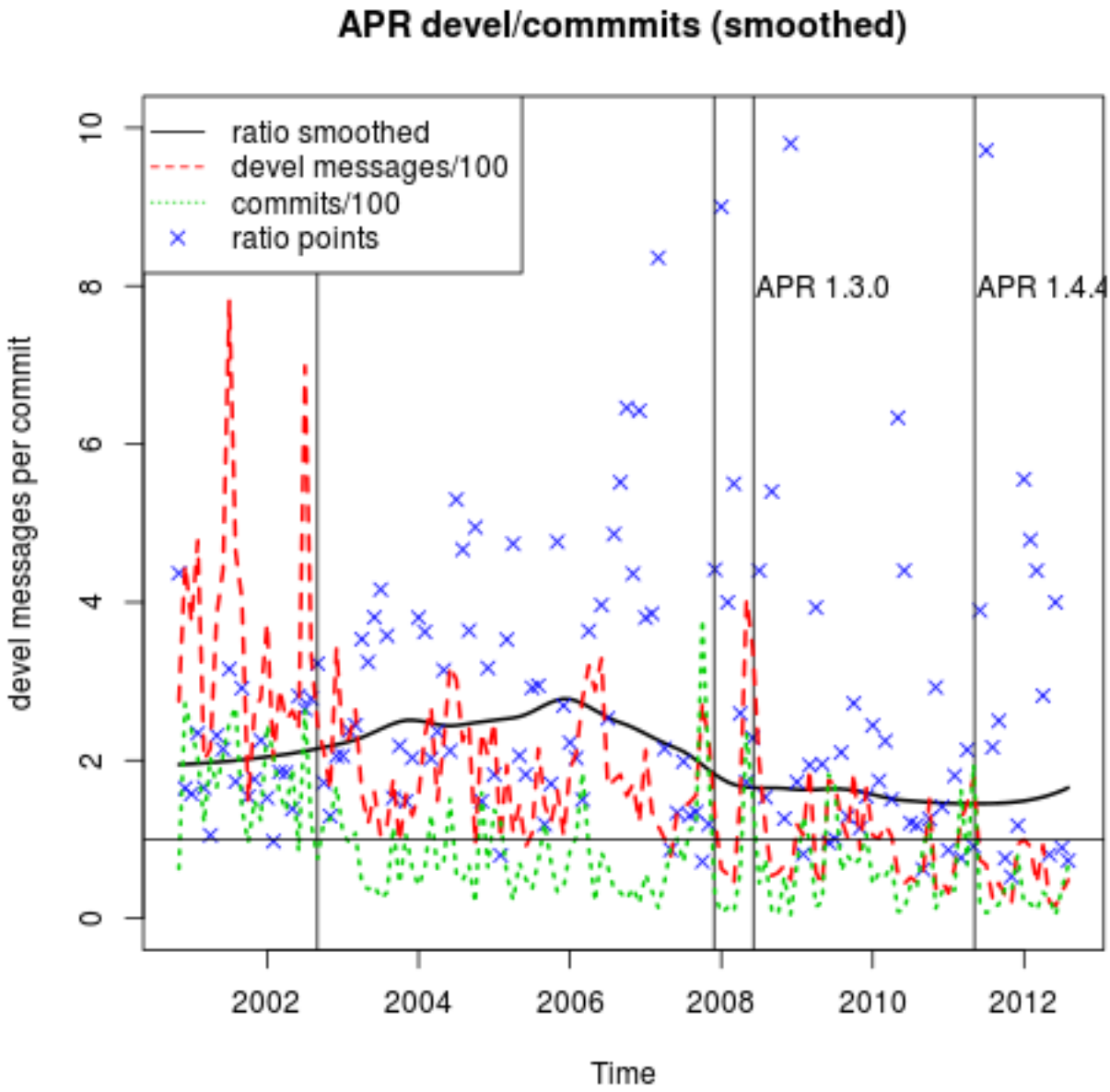
4.2.1 Apache Portable Runtime

The Apache Portable runtime (**apr**) is a library that eases porting of Apache to different platforms. It is also used by Subversion and other projects. Its traffic patterns are very different of the ones from **httpd**, with peaks of total traffic and commit traffic not much

related with the releases or activity of httpd, specially as it matured.
The pattern of total and devel traffic is very similar, as one can expect in a project driven by developers. On the other hand, the pattern of traffic is dual: undergoing bug fixing and maintenance, on one side, with strong spikes of traffic before releases.



Plotting the main releases, one can see that the runtime activity is polarized around their releases, and fairly decoupled from the server, but significant spikes of traffic followed major httpd server releases, especially 2.2, meaning probably that a number of bugs or design problems were discovered after further exposure of the code.



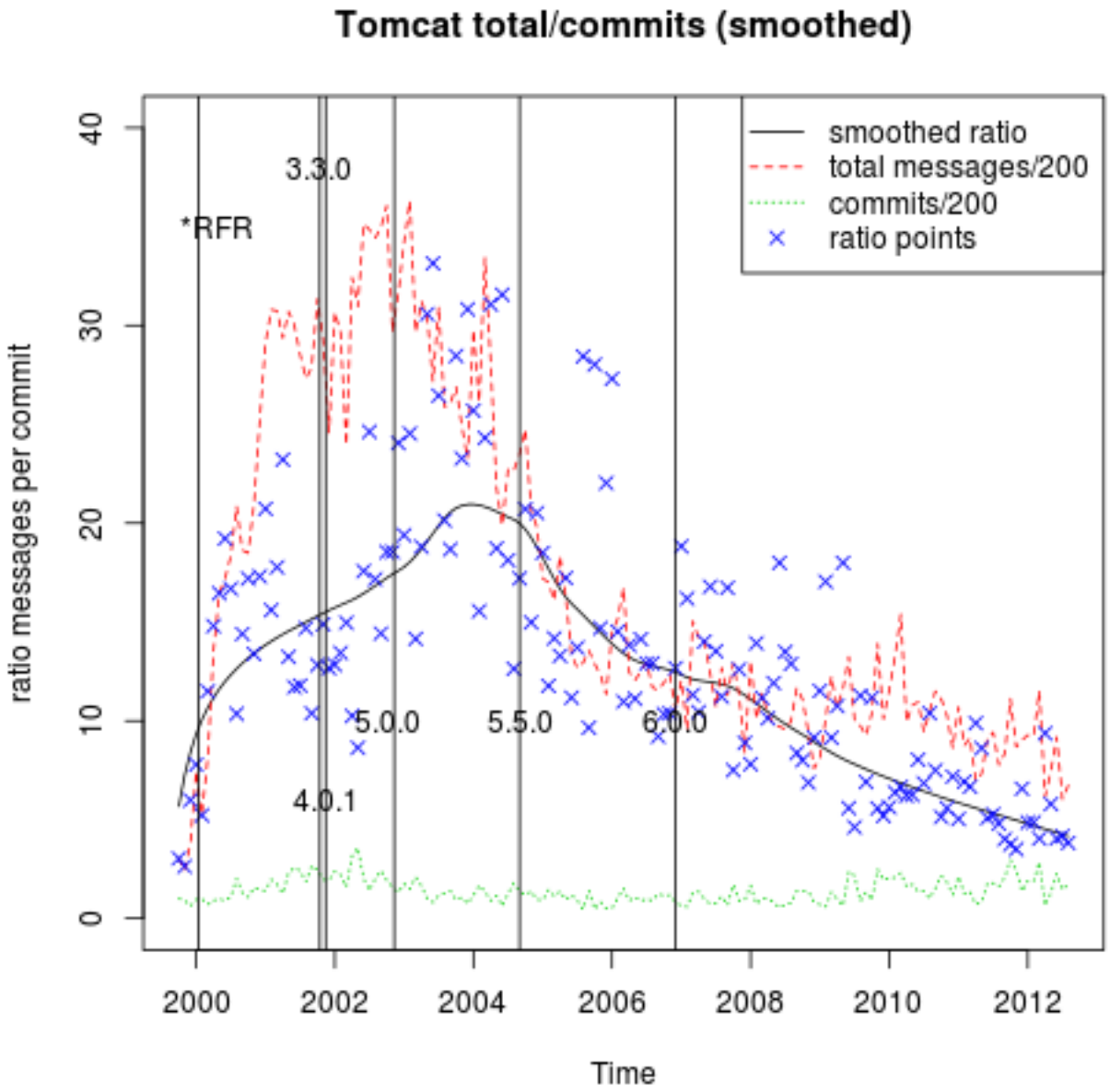
The **apr** library is used by an increased number of projects, but the main ones are Apache **httpd** and Apache **subversion**. Subversion started as a project in tigris.org, but joined the ASF in 2011.

4.3 Tomcat

Tomcat was one of the first projects in Apache involving a code donation. In Jakarta there was a project called JServ, which was developed to support the Servlet API 2.0. Sun Microsystems, at the time, had a product called JWSDK, reference implementation

of the Servlet API. They negotiated with people in the ASF to have those two products merged as an Open Source reference implementation.
Talks where held in a room called Jakarta, between the jakarta developers and Sun officers. The project was hosted in a new ASF project called Jakarta after the room name, that was charted to take all java projects inside the Apache Software Foundation. The official announcement was done in June 1999 during the Sun Java One conference. The project attracted a lot of attention since day one: JavaEE (the called J2EE) was hot in enterprises: java was definitely the new corporate programming language. And the web was exploding. So a HTTP server side programming API, part of J2EE and in a language like java, was bound to be evaluated by most companies writing dynamic web sites.
When Tomcat had released 3.2 there was a strong discussion about the future evolution. A long group of threads, with *Subject* starting with *[LONG TERM PLAN]*, was started by Craig R. McClanahan. 36 such mails can be found in Dec, 1999, and 61 during January 2000.
Craig recounts the history in a message :
I joined the Apache JServ project in January, 1999. The personal itch I wanted to scratch (with apologies to ESR) was to bring Apache JServ up to compliance with the 2.1 servlet API, which was still current at the time. As you probably remember, Apache JServ is based on the 2.0 API, and there were some really significant changes between 2.0 and 2.1.
After working with the existing code for a while, it became obvious that some fundamental design decisions were going to need to change to support 2.1. In addition, many of the then-current developers of Apache JServ started getting interested in other things (Jon -> ECS, Village, Town, and Turbine; Pier -> XML; Stefano -> Cocoon). I therefore took advantage of the opportunity to propose a rearchitecting of Apache JServ to both bring it up to date, and provide a basis for future flexibility. The idea was accepted, and work proceeded – not a lot different than the Tomcat.Next proposal being discussed here.
The code that was created for this is actually funtional, although it has not been stress tested or performance tuned. It is still in the Apache JServ CVS tree, under branch "JSERV1_1DEV" (originally it was going to be called 1.1).
Now, fast forward to June, 1999, and the announcement at JavaOne. Sun was doing what open source advocates had been suggesting for quite a while – they were contributing the source code of the JSWDK to the Apache Software Foundation! And they would continue to support the project, including some developers paid to contribute enhancements. Further, Tomcat would remain the reference implementation for the servlet and JSP APIs.

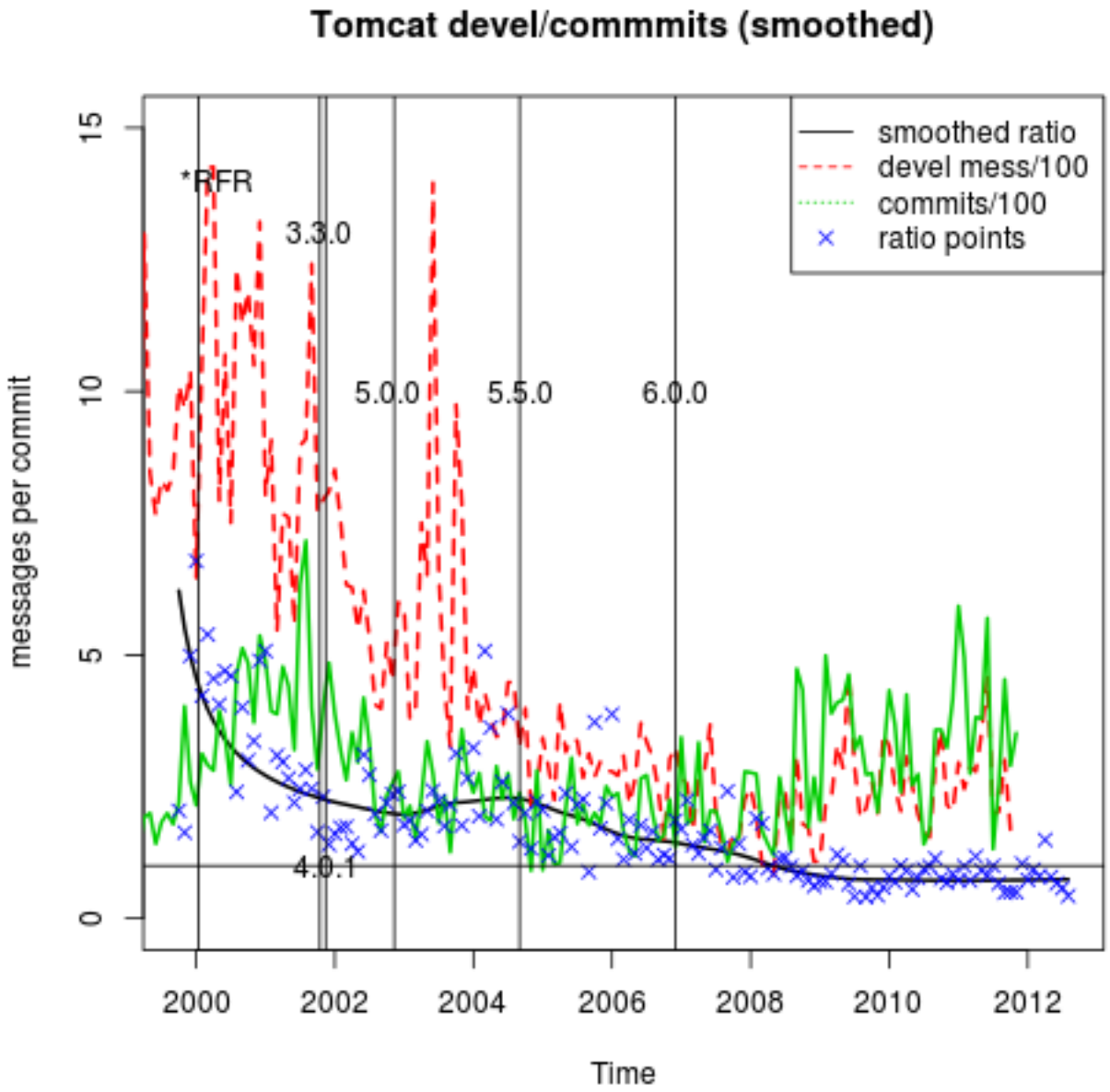
Costin Manolache, then a worker in Sun, expressed opposition to the refactoring . He preferred to have incremental improvements instead of a big refactoring/merge leading to a new major version in a branch. The discussion stalled, and on Jan 13th James Duncan Davidson sent an email with Subject Rules for revolutionaries [DAV1] [MAZ1]. The claimed that *there's no way that anybody can expect an open source organization to work the same way that a team in a corporate setting can.* Also, *to allow revolutionaries to co-exist with evolutionaries*, he proposed that anybody got the right to start a revolution **in a separate branch**, and propose a merge back when the branch was ready.
As a consequence, during a relatively long period, tomcat was releasing 3.2.X, 3.3.X and 4.0.X versions. While this feature can be thought to cause the spike in devel traffic during 2000, the user traffic was most probably not related: it was coming from the increase in use that the frantic development of the web and the consolidation java as a server side platform was bringing.
We can see in the next plot how 3.3 and 4.0.1, the first releases of the two competing designs, were released very close in time. There was a big activity releasing fixes, optimizing performance, etc. during the next years. At the end of this fork, tomcat 5 was based on the tomcat 4 architecture (catalina) and not in the approach that 3.3 was carrying on.



Some releases are signaled in the plot, including the RFR marker, which outlines the date of the *Rules for Revolutionaries* email message referenced previously. After this messages and in the big discussion that ensued, the number of messages grew up to values around 6100/month, and later 7200 around the release of 5.0.0

The total traffic numbers are very high, in spite of the fact that Tomcat always had much less use than httpd. A big difference, that merits further study, is how httpd was packaged and distributed with most linux releases since very early, and the list and issue trackers of those distributions were used as a first tier technical support. Tomcat, due to the non-free nature of java, took until about 2004-2005 to be packaged by distributions,

and even now there is a need to use non-free repositories for the Oracle JRE, and support by linux distributions is limited.
Also, linux developers and users typically install packages using the distribution package repository, and update as new versiones are released. They are also typically more familiar with cultural issues: where to find documents, how to proceed when a bug is found, etc. In contrast, typical windows developers *discovering* java in the early 00's would be forced to download, install and upgrade on their own, and less aware of cultural conventions, and thus make more noise in lists and be in more need of support. The arrival of windows developers, atracted to Open Source by the corporate commitment to java and the nature of the typical servers being open source, explains this strong surge in total traffic, not followed by traffic in the developer list.

