Intensive Metrics for the Study of the Evolution of Open Source Projects: Case studies from Apache Software Foundation projects

Santiago Gala-Pérez
Apache Software Foundation
sgala@apache.org

Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
grex@gsyc.urjc.es

Jesús M. González-Barahona
GSyC/LibreSoft
Universidad Rey Juan Carlos
jgb@gsyc.urjc.es

Israel Herraiz
Universidad Politécnica de Madrid
israel.herraiz@upm.es

***Abstract*—Based on the empirical evidence that the ratio of email messages in public mailing lists to versioning system commits has remained relatively constant along the history of the Apache Software Foundation (ASF), this paper has as goal to study what can be inferred from such a metric for projects of the ASF. We have found that the metric seems to be an intensive metric as it is independent of the size of the project, its activity, or the number of developers, and remains relatively independent of the technology or functional area of the project. Our analysis provides evidence that the metric is related to the technical effervescence and popularity of project, and as such can be a good candidate to measure its *healthy* evolution. Other, similar metrics -like the ratio of developer messages to commits and the ratio of issue tracker messages to commits- are studied for several projects as well, in order to see if they have similar characteristics.**

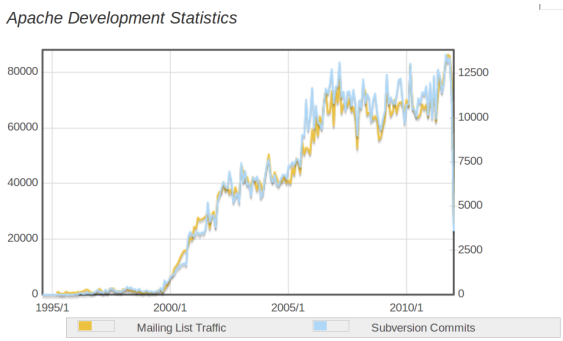

Fig. 1. Recent snapshot of the commits and mailing list messages for all ASF projects. Source: http://www.apache.org/dev/stats/

I. Introduction, Motivation and Goals

On Nov 29, 2008, Joe Schaeffer, an Apache Software Foundation (ASF) member and at that time employee for infrastructure work at the ASF, sent an email to the private ASF member discussion mailing list, which informed about " *[...] something I[n] M[y] O[pinion] interesting about the ASF: the fact that the number of commits and the number of mailing list posts have (sic) grown in linear relationship to one another over the years. [...] Note how well the charts for total commits and total emails align even though they're graphed against 2 different scales.*" In the subsequent discussion on the importance and meaning of the result, Joe added that "*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*". Since then, a dynamic web site offers a graphical overview of the evolution of these two factors of which Figure 1 provides a recent snapshot.

This paper aims to study in depth this metric, and what can be inferred from its use on software projects, especially if they can describe the state and the evolution of software development projects. Our more ambitious goal is to identify what are known as intensive properties. In the natural sciences, an intensive property is not dependent on the size of the system or the amount of material in the system: it is scale invariant. By contrast, an extensive property is one that is additive for independent, noninteracting subsystems, so directly proportional to the amount of material in the system [1].

The identification of intensive metrics would be especially suited for comparing projects of different sizes. Most metrics of projects are difficult to compare since they depend on the size of the project. This is additionally complicated in the software domain because successful projects commonly undergo transitions in size of two orders of magnitude during their development.

The metric that we will study in this paper relates the communication flow with the development activity. Our assumption is that if this ratio is an intensive metric, we will be able to infer information from this metric from a software evolution point of view. Thus, we expect to have a metric that varies with the maturity, technology or community composition of a project, but not, or at least not much, with its size.

Our focus is in particular on free software projects, and among them, those projects 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 mainly in the study of the ratio between email traffic and number of source code commits for a selection of projects in the ASF.

Hence, the contributions of this paper can be summarized as follows:

- It presents the concept of invariant metrics applied to software projects
- It introduces the usefulness of such metrics for the study of software evolution

The structure of this paper is: Next, the methodology is presented, including the identification of the data sources that have been used. The following section contains the analysis of several projects of the ASF. Lessons learned are presented and discussed in section IV, while Section V tries to find other ratios that may serve as invariant metrics. The following section contains related research on the topic. The paper concludes with sections on threats to validity, hints about how to reproduce it and conclusions and further research.

II. Methodology

Studying the communication flows in a project is a tricky task, as the number of channels in use is vast and may depend on role, circumstance, task and other issues. However, in some free software projects, such at is the case of the ASF, there is a strong cultural trait that all decision-making should happen through mailing lists. Justin Erenkrantz, one of the ASF board members, states that *"all technical decisions about a project are made in public (mailing lists)"* and quotes a famous Roy Fielding mantra where decisions have to occur: "*If it didn't happen on-list, it didn't happen*"[1]. Therefore, synchronous media such as IRC, AIM, Jabber, etc. are not suitable for decisions, and if they are important for any discussion, the conversations are to be included in a mail message.

The fact that mailing lists are used at the ASF as a central coordination point for projects has another advantage for our research, as all interactions with the main development systems are submitted to lists as well. So,

- issue trackers will send an email to specific lists for every issue opened, commented, modified or closed,
- code review tools, automated builds -specially when the build gets broken-, or wiki page edition are notified per e-mail,
- and meta-data from commits to the source code versioning system are reflected in the project lists allowing to discuss the change.

The ASF makes all its public mailing list archives available using mod-mbox, an Apache Web Server module that serves email archives in mbox format[2]. The archives are available at a web site[3], are indexed by list, in monthly archives.