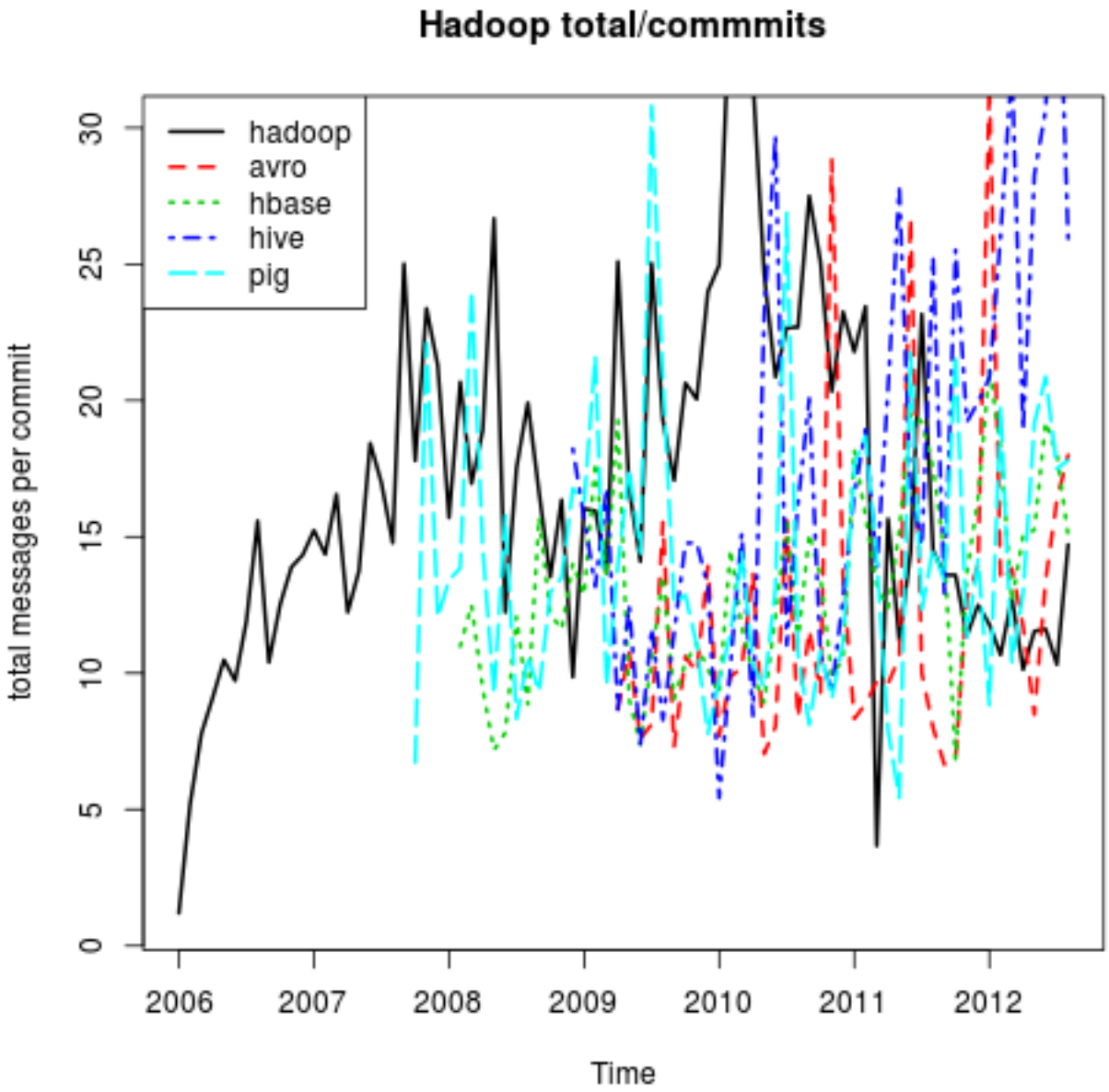


The number of commits was relatively stable during this spike, which made the ratio grow up to values around 20 messages/commit. As we see in the previous plot, the developer messages didn't grow so much as the rest of the traffic.

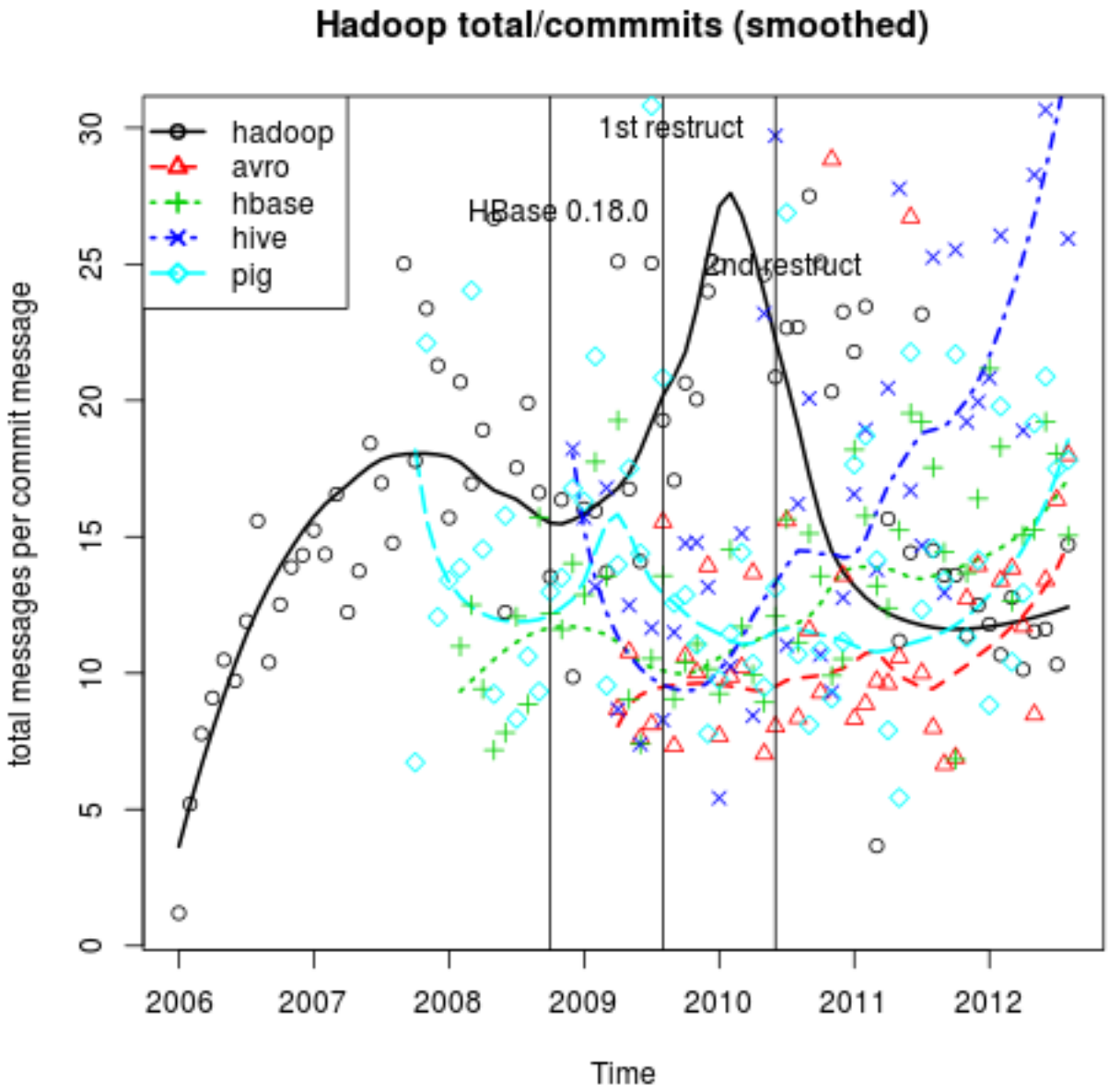
Tomcat devel traffic is getting progressively smaller as the Servlet API platform is mature and well tested. As with other projects, one can see that the number of commits increase. It remains to be seen if this increase is due to lowering the barrier for participation of non-developer users via distributed SCM or code review tools, or if it comes because the commits are more fine-grained and carry less changes per commit.

4.4 The Hadoop ecosystem

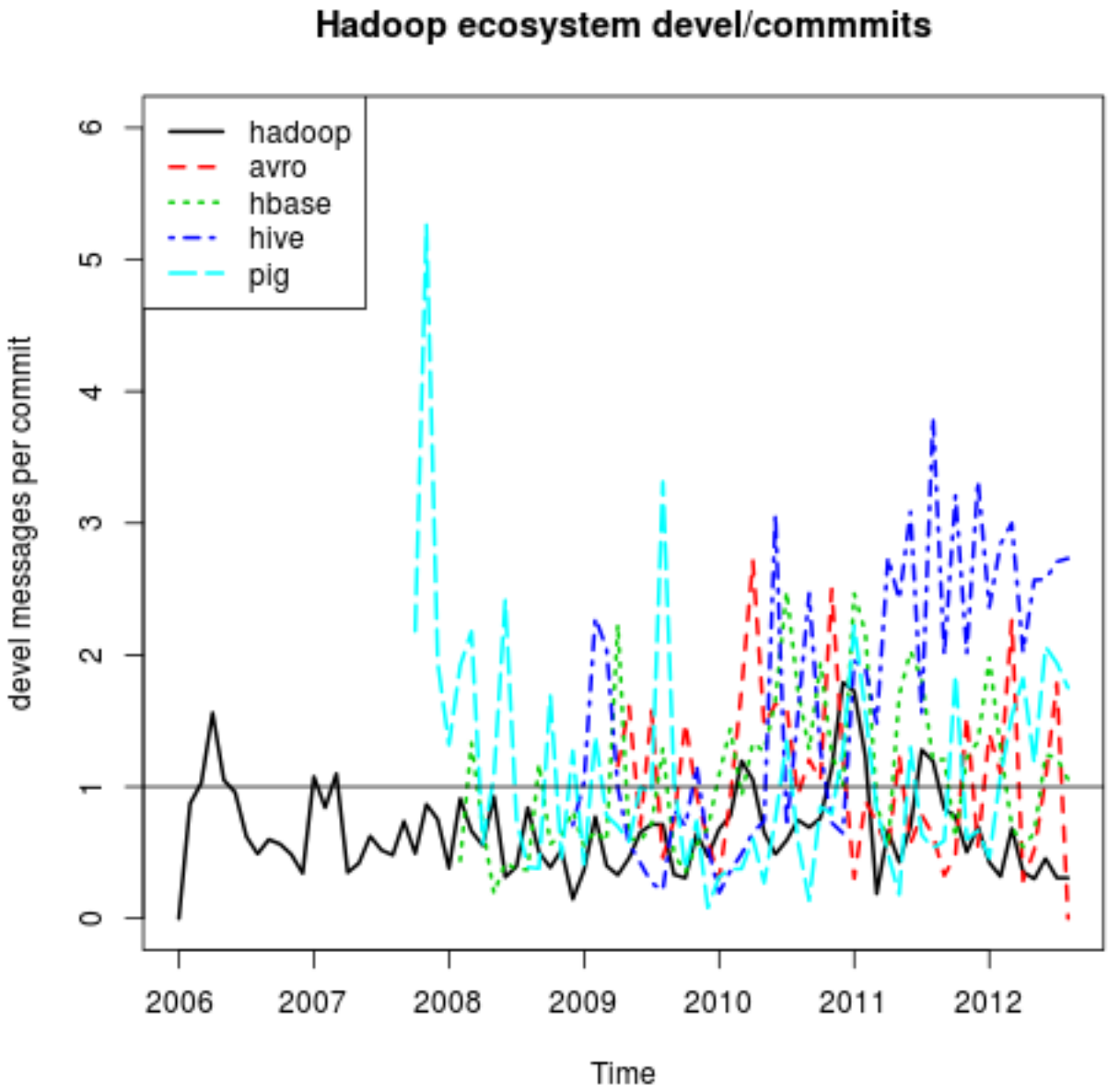
Hadoop is a framework for running applications on large cluster built of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named Map/Reduce, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. In addition, it provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both MapReduce and the Hadoop Distributed File System are designed so that node failures are automatically handled by the framework.
The Hadoop ecosystem contains, probably, the most popular ASF projects during the last 5 years. It is seeing development at a frantic pace since 2008-2009
We take significant events from the hadoop web site. In July 2009 a set of new hadoop subprojects were spawned. Hadoop Core was renamed **Hadoop** Common. **MapReduce** and the Hadoop Distributed File System (**HDFS**) turned into separate subprojects. Also **Avro** and **Chukwa**.
In May 2010, **Avro** and **HBase** graduated to become top-level Apache projects. Apache **Avro** became http://avro.apache.org/ ; Apache **HBase** is at http://hbase.apache.org/ .
Apache **Avro** is a data serialization system, and **HBase** is the Hadoop database, a distributed, scalable, big data store.
A few months later, in September 2010, **Hive** and **Pig** graduated to become top-level Apache projects. Apache **Hive** can now be found at http://hive.apache.org/ **Hive** is a data warehouse system for **hadoop** that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in **hadoop** compatible file systems. Apache **Pig** can now be found at http://pig.apache.org/ . **Pig** is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs, provided by **MapReduce** and the rest of **hadoop**.
In January 2011 a new project, **ZooKeeper** graduated to become a top-level Apache project. **ZooKeeper** is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
We have plotted the first 5 such projects. One can see that the ratio of total messages to commits is much higher than other Apache projects.



The smoothed ratio shows a very high spike of **hadoop** in 2010, and an even higher spike of **hive** right now. It is easy to understand if we know that all companies are gathering tremendous amounts of data and analyzing them, and hadoop is a very popular tool for this.

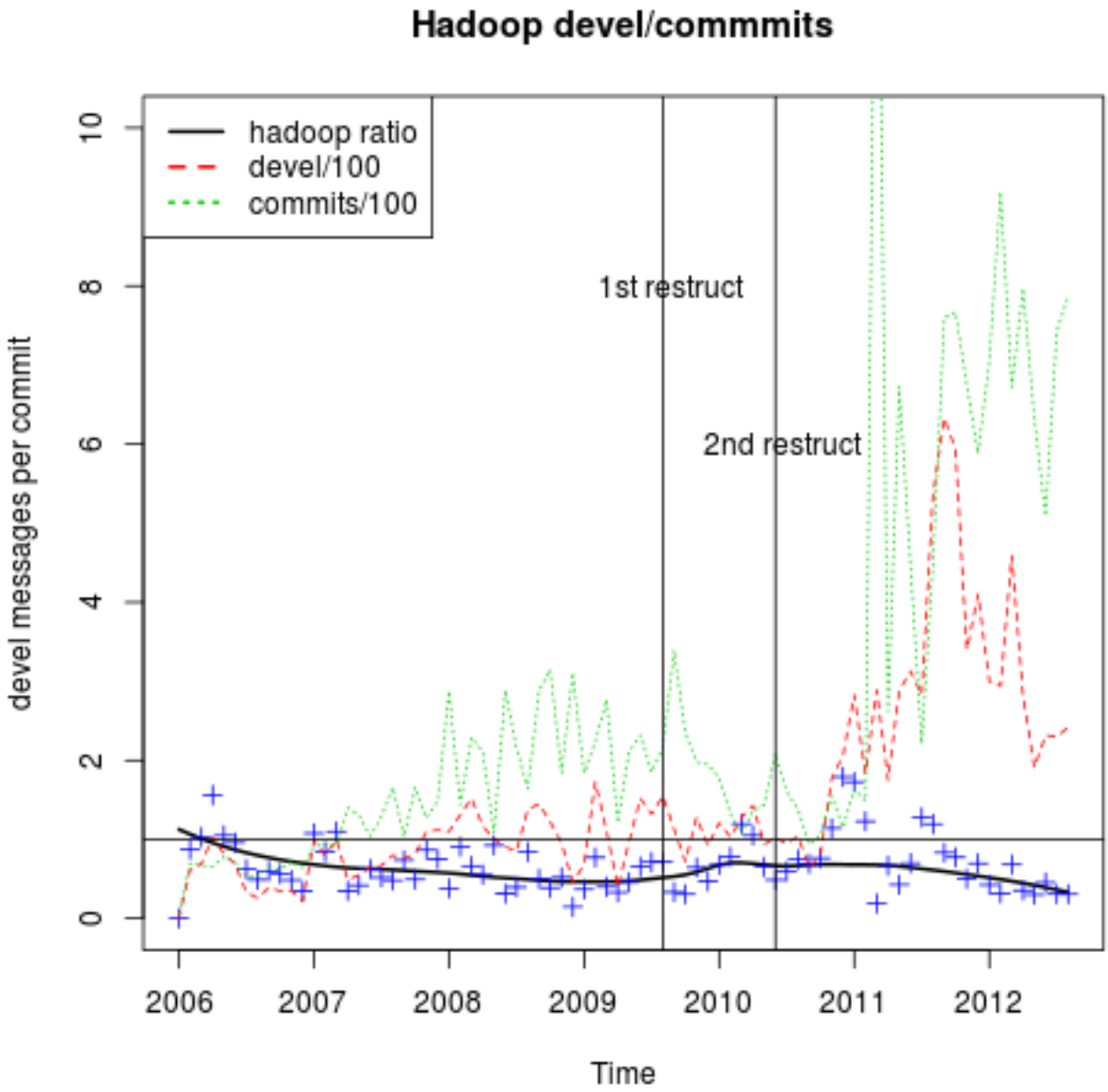


All the projects are seeing very high ratios of messages to commits. Since 2007 all the projects are above 10 most of the time. This points to a lot of user activity as compared to developer, or to a workflow that uses the issue tracker for most communications. We are counting the issue tracker in total, but not in devel.

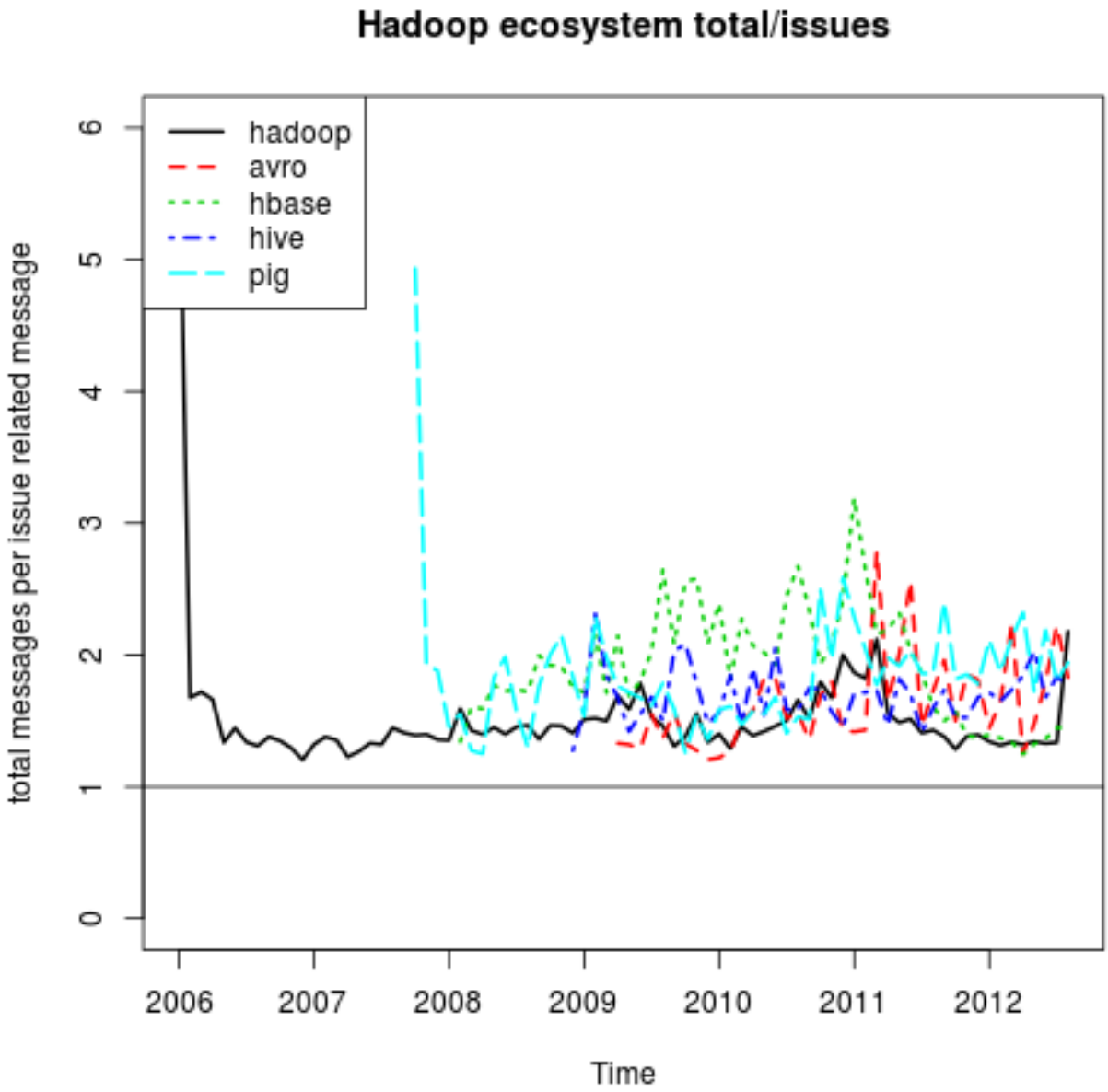


The ratio of developer messages to commits is, though, rather small, below one most of the time. This indicates that the commits don't need coordination in the developer list, but are coordinated via, probably, the issue tracker, or perhaps code review or through the use of github pull requests. The fact that a common vocabulary and culture develops might also explain a decrease of the developer traffic, but of the **hadoop** ecosystem, only **pig** shows this pattern.

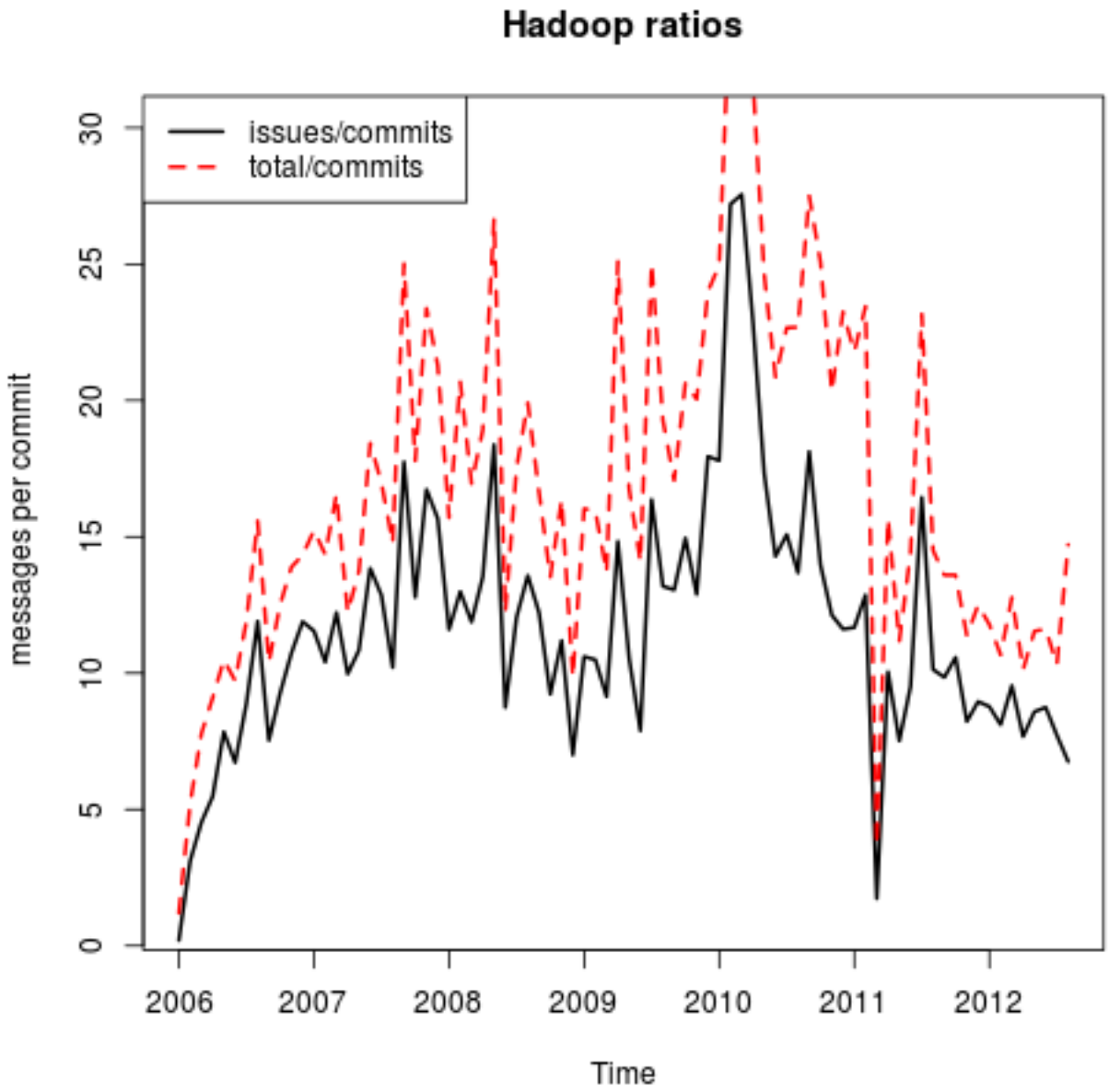
A not so healthy interpretation would be that the groups of committer, which have been typically hired by a small number of companies, are coordinating in closed lists or just chatting during lunchtime. This hypothesis merits further study.



Effectively, if we graph total traffic/issues, we find that issues are close to half of the traffic in hadoop. The next plot shows that a substantial part of the project traffic is in the issue trackers, and this explains both the high total/commits ratio and the low devel/commits ratio: developers coordinate with users and together through the issue tracker rather than using the developer list.



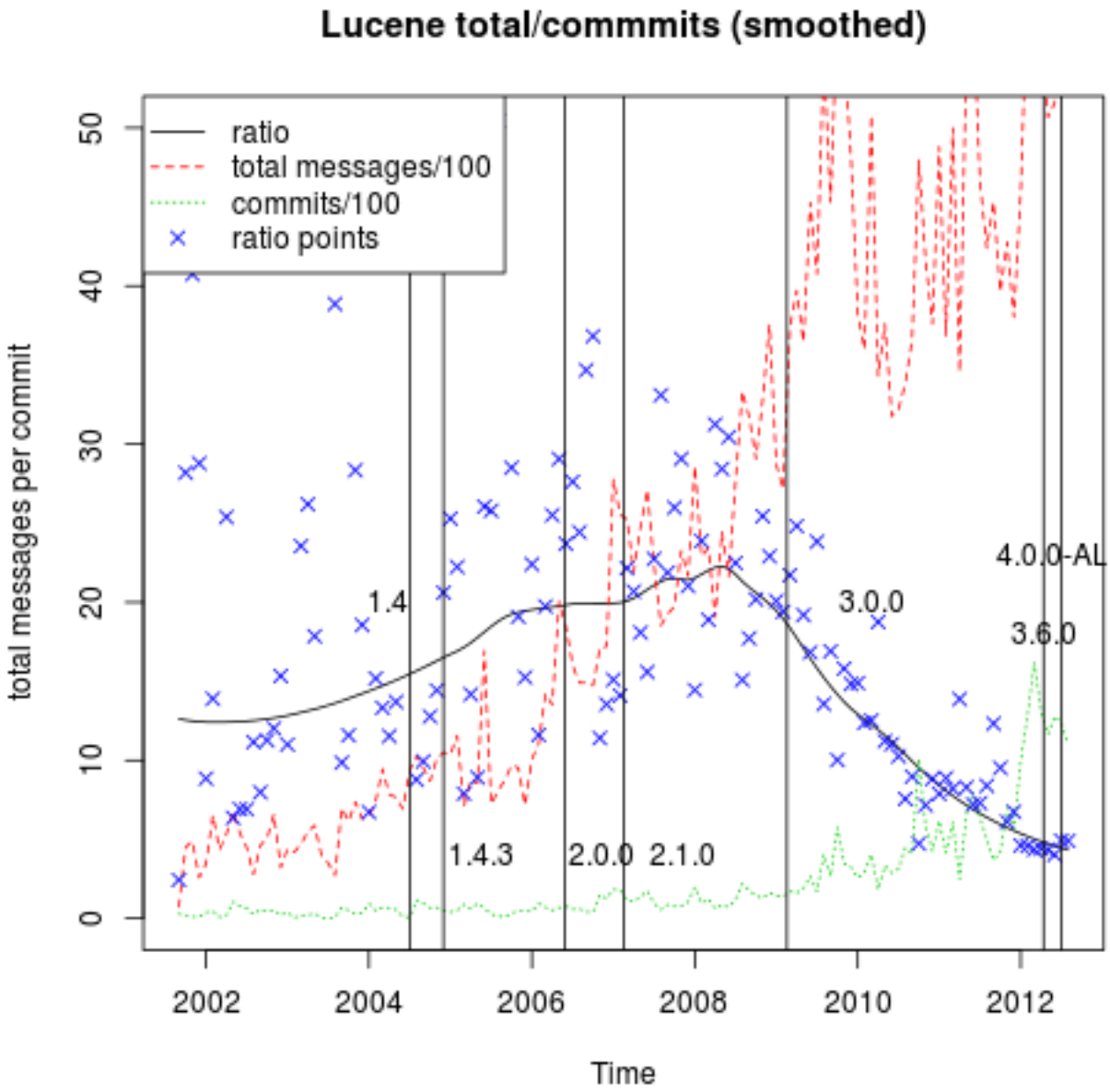
Obviously, the activity in the hadoop ecosystem is issue tracker driven, with between 50% and 100% of the project exchanges taking place in the issue tracker.
In the next figure we plot the ratio of issues related messages to commits, compared with the total/commits ratio that we are using. We can see that it is similar to the total traffic plot, as hadoop development is highly tied to issue tracker activity.



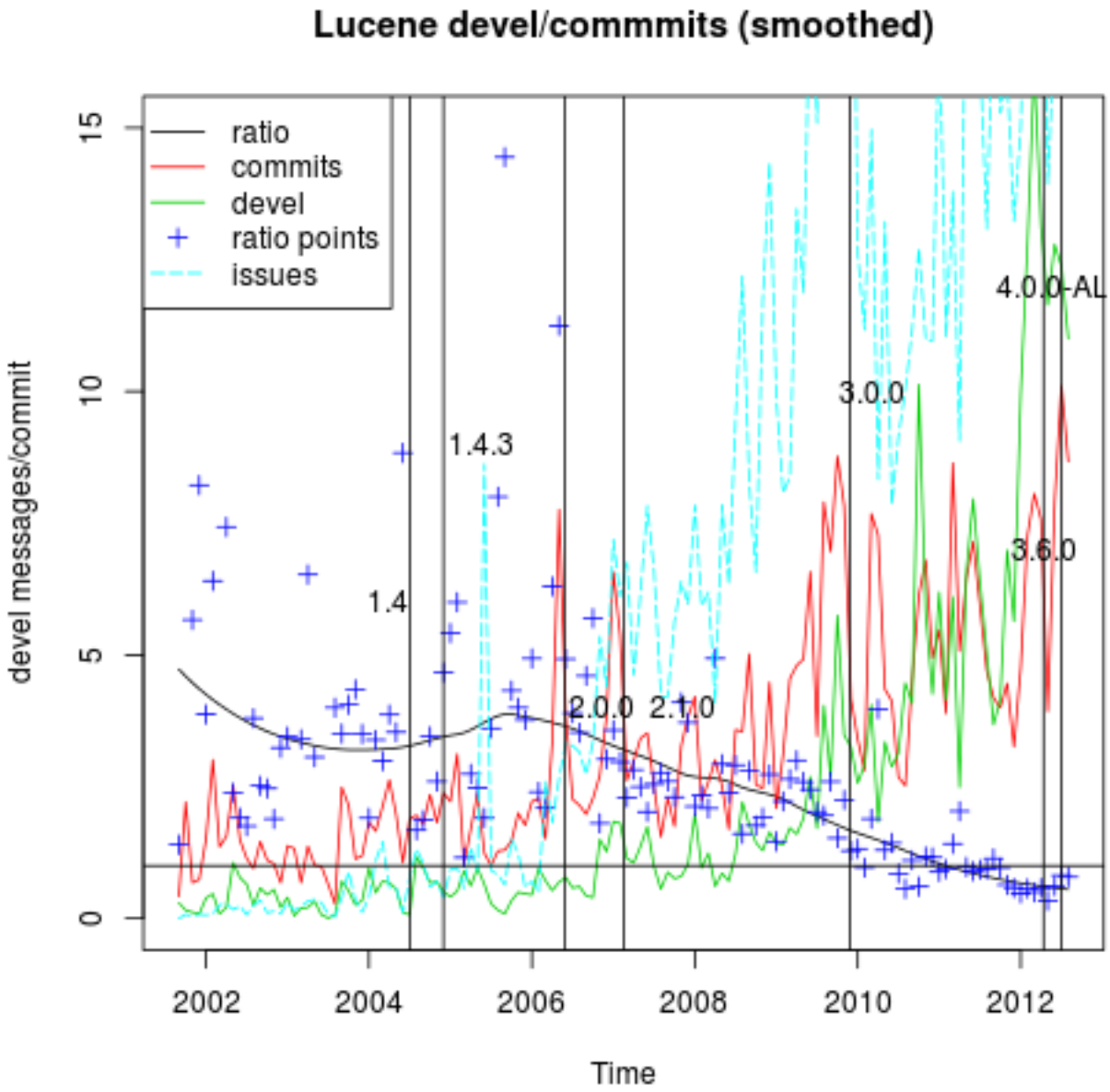
4.5 Lucene

Lucene is a Java based indexing, search and also spell checking, hit highlighting and tokenization

The ratio total_messages/commits ranges from 30 - 51 during the first few months, to diminish slowly and stabiliz between 4 and 6 during the next years. Currently it is around 4. In spite of that, it saw a tremendous increase of traffic since 2009, but the number of commit is growing at a faster pace and outgrows messages.



There is a significant decrease in activity in June and July 2004. It looks it is due to testing of the different release candidates and the Lucene 1.4 release. The points from 2002 to 2006 show a lot of fluctuation, due to the fact that most of the commits were done by a few committers, and were very sensitive to their availability.
The peak of the ratio in 2008 is not, like in Beehive, due to a decrease in the number of commits, but rather to a sharp growth in the total number of messages. The increase of commit activity was delayed, showing that at this stage of things Lucene was *pulled* by users, rather than *pushed* by developers. Effectively Lucene is used in more and more contexts as search engine.