A Shell script has been used for processing the mbox email files. The result is a file containing lists of *comma-separated value* (CSV) tables per project. A number of regular expressions are then used to select, or filter out, automatically generated messages or messages belonging to the various types of traffic. Then, a Python script has been created to break the results into a single CSV file per project. The output is used with the R statistical environment: the data is read, transformed into data frames. Later on, it is processed so that it takes the form of a time series (ts) with the leading and trailing months without activity stripped out, and a series of derived and smoothed ratios are computed and added. For smoothing we have used *loess*, a polinomial estimate [2]. Finally, a lot of small plotting scripts are used to generate the graphics shown in this paper.

A number of times, 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 certain types of messages that have been filtered out:

- Automated build failures.
- Incidents where a developer used `Subversion` lock/unlock, and as other committers tried to commit a big number of mails were generated.
- 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 developer mail messages and commits with regards to other lists.

III. Analysis

We have selected a group of projects from the ASF and have performed an exploratory analysis of them. The projects have been chosen on the basis that they differ on issues such as size (in terms of source code and number of developers and users), age, endeavor (end-user or developer oriented), technology and maturity, among others. We have even included some projects of which we know that there is low activity or that have been moved to the Attic, where Apache projects get archived if their development ceases.

The amount of communication exchange in the projects under study ranges from the 11,500 emails/month that Apache `Hadoop` generated in Aug 2012 to projects such as `Turbine`, that has generated a maximum of 175 messages per month during the last 5 years.

In addition, we will analyze what we have called ecosystems, closely coupled group of projects that typically consist of an original project that sprout children as it grew, or projects that depend on a common platform. Examples of these are:

- `httpd` created `apr` as its runtime, and they remain related.
- `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 ASF and have an ample common committer pool.
- `Hadoop` is a *Big Data* framework. Being currently the most popular ASF project, there are a lot of projects spawning off `Hadoop`: `HBase`, `Avro`, `Hive`, `Pig`, `Zookeeper`, and `Whirr`.

[1] http://www.erenkrantz.com/apachecon/JASIG%20-%20No%20Jerks%20Allowed.pdf

[2] http://www.qmail.org/man/man5/mbox.html

[3] http://mail-archives.apache.org/mod_mbox/

A. Apache Httpd

The Apache `httpd` server is the project that originated the ASF as such. 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). The project is still being actively developed, and has generated a maximum of 5,200 messages in Apr 2002; currently, the activity lies around 1,000 messages/month. Its current size is 156 KSLOC.

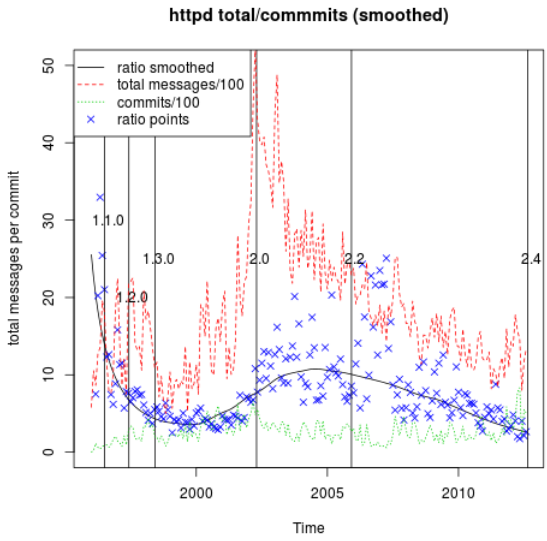

Fig. 2. httpd measures. Main releases are annotated with vertical lines.

Figure 2 shows that the total traffic started relatively high, with spikes of 2,000 messages/month, but went down to around 1,000 by 1999. Commits started growing, to around 100 at the beginning of 1997, and up to 300 around 1999-2000. This reflects the maturation 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 2.2. 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 stabilize 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. After this, the ratio of commits to total messages has been going down steadily, and even the recent release of 2.4 didn't suppose a major change.

B. Apache Portable Runtime (APR)

`apr` is a 66 KSLOC library that eases the porting of `httpd` to different platforms. It is also used by `Subversion` and other projects. Its traffic patterns are very different of the ones in `httpd`, with peaks of total traffic and commit traffic not much related with the releases or activity of `httpd`, specially as it matured.

Figure 3 shows that the activity is polarized around the releases, and fairly decoupled from `httpd`, although 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.

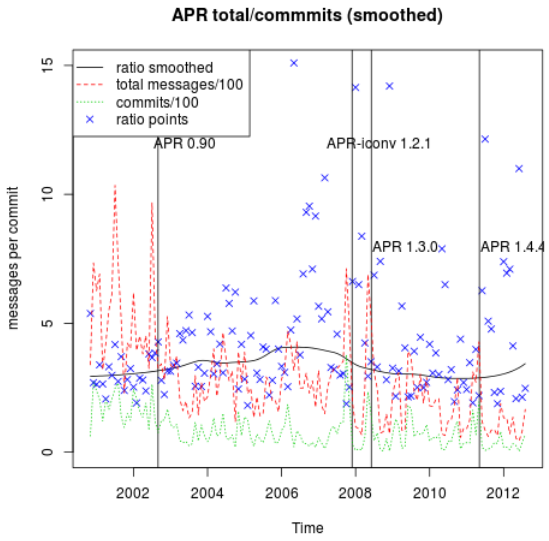

Fig. 3. apr measures. Main releases are annotated with vertical lines.

`apr` is what we have identified as a developer oriented project, as it has little impact outside the developer community. We have analyzed if the pattern of the traffic in the developer mailing list is different from the shown above which includes the total traffic, and have found that they have a similar shape.

C. Apache Lucene

`Lucene` is a full text indexing and search solution. Its current size is 414 KSLOC. It is showing