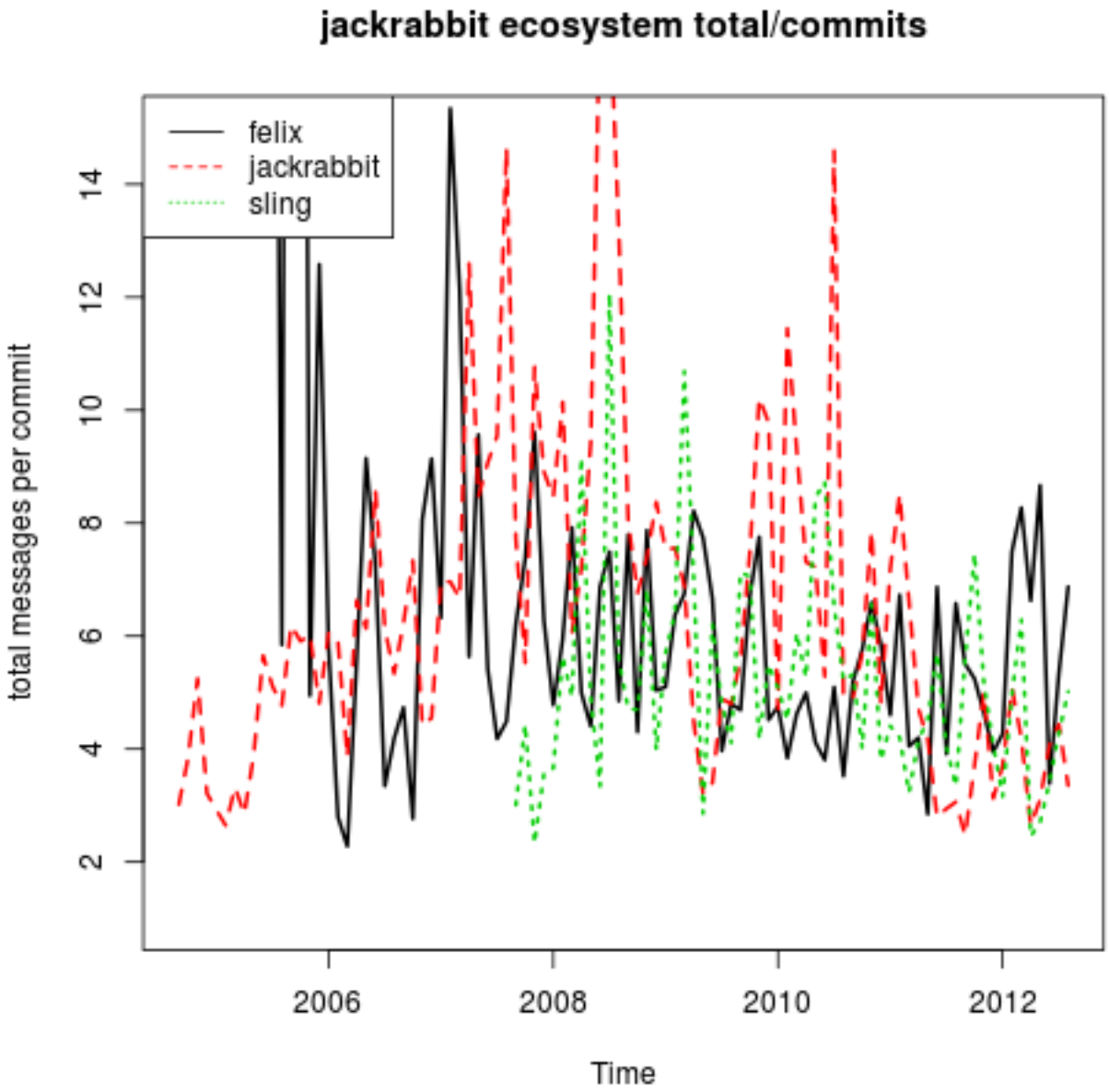


A similar pattern is visible in the devel list: an sharp increase in developer traffic between 2004 and 2006, while the commit traffic only starts following the pattern around 2006, when the ratio peaks. Since then the ratio has greatly diminished to typical values of a mature project, getting smaller than one from 2011 onwards.

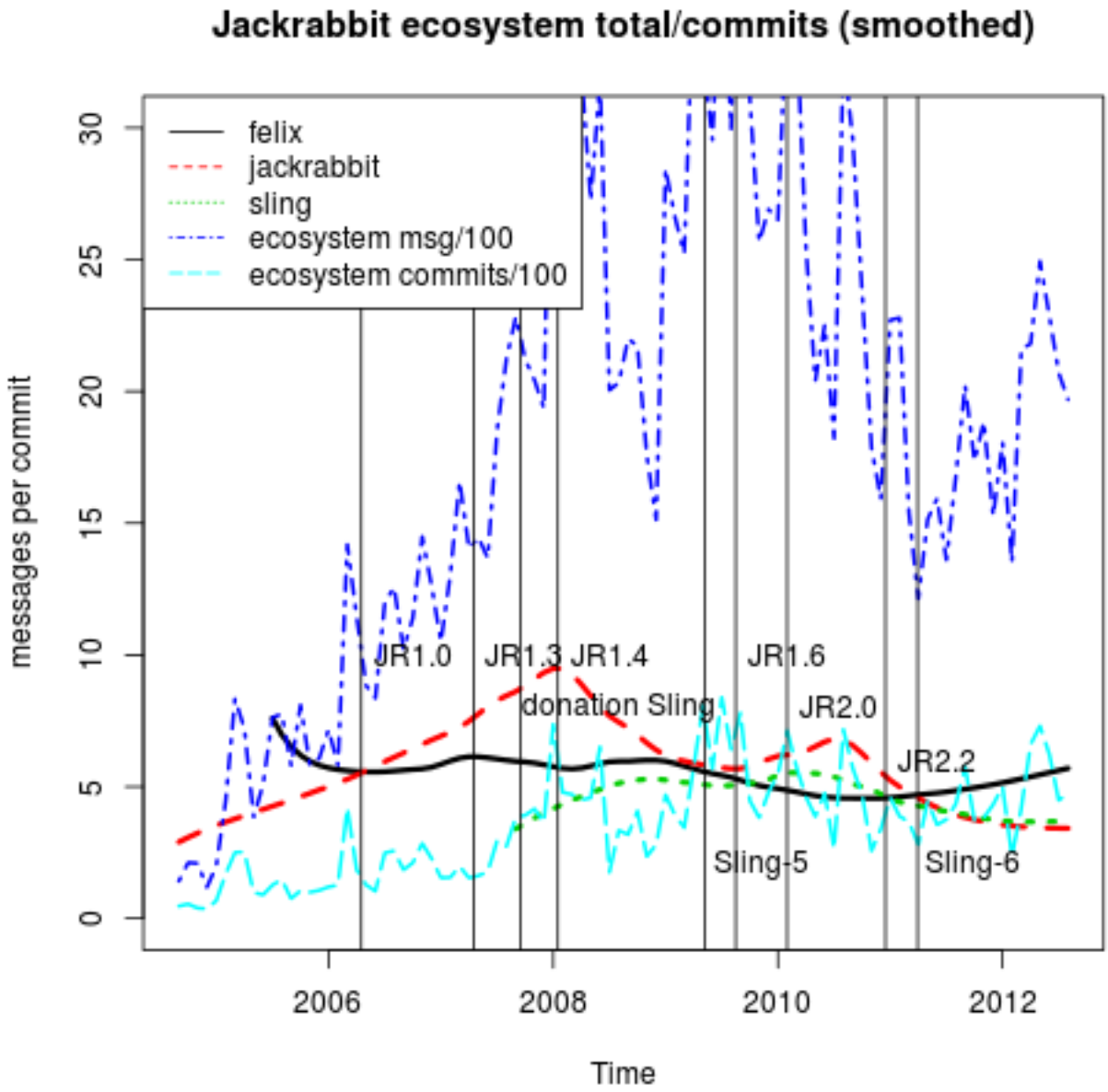
Lucene is not so much issue driven as the hadoop ecosystem, so its shape would merit further study to find the reason why the shape changes in such a way, with such a big lag between traffic and comits.

4.6 The Jackrabbit ecosystem

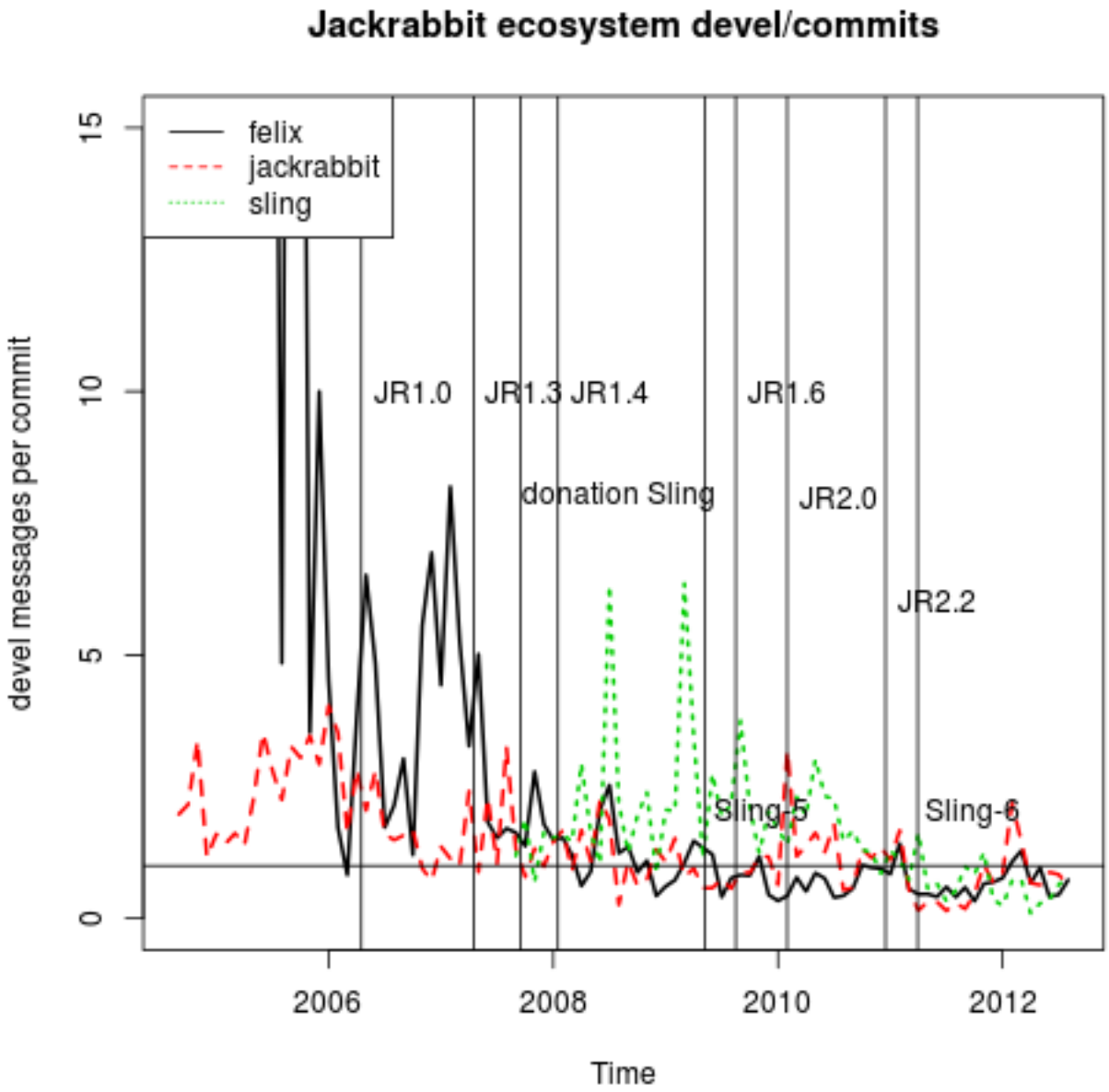
The JSR-170 (Content Repository for Java) specification was chaired by David Nuescheler starting in February 2002, and got approved on June 2005. The idea of having Apache Slide implement JSR-170 was around since early, and in February 2003 it was proposed that Slide was a reference implementation of JSR-170. Prototype code produced by Day Software was imported into the Slide code repository at the time, to be removed one year later because it had no clear status and was being abused as being **the** reference implementation. In August 2004 the early reference implementation code was re-sent, this time as a separate project, to the incubator, and eventually graduated as **jackrabbit**. Apache **jackrabbit** is a hierarchical content store with support for structured and unstructured content, full text search, versioning, transactions, observation, and more. It has got a relatively successful product. Day Software based its CQ product on it, and donated parts of it back to the ASF: **Felix**, a OSGi Service Platform and **Sling**, a web application framework.



Due to the use of OSGi bundles, **Sling** is very modular. Bundles have fine grain dependencies, and the concept of a full release is only relevant for the standalone binary download. A big number of component bundles can be tested and released separately. In the next figure one can see how global activity increased a lot in 2009-2010 with commits proportionate, so that the ratios have remained stable.

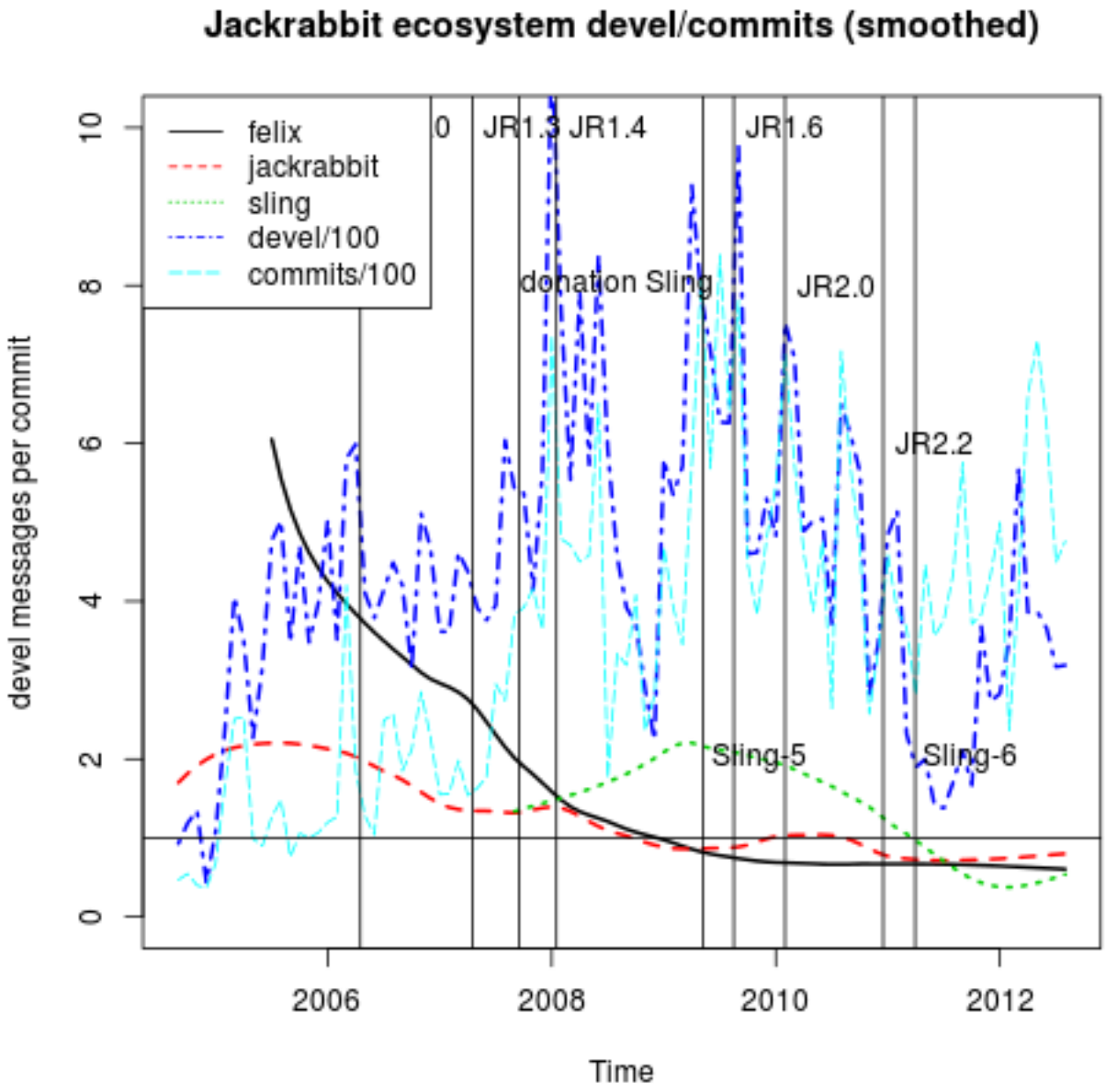


The ecosystem has attained success, with **Jackrabbit** having processed 1500 messages in July 2009, **Felix** 2168 in September 2009, and **Sling** 1688 in July 2008, and 1616 in August 2010. **Felix** continues to be relatively successful, being one of the two top OSGi Service Platforms, and the one independent of Eclipse, but used by netbeans, and it is seeing further adoption. **Sling** has slightly slower activity. Probably the reason is that there is a big number of Content Management platforms in PHP, python, Ruby, etc., and the main inroads of java based CMS systems into enterprise use are in big, java-only shops, which tend to trust commercial vendors and not feed public, community based activity.



The ratio of devel messages to commits shows that all three projects are fairly mature, with not very *exciting* evolution. **Jackrabbit** probably has most of its architecture/design decisions taken in the JSR-170 group, not visible here, and **Sling** was donated to the ASF as a mostly functional product. **Felix** shows more initial activiy.
The three projects show ratios of devel/commit messages below 1, even with healthy numbers of commits and messages. This is a sign of a well integrated team and a mature and well structure work group, as the need of coordination between developers falls down when the teams are well formed, though it could be also due to a growing number of finer grain commits. It might also be due to communications off-list, as a substantial

number of committers are coming from Adobe/Day.

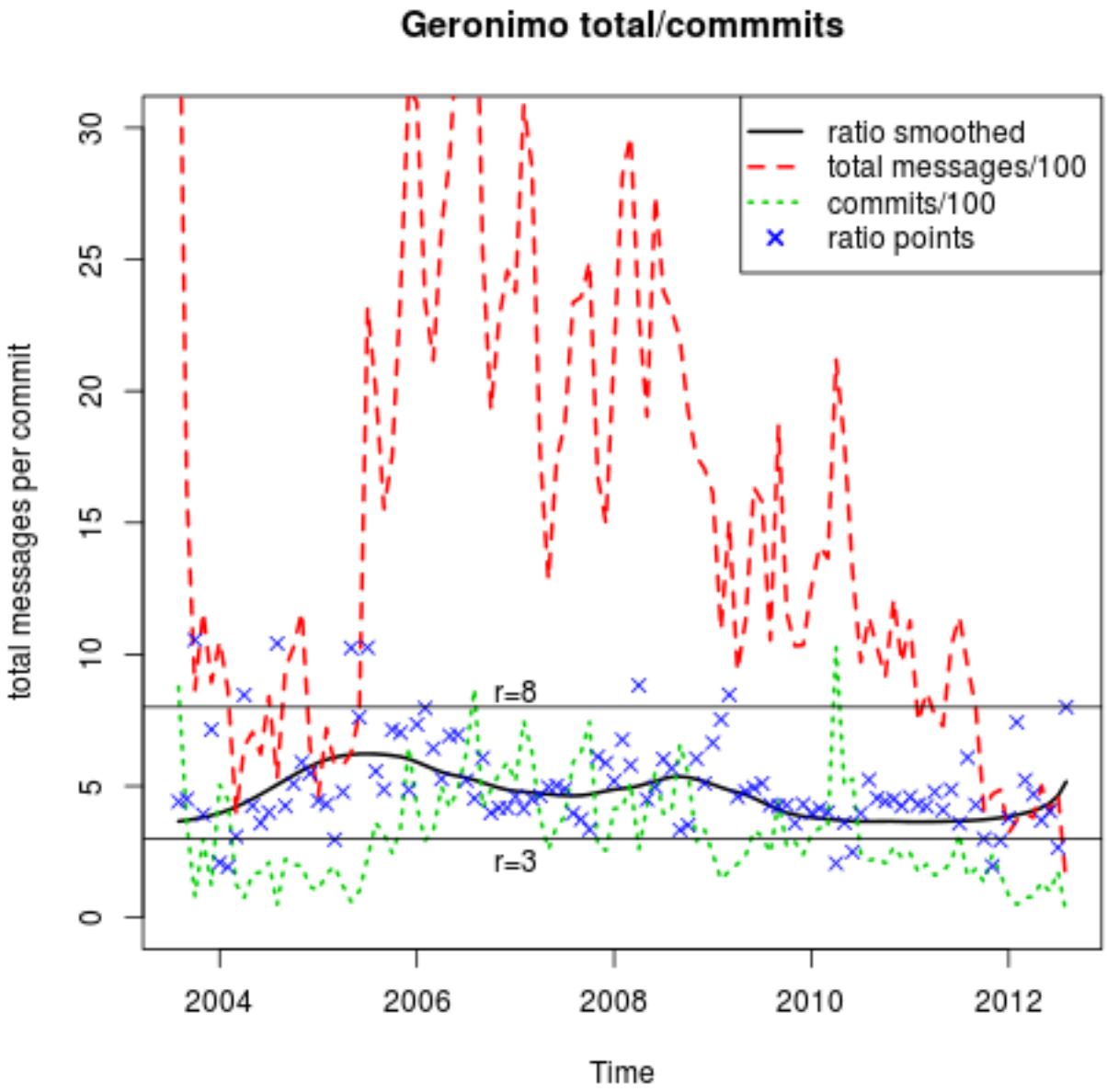


4.7 Apache Geronimo

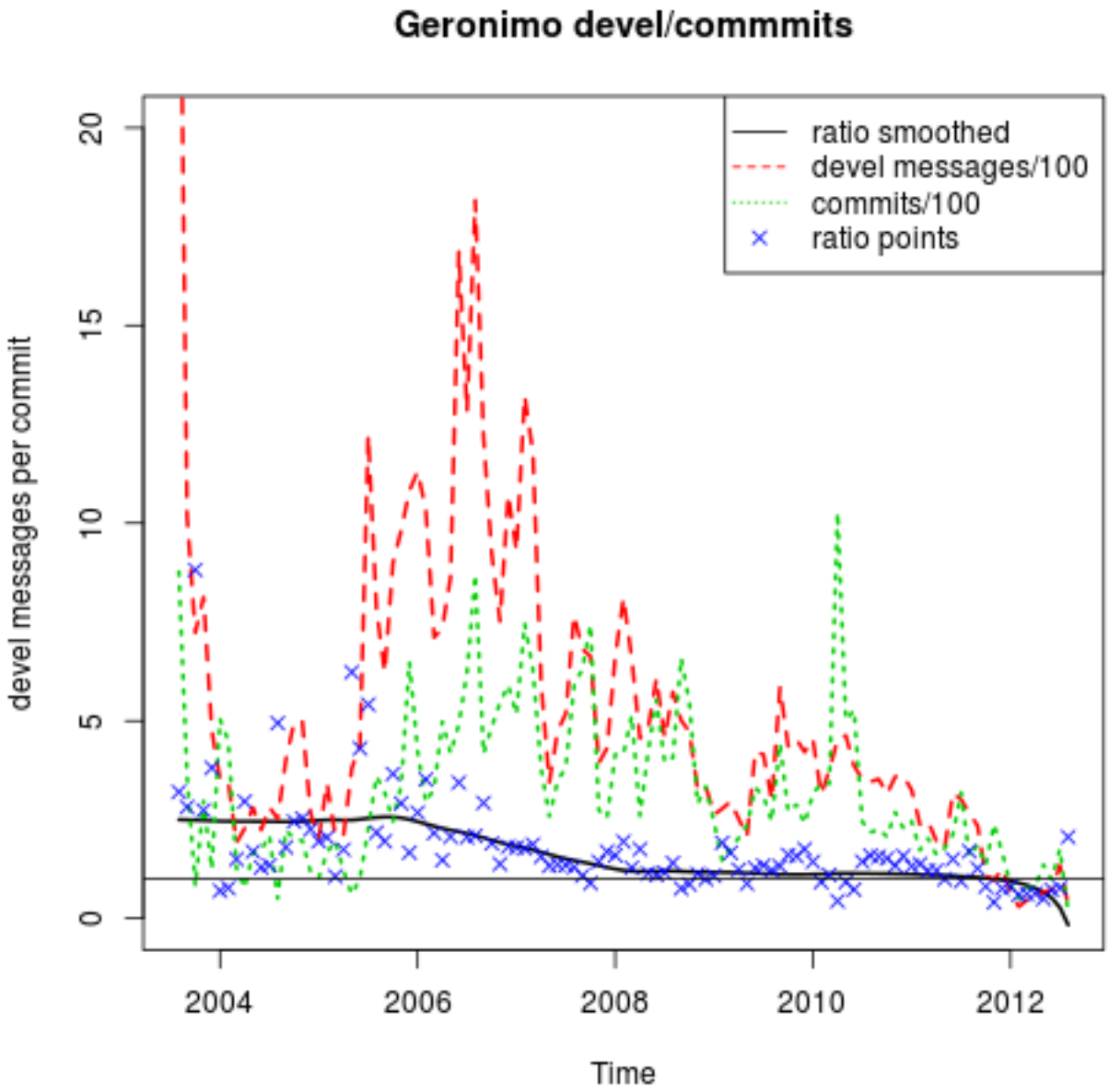
Apache Geronimo is an open source server runtime that integrates the best open source projects to create Java/OSGi server runtimes that meet the needs of enterprise developers and system administrators. Their most popular distribution is a fully certified Java EE 6 application server runtime.

Geronimo was born when a group of developers involved in several JavaEE components

got in a conflict with JBoss and asked that Apache would release a certified JavaEE Application Server passing the Test Suite. It was adopted by IBM as a community edition of their WebSphere Application Server. It shows a uniform ratio of traffic to commits.



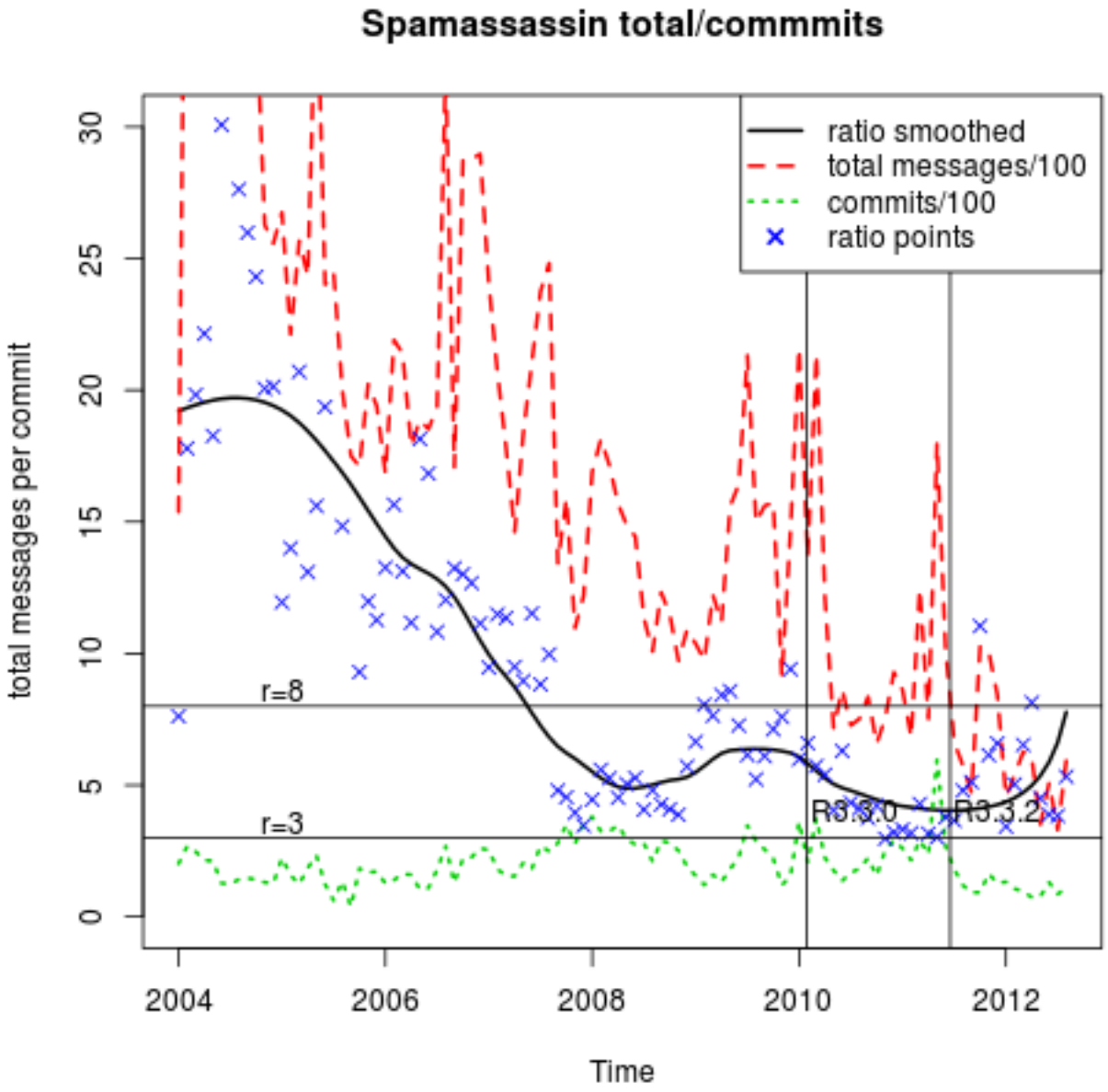
As the project matures, it shows less developer activity, but the commit traffic stays high: this makes the project show more commits than developer messages since 2012.



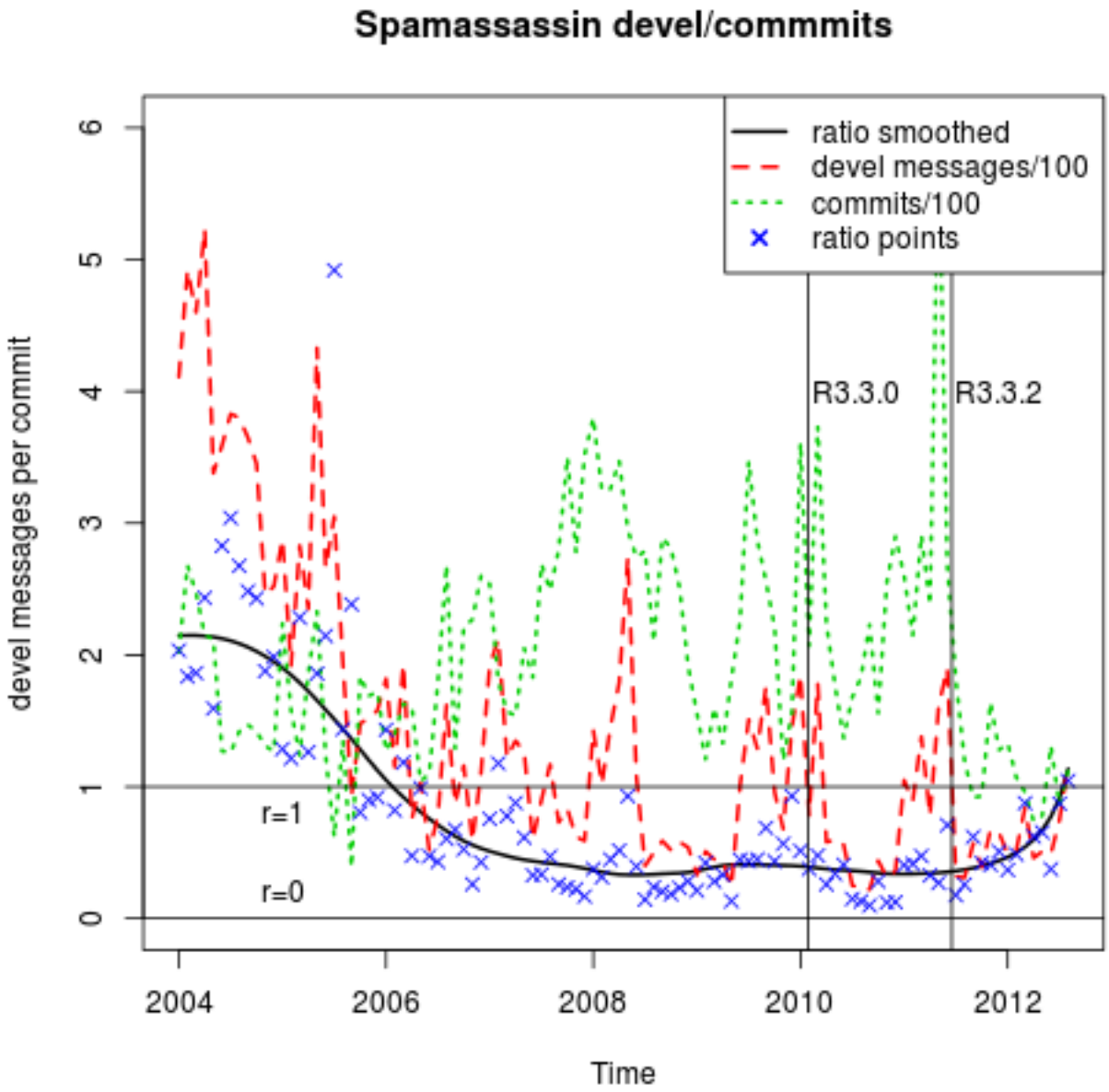
4.8 Spamassassin

SpamAssassin, according to wikipedia, was created by Justin Mason who had maintained a number of patches against an earlier program named filter.plx by Mark Jeftovic, which in turn was begun in August 1997. Mason rewrote all of Jeftovic's code from scratch and uploaded the resulting codebase to SourceForge.net on April 20, 2001. In summer 2004 the project became an Apache Software Foundation project and later officially renamed to Apache SpamAssassin.

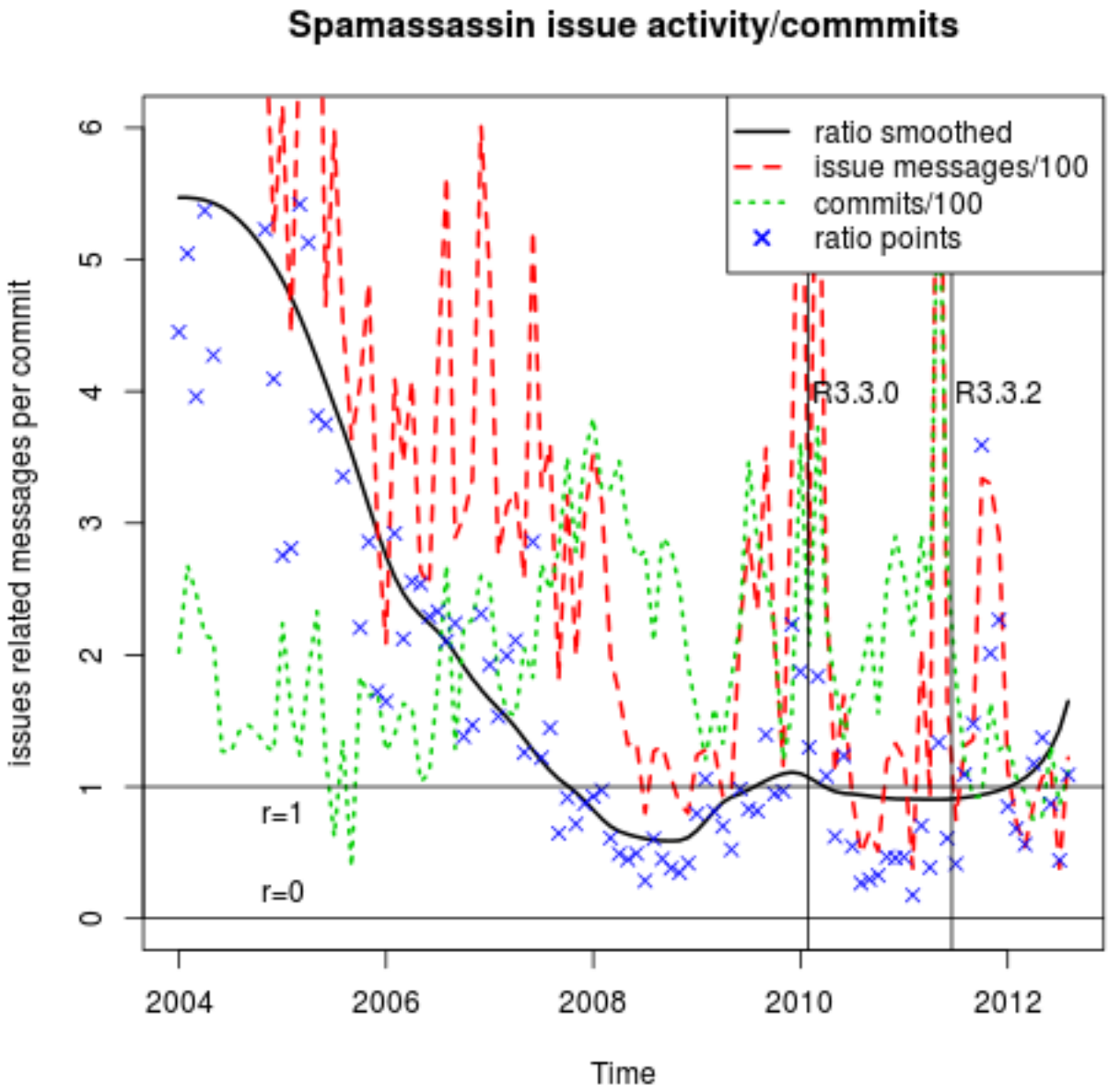
The plot of total activity against commits shows how the activity settles as the project matures. In spamassassin the design was fairly mature already when the project joined the Apache Software Foundation. So the total ratio has gone down to 5, and stayed between 3 and 8 since 2007. Spikes of activity are related to new releases.



The developer message to commits ratio is below 1 since 2006, meaning that most commits don't need any discussion between developers, being done either by consensus or as a result of issue related fixes.



One can see that issue-related activity pattern is similar to devel activity. But while devel list activity goes down as the project matures, most of the commits now are mediated by issues rather than by discussion in the devel list. Also, as spamassassin uses bayesian rules to assess email signatures and classify email, a lot of activity goes into training and testing rather than typical development.

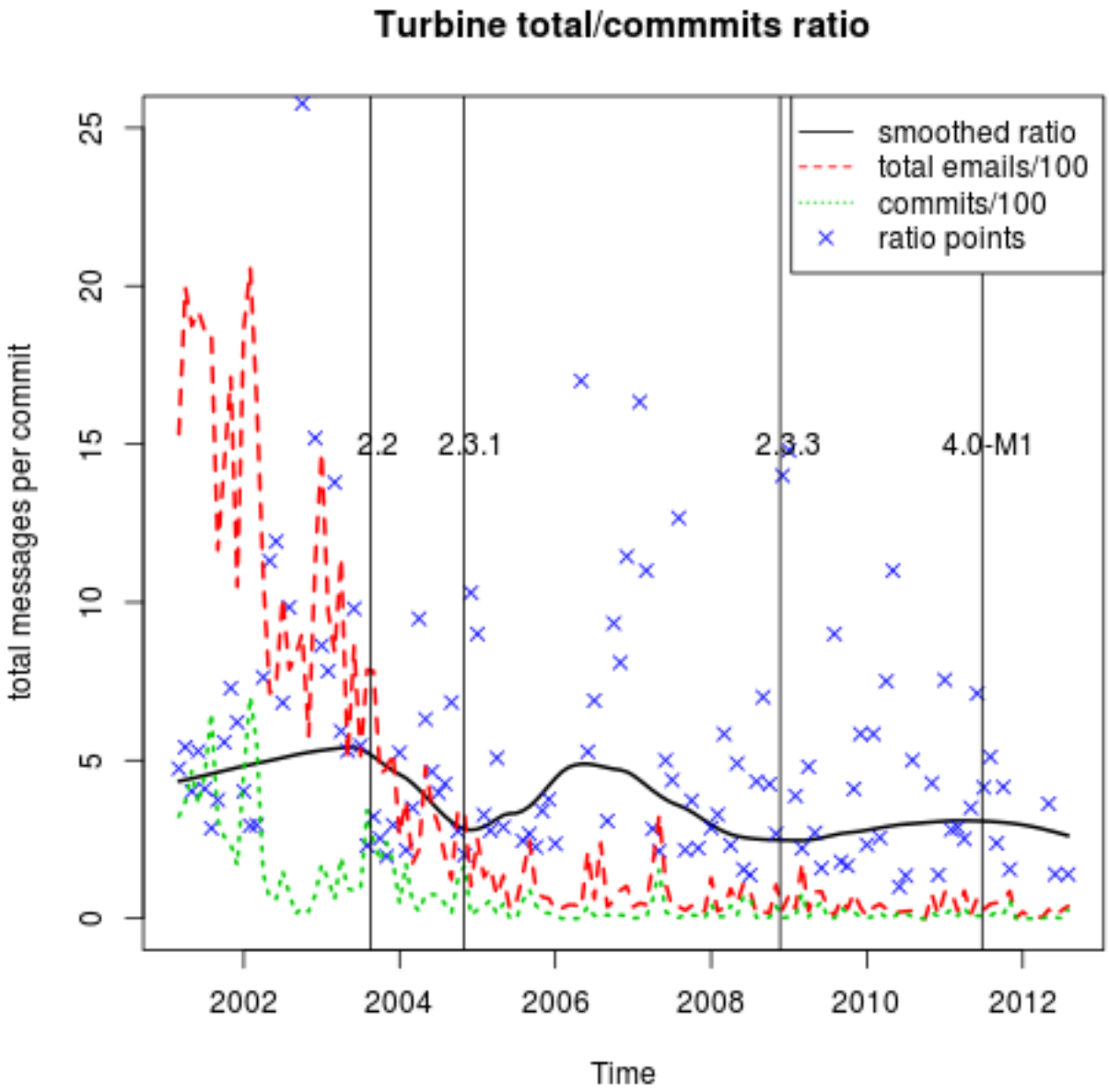


Issue related activity behaves similar to development activity, with a certain delay and more related to use of the project.

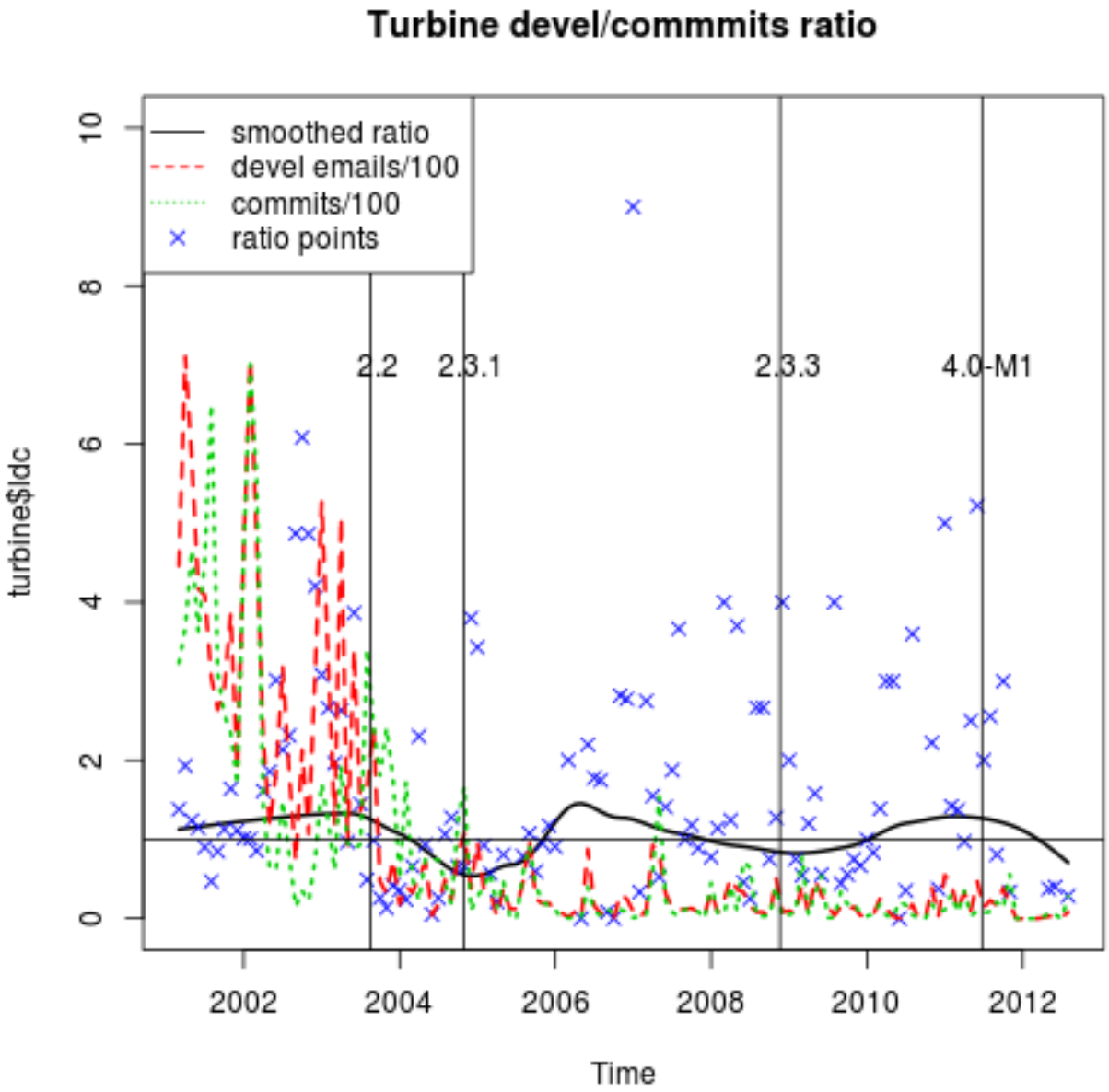
4.9 Turbine: a mature project

Turbine is a first generation Web framework, a very early one. It was popular in 2000, and it was slowly phased out as next generation frameworks such as Java Server Faces (JSF) or Struts took its place. We bring **turbine** as a good example of a project that was very big and got stagnant.

The archives for **turbine** from 1998 to 2001 were not in Apache and got lost. One can see that the total email traffic went down since 2001-02, where our data series begins. In January 2006 the traffic was 26, down from 1865 in January 2002. Two orders of magnitude of decrease. Now, the ratio messages/commits at those two periods were of 2.4 (Jan 2006) and 4.0 (Jan 2002), and the smoothed ratios where both around 4.5
Turbine 3 never was released. People were moving to different alternatives at the time, and it went victim of a continuous series of refactorings where the component functionality and APIs never went stable but keep changing and breaking dependent projects such as **Jetspeed**. Even as the total traffic went down by a factor 100, a small group of developers is still using it and working on it, and the ratio of messages to commits shows a steady progress. Version 4-M1 is being currently tested.



One can see how the project was processing peaks of 2000 emails per month in 2002, and it had fell down one order of magnitude by 2005. as users moved to different web frameworks like Struts or JSF. As the project gets smaller the ratio of messages to commits starts being much more noisy, because the number of commits is no longer predictable, but the smoothed values remain in the same range. In fact, one can see that the curve is relatively smooth, without big signs of attracting new users or developers. The project is mostly stable, and still alive.

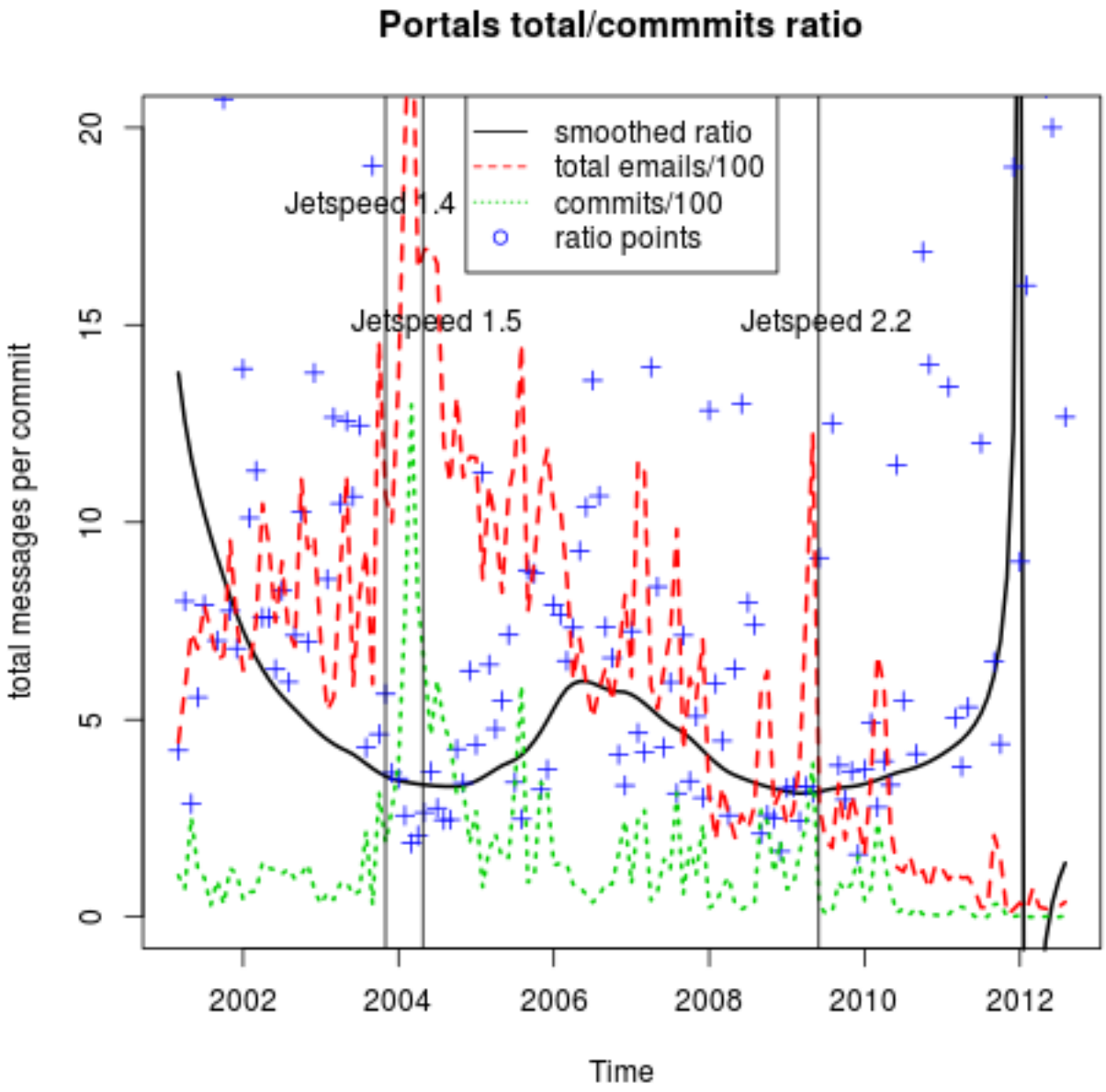


One can see in the devel/commits ratio that the development list traffic pattern is similar to the total. This is typical in projects *pushed* by developers. Other projects are more *pulled* by users, see **Lucene** and **Hadoop** as two good examples.

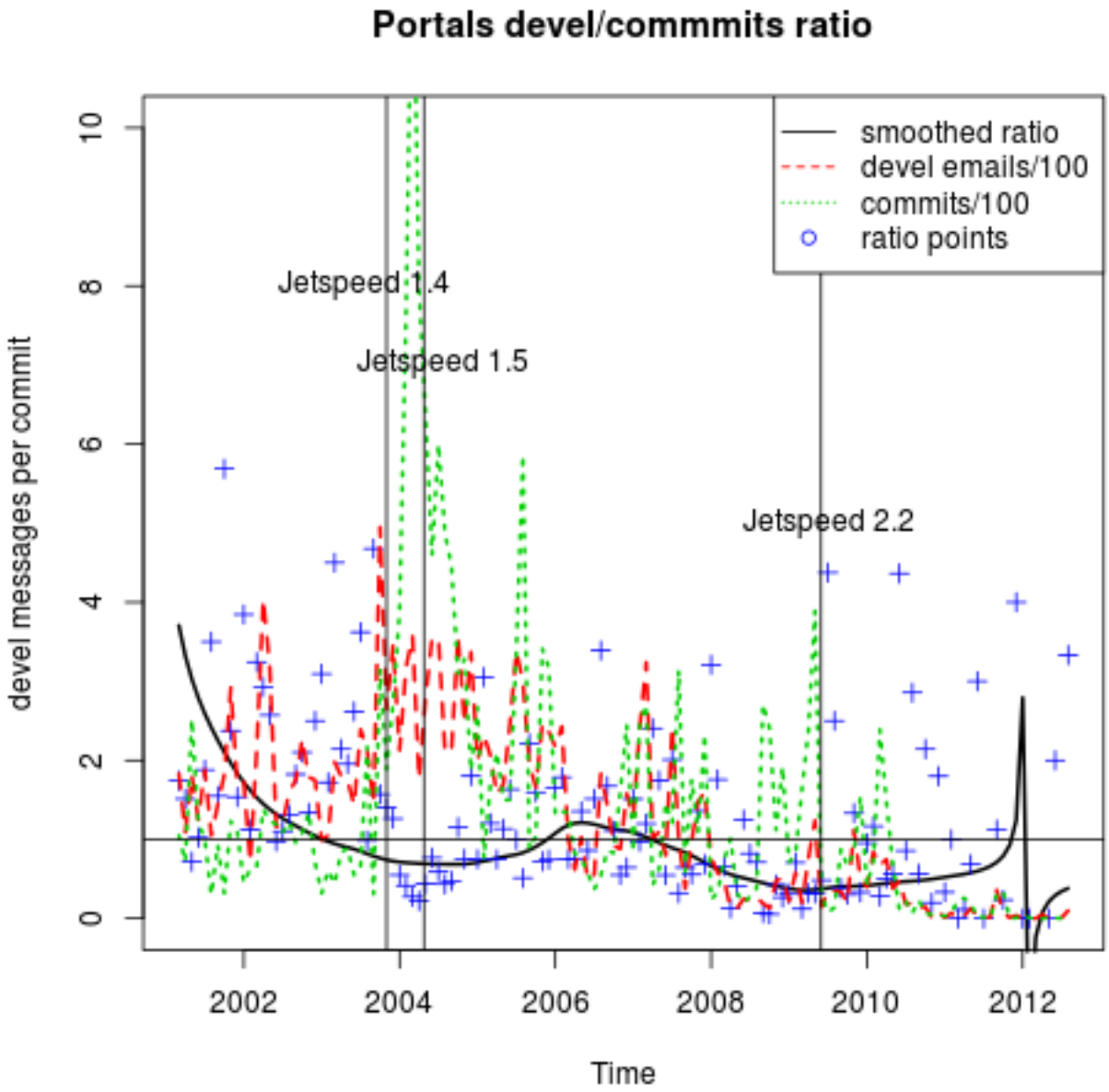
4.10 Portals

The main product of **portals** used to be **jetspeed**, initially based on **turbine**. The vision of **jetspeed** is to reproduce the concept of graphic window in the web. Initially this was done with ad-hoc APIs, but JSR-168 was created to make a standard (PortletAPI).

The reference implementation of this API, **pluto**, falls also under this project.



The evolution of the project does not look good in 2012: the shape of the ratio since 2011 onwards is an artifact of the smoothing polynomial fitting used, and indicates that the project is at zero or low number of commits in one or more months. This indicates that the project has, at least temporarily, stopped being developed or even maintained. For instance, while in Jun 2011 there was only one commit, there were 69 messages, which leads to a very high ratio.

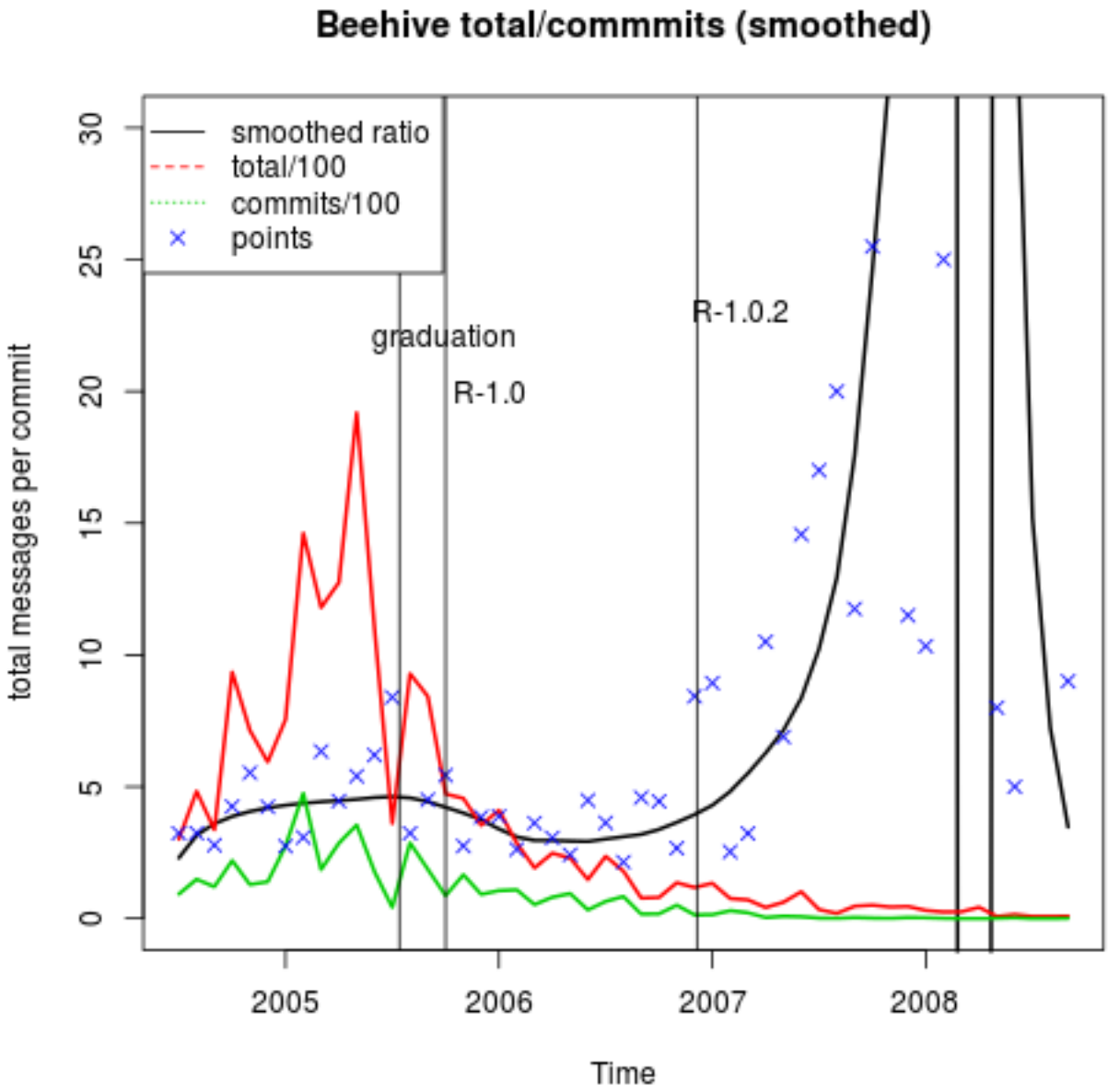


One can see that the devel traffic fell a lot, and has been in no need of coordination, i.e. stagnant or in *maintenance* mode, since about 2009. Even while the number of commits was still great, showing barely no traffic in the developer list and a very reduced number of commits is a signal of a stagnating project.

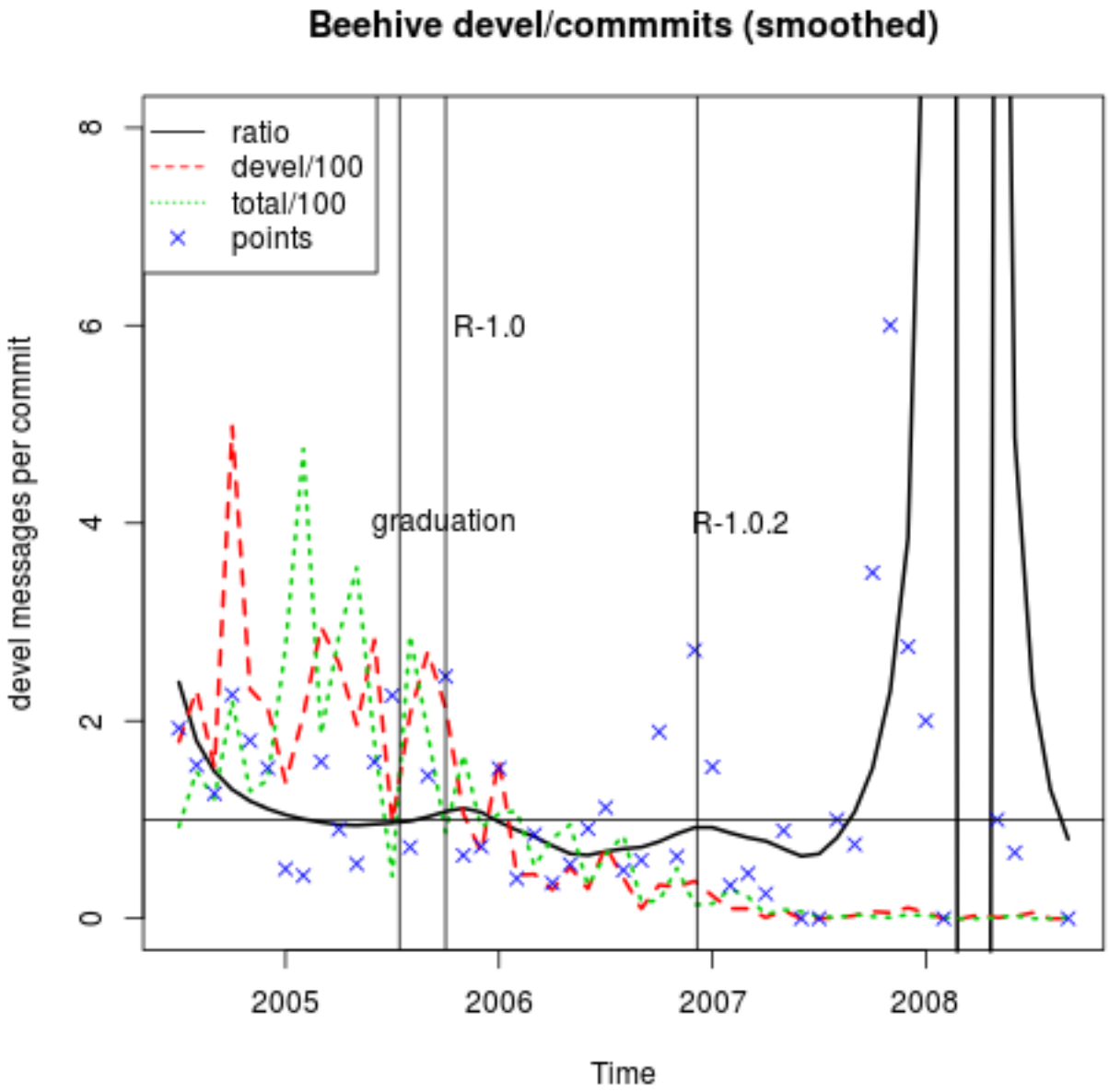
4.11 Beehive, a project in the Attic

Beehive has a lifespan starting around September 2004, when it was donated by BEA to Apache for incubation, to about September 2008. After is was relatively abandoned,

it was the first Apache project to be considered dead and moved to the Attic, in January 2010.
Beehive makes J2RR programming easier by building a simple object model on J2EE and Struts. It used JDK1.5 annotations to express persistence on top of java objects. We have data on the project since 200408 to 200810, which basically is the time passed since it was donated. The ratio of total_messages/commits ranges from very high values (48 and 304) during the initial incubation discussions, to stabilize quickly between 3 and 6 during the next years, and decline later.
We can see a list of events of this project. There is nothing of significance since the release of 1.0.2 and the decision to retire the project to the Attic. In fact the project showed null activity after September 2008.
2005-07-12: **Beehive** graduates from incubator (data prior to this date is traffic from incubator)
2005-07-28: **Beehive** becomes a top-level Apache project.
2005-09-30: **Beehive** releases 1.0
2006-02-13: **Beehive** releases 1.0.1 (includes 90 bug fixes!)
2006-12-04: **Beehive** releases 1.0.2
2008-09-01: No activity at all in the project
2010-01-11: Apache **Beehive** is retired to the Attic
The sharp peak in the ratio since June, 2007 to 2008 is not due to a sudden increase of the number of exchanges, but rather to a sharp decrease in the number of commits. EJB 3 was being developed at the time, and it rendered mostly redundant Beehive. Quoting from Wikipedia : "*When EJB 3 came around, such simplification was already provided by the EJB specification itself, and Beehive controls were of little further use here.*"
The Attic is a project which is in charge of archiving and managing dormant projects. Once a project shows no activity, and a PMC (Project Management Commitee) is not functional, the mail lists are closed to ease the moderation burden, and its web site is marked so that people know that the project is no longer being actively developed. If people wants to ask questions about the code base, or try to gather a new community and revivify the project, Attic is the place to ask.
Beehive was the first project to get archived in the attic. A number of projects have followed suite. The reason why we added beehive to the study is because it was coming from a donation and it never attained the level of success that would have keeped it under active development, or at least maintenance.



So the project shows a moderate initial activity, with values of the messages/commits ratio that are not so big, to quickly stagnate and die from 2006 onwards. The spikes of activity after 2007 result from a very low commit count. They bring an issue with our metric: its should not be computed for very small commit counts, say less than five per month. In other words, it is only valid for active projects.
Commits from April 2007 onwards are always less than 10 a month, in 2008 less than 3 per month. In the devel list there is less than 10 messages/month since 2007 onwards, except for December 2007, with 11.



A high message/commit ratio is typical of the brainstorming phase of nascent projects. It happens also when the user base grows, with increasing bug reports or feature requests, until new committers are appointed and the projects returns to a stationary state. It happens also in a different scenario, when a user community, no matter how small, keeps pushing information to the lists, but there is nobody to commit it. There is a self-reinforcing tendency, as the fact that committers are abandoning the project will prompt users activity to question what is going on. We can see this process going on in Beehive: after release 1.0.2 there was not much development activity. The ratio went up, with discussions, and then down, and the project went to the Attic.

5 Findings

The ratio of total human generated email traffic and the number of commits forms a time series that behaves as a metric for the selected projects the Apache Software Foundation. As the selected projects spawn the whole rank of sizes, technologies and ages, it looks like this is a useful metric for assessing the health and maturity of an Open Source project.

- One can characterise a healthy Apache community as one that has a smooth ratio messages/commits.
- Projects showing little activity or those where a few developers are responsible of most commits will show more noisy ratios than active projects with a balanced groups of active developers.
- Typical historical shape of the ratio starts with high values, as brainstorming sessions or discussion about design happen. For mature projects the ratio goes down to values between 3 and 8.
- When the project matures and is in a maintenance mode the ratio goes steadily down and stabilises around 3-4.
- Some projects show a ratio that has remained relatively low and smooth. For instance, smoothed data from **geronimo** shows a maximum of 6.2 in July 2005 and a minimum of 3.7 in February 2011, and varied slowly. With a similar behavior, **turbine** has stayed between a minimum of 2.5 in March 2009 and 5.4 in May 2003. In both case we think the behavior is due to the fact that those projects have been done by developers for themselves, without heavy spikes of user traffic. **Felix** and **sling** show the same pattern, and we suspect it is for the same reasons. Even **portals**, a similar project, had a strong spike of 13.8 in March 2001, in the heat of the portletAPI discussions, when a big group of external people joined the discussion.
- Other projects, like **portals** or the **jackrabbit** ecosystem, show slightly higher rates during some phases of their development.

- Projects can be considered dormant when the number of commits stays below 10 per month over a sustained period. **Beehive** is a good example. After one such period the activity ceased completely, and it was retired to the Attic two years after the ceasing of activity.

- Apache **Portals** shows in 2012 a pattern similar to the final one of **beehive**, of ceasing commit and total activity, which points to the fact that the portlet API based portals are being abandoned. Seems like HTML 5, Ajax plus jquery, tools like SWT or technologies such as widgets are slowly replacing portals.

- The most successful projects that we have studied are, in the years 1995-2004, **httpd** and **tomcat**, and later **lucene** and the **hadoop** ecosystem.

A number of times that we found an isolated major peak in the ratio between information exchanges and commits, that distorted the result, manual inspection of the messages led to the discovery of automated emails saturating a list:

- automated build failures were not being removed from the count of messages

- There were a number of incidents where a developer used svn lock/unlock, and as other committers tried to commit a big number of mails were generated. Filtering out those emails got results closer to expected.

- Because of a data loss that happened in jakarta.apache.org in September 2003, the email archives were reconstructed, leaving some files behind called 200308-incomplete.gz or 200309-incomplete.gz. Those files were matched by our script, causing duplicity of devel messages and commits with regards to other lists.

- Other peaks observed might have similar causes, as there was a limited time we could dedicate to manually inspect and correct the data set

Noisy behaviour of the information exchange/commits ratio is typical of projects with low activity.
We still have some projects with noisy behaviour and fair levels of activity, notably the first years of **lucene**. In this case, the number of commits falls to very small numbers during some months, causing erratic behavior of the ratio. We need to inspect the data further to know the causes. Probably, as most of the commits during this period were done by the same developer, his limited availability could cause those spikes.
Our ratio should not be computed when the number of commits per period is small, as it has artifacts (see the Beehive section). It is not suitable for the analysis of projects that are not reasonable active, with at least a steady flow of, say, 10 commits per month.

There is a further anomaly that merits comment: the number of commits seems to be accelerating since 2008 onwards, specially in some projects, like the **hadoop** ecosystem, **lucene**, **httpd**, . We can think about some causes that might explain it:

- the use of code review tools ease the ability to contribute patches to projects
- the use of distributed source code management, for the same reason
- an increase to literacy of users, more able to generate patches and send them for evaluation
- smaller granularity of commits, i.e., less information per commit. This could be mediated for the speed of offline commits and the ability to rewrite history coming from tools like git.

One alternative way of thinking about this trend is to think that the developer (coordinating) traffic needed per commit is getting smaller. Several alternative reasons can be hypothesized:

- Cultural learning by committer teams makes them in less need of explicit coordination, so commits *speak by themselves* most of the time, and developer traffic is lessened.
- Committers get progressively employed by related companies, and some or even most of the coordination work takes place off the public development lists. This hypothesis does not imply malevolence: a committer might find easy to use internal channels, or even just talk over lunch about an issue, than to send a formal, explicit message to the devel list.

6 Conclusions

The process of extracting the data from the mail archives required a number of iterations, due to bugs discovered in our scripts. It was performed, also, in an exploratory way, were the analysis of a project led to addition of other projects to the list to be analyzed, for sake of comparison.

The ratio of total information exchanges to commits is an intensive property of projects, in the sense that it is independent of project size and depends on other parameters such as maturity, or ratio of committed users to developers. Other ratios, such as the ratio between developer messages to commits or issues to commits, are also useful to assess the health of the project. We essayed other ratios, but none was as promising as those.

There is a number of questions raised by this work. In particular the trend to have more commits in the last couple of years in several projects. This growth happens in spite of the fact that the developer, or even total, message number is diminishing. We think this trend merits further work. We also would like to understand some other anomalies, like the fact that the number of commits in projects like **lucene** fluctuated heavily during their early years. This fluctuation was more than what one would expect from the project size. **Hadoop**, also shows an anomaly: has a number of messages per commit much higher than the rest of the projects, showing that it is still a young ecosystem and has much way to go. Strangely, the number of developer messages per commit is rather small. This suggest a development model based in code reviews and patches attached to issues, where code comes from outside of the project via issues, rather than after public discussion between a set of core developers on list.

7 Bibliography

[**ASF1**] The Apache Software Foundation: How it Works
http://www.apache.org/foundation/how-it-works.html Consulted on Jul 2012
[**COA1**] Ken Coar, Release dates (apache-announce messages), message sent to httpd-dev
http://mail-archives.apache.org/mod_mbox/httpd-dev/199905.mbox/%3C3741B9EA.DF4D133@Golu
[**DAV1**] James Duncan Davidson, Rules for Revolutionaries, sent to tomcat-dev@jakarta.apache.org and jakarta@apache.org on 2000-01-13
http://incubator.apache.org/learn/rules-for-revolutionaries.html
[**ERE**] OSBC: No Jerks Allowed!
Justin Erenkranz (President, Apache Software Foundation) Session in Open Source Business Conference, 2010
http://www.erenkrantz.com/apachecon/OSBC%20-%20No%20Jerks%20Allowed.pdf
[**FIE1**] A Case Study of Open Source Software Development: The Apache Server
Audris Mockus, Roy T. Fielding. James Herbsleb. ICSE 2000 Limerick Ireland
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=870417
[**FIE2**] Two Case Studies of Open Source Software Development: Apache and Mozilla
Audris Mockus, Roy T. Fielding. James Herbsleb. ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3, July 2002, Pages 309–346.
http://dl.acm.org+citation.cfm?id=567793.567795$coll=DL$dl=ACM$CFID=111446298$CFTOKEN
[**HAR1**] An Entropy-Based Measure of Software Complexity
Warren Harrison - Software Engineering, IEEE Transactions on, 1992
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=177371
[**KIE1**] Self-organization in open source to support collaboration for innovation: the Drupal case
Mixel Kiemen, VUB. ISPIM 2011
http://www.mixel.be/files/pdf/ISPIM2011.pdf
[**MAZ1**] Rules for Revolutionaries, sent to community@apache.org on 2002-11-07 13:33:28
http://marc.info/?l=apache-community$m=105712356508947$w=4
[**MAZ2**] Email in private list, quoted by permission via private email 2012/09/04
[**PRI1**] Time, Structure and Fluctuations. Nobel Lecture, 1977. Ilia Prigogine
http://www.nobelprize.org/nobel_prizes/chemistry/laureates/1977/prigogine-lecture.pdf
[**PRI2**] Self-Organization in Nonequilibrium Systems

[**RON1**] A model based on information entropy to measure developer turnover risk on software project
J Rong, L Hongzhi, Y Jiankun, F Tao, Z Chenggui... - 2009 - IEEE Explore
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5234813
[**RON2**] An Approach to Analysis and Measurement of the Stability of Web Community Based on
Dissipative Structure Theory and Information Entropy
Jiang Rong, Liao Hongzhi, Yu Jiankun, Zhang Dehai, Chen Lihua, Sun Yafei - 2009 - IEEE Explore
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5365461
[**SCH1**] Private list email, quoted by permission given to the author by Joe Schaeffer via private email, on 2008/12/19.

8 Appendices

The next appendices list the code that was used to obtain this results. The Apache public mailing lists are archived in
http://mail-archives.apache.org/mod_mbox/
The archives include the ability to download monthly mbox files per list, but we used shell access to people.apache.org, available for ASF committers and members, to process the files locally. The archives are gzipped mbox files, accessibles at /home/apmail/public/archives/<project>.apache.org/<list>/<year><month>.gz

8.1 Appendix A. Mbox emails processing

A shell script, get_data_from_apache.sh, used for the processing of mbox emails and writing a file containing lists of *comma-separated values* tables per project. A number of regular expressions are used to select, or filter out, automatically generated messages or messages belonging to a different kind of traffic.

```
#!/bin/bash
projects="beehive hadoop avro hbase hive pig httpd apr geronimo lucene
    tomcat turbine felix jackrabbit portals sling spamassassin"
BASE="/home/apmail/public-arch"
#BASE="/home/apmail/public-arch/incubator.apache.org"
#incubprojects=$(ls -d $BASE/*-* | sed -e "s@.*/@@" -e "s@-.*@@" | sort -u
    )
#for project in $incubprojects;
for project in $projects;
do
    echo "===$project==="
    echo "year,month,build,issues,commits,wiki,devel,user,total";
    years=$(ls $BASE/$project.apache.org/{announce,bugs,*issues,*commits,
        cvs,scm,*dev,*general,*user*,wiki-changes}/*.gz | sed -e "s@.*/@@"
        -e "s@[0-9][0-9][^0-9]*\$@@" | sort -u)
    for year in $years;
    do
        for month in {01..12};
        do
            i=$(zgrep -E "^Subject:.*(\[Bug .*\]|\[jira\])" $BASE/$project
                .apache.org/{bugs,*issues,*dev}/$year$month.gz 2>/dev/null
```

```
            | wc -l);
        c=$(zgrep -E "^Subject: ... commit" $BASE/$project.apache.org
            /{*commits,scm,cvs,*dev}/$year$month.gz 2>/dev/null | wc -l
            );
        w=$(zgrep -E "^Subject:.*(_|\[.*)Wiki" $BASE/$project.apache.
            org/{*commits,scm,cvs,*dev,wiki-changes}/$year$month.gz |
            wc -l);
        b=$(zgrep -E "^Subject:.*(\[GUMP.*\]|Jenkins |Build.*Hudson)"
            $BASE/$project.apache.org/*dev/$year$month.gz 2>/dev/null |
             wc -l);
        d=$(zgrep -h "^Subject: " $BASE/$project.apache.org/*dev/
            $year$month.gz 2>/dev/null | grep -Ev "^Subject: (...
            commit|... .*lock|.*(\[Bug .*\]|\[jira\])|\[GUMP.*\]|.*
            Jenkins|Build.*Hudson)" | wc -l);
        u=$(zgrep "^Subject: " $BASE/$project.apache.org/{announce,*
            general,*user*}/$year$month.gz 2>/dev/null | wc -l);
        t=$(zgrep -h "^Subject: " $BASE/$project.apache.org/{announce,
            bugs,*issues,*commits,scm,cvs,*dev,*general,*user*,wiki-
            changes}/$year$month.gz 2>/dev/null | grep -Ev "^Subject:
            (... .*lock|\[GUMP.*\]|.*Jenkins|Build.*Hudson)" | wc -l);
        echo "$year,$month,$b,$i,$c,$w,$d,$u,$t";
    done;
  done #>commits-mails-$project.txt
done
```

8.2 Appendix B. Breaking result into per-project files

Trivial python script for breaking the results per project. It is used to break the result of the previous script into one CSV file per project, to import it into R with the next script

```
import re
all=re.compile("===\n?",re.M).split(open("mails.csv").read())
for i in range(1,len(all),2):
    open("mails-"+all[i]+".csv","w").write(all[i+1])
```

8.3 Appendix C. Statistic calculations

RScript to generate the intensive properties from the extensive ones and smoothing. It reads the data, turning it into a data.frame. It is later processed so that it takes the form of a ts (time series) with the leading and trailing zero months stripped out, and a series of derived ratios and smoothed ratios are computed and added. For smoothing we used loess, a polinomial estimate.

```
#!/usr/bin/Rscript --save
# R process for a project
projects<-c("httpd", "apr", "spamassassin", "turbine", "tomcat", "geronimo
    ", "hadoop", "avro", "hbase", "hive", "pig", "lucene", "beehive", "
    felix", "jackrabbit", "portals", "sling") # TODO more?
f<-function(filename) {
    print(filename)
    proj=read.csv(file=filename) # created a data frame with info
    proj$start<-I(c(proj$year[proj$commits!=0][1], proj$month[proj$commits
        !=0][1]))
    proj$end  <-I(c( max(proj$year[proj$commits!=0]),
                     max(proj$month[proj$commits!=0 & proj$year==max(
                         proj$year[proj$commits!=0])])))
    proj1<-data.frame(
                total   = window(x=ts(proj$total   ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                commits = window(x=ts(proj$commits,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                devel   = window(x=ts(proj$devel   ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                issues  = window(x=ts(proj$issues  ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                user    = window(x=ts(proj$user    ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                year    = window(x=ts(proj$year    ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2]),
                month   = window(x=ts(proj$month   ,frequency=12,start=c(
                    proj$year[1],1)),start=proj$start[1:2],end=proj$end
                    [1:2])
                )
    #print(proj1)
    proj1$start  = proj$start[1:nrow(proj1)]
    proj1$end    = proj$end[1:nrow(proj1)]

    proj1$tc<-ts(proj1$total/proj1$commits, frequency=12,start=proj1$start
        ,end=proj1$end)
    proj1$dc<-ts(proj1$devel/proj1$commits, frequency=12,start=proj1$start
        ,end=proj1$end)
    proj1$tb<-ts(proj1$total/proj1$issues, frequency=12,start=proj1$start,
        end=proj1$end)
```

```
    proj1$db<-ts(proj1$devel/proj1$issues, frequency=12,start=proj1$start,
        end=proj1$end)
    proj1$bc<-ts(proj1$issues/proj1$commits, frequency=12,start=
        proj1$start,end=proj1$end)
    lc <- loess(commits ~ x, data.frame(commits=proj1$commits, x=1:length(
        proj1$commits)), span=0.50)
    lt <- loess(total ~ x, data.frame(total=proj1$total, x=1:length(
        proj1$total)), span=0.50)
    ld <- loess(devel ~ x, data.frame(devel=proj1$devel, x=1:length(
        proj1$devel)), span=0.50)
    li <- loess(issues ~ x, data.frame(issues=proj1$issues, x=1:length(
        proj1$devel)), span=0.50)
    proj1$ltc<-ts(lt$fitted/lc$fitted, frequency=12,start=proj1$start)
    proj1$ldc<-ts(ld$fitted/lc$fitted, frequency=12,start=proj1$start)
    proj1$lic<-ts(li$fitted/lc$fitted, frequency=12,start=proj1$start)
    proj1
}

for(p in projects) {
    assign(p,f(paste('mails-',p,'.csv',sep='')))
}
```

8.4 Appendix D. Plotting

A lot of small plotting scripts are used to generate the graphics. As an example we post here a couple of those. Note that both start by loading the data set, stored in the given path in the author computer .

- The total/commits one for the Sling ecosystem is one of the more complex, because it computes the sum of all three projects traffic and commit number.

```
load(file="/home/sgala/Documentos/master/TFM/data/.RData")
ts.plot(felix$ltc,jackrabbit$ltc,sling$ltc,col=1:10, lty=1:10,lwd=2, ylim=
    c(0,30), main="Sling ecosystem total messages/commits (smoothed)")
a=window(x=sling$total,extend=TRUE,start=jackrabbit$start[1:2],end=
    jackrabbit$end[1:2])
a[is.na(a)]<-0
b=window(x=felix$total,extend=TRUE,start=jackrabbit$start[1:2],end=
    jackrabbit$end[1:2])
b[is.na(b)]<-0
lines((jackrabbit$total+a+b)/100,col=4,lty=4,lwd=2)
a=window(x=sling$commits,extend=TRUE,start=jackrabbit$start[1:2],end=
    jackrabbit$end[1:2])
a[is.na(a)]<-0
```

```
b=window(x=felix$commits,extend=TRUE,start=jackrabbit$start[1:2],end=
    jackrabbit$end[1:2])
b[is.na(b)]<-0
lines((jackrabbit$commits+a+b)/100,col=5,lty=5,lwd=2)
abline(v=2006+(4-1)/12+15/365)
text(2006+(4-1)/12+15/365,14,"JR1.0")
abline(v=2007+(4-1)/12+15/365)
text(2007+(4-1)/12+15/365,14,"JR1.3")
abline(v=2008+(1-1)/12+15/365)
text(2008+(1-1)/12+15/365,14,"JR1.4")
abline(v=2009+(8-1)/12+15/365)
text(2009+(8-1)/12+15/365,14,"JR1.6")
abline(v=2010+(1-1)/12+30/365)
text(2010+(1-1)/12+30/365,11,"JR2.0")
abline(v=2010+(12-1)/12+15/365)
text(2010+(12-1)/12+15/365,10,"JR2.2")
abline(v=2007+(09-1)/12+15/365)
text(2007+(09-1)/12+15/365,2,"donation Sling")
abline(v=2009+(05-1)/12+6/365)
text(2009+(05-1)/12+6/365,2,"Sling-5")
abline(v=2011+(03-1)/12+28/365)
text(2011+(03-1)/12+28/365,2,"Sling-6")
legend(x="topleft",legend=c("felix","jackrabbit","sling","ecosystem msg
    /100","ecosystem commits/100"),lty=1:length(projects),col=1:length(
    projects))
```

- The **hadoop** ecosystem project plot containing *Hadoop total/commits (smoothed)*:

```
load(file="/home/sgala/Documentos/master/TFM/data/.RData")
ts.plot(hadoop$ltc,avro$ltc,hbase$ltc,hive$ltc,pig$ltc,col=1:10, lty=1:10,
    lwd=2, ylim=c(1,30),main="Hadoop total/commmits (smoothed)")
points(hadoop$tc,col=1,pch=1)
points(avro$tc,col=2,pch=2)
points(hbase$tc,col=3,pch=3)
points(hive$tc,col=4,pch=4)
points(pig$tc,col=5,pch=5)
abline(v=ts(2008+9/12,frequency=12, start=hadoop$start))
text(2008+9/12,27,"  HBase 0.18.0")
abline(v=ts(2009+7/12,frequency=12, start=hadoop$start))
text(2009+7/12,30,"  1st restruct")
abline(v=ts(2010+5/12,frequency=12, start=hadoop$start))
text(2010+5/12,25,"  2nd restruct")
legend(x="topleft",legend=c("hadoop","avro","hbase","hive","pig"),lty=1:
    length(projects),col=1:length(projects))
```

8.5 Appendix E. License

8.5.1 Documentation

This Report can be distributed under the Creative Commons Attribution 3.0 Unported (CC BY 3.0) License (http://creativecommons.org/licenses/by/3.0/), included below.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

"Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

"Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

"Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

"Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

"Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

"Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

"You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

"Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

"Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic

medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
to Distribute and Publicly Perform Adaptations.
For the avoidance of doubt:

Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.
The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(b), as requested.

If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv), consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the

credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those

licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not

intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

8.5.2 Software

The accompanying scripts can be distributed under the Apache License 2.0 , included below

Copyright 2012 Santiago Gala-Perez

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
.
See the License for the specific language governing permissions and
limitations under the License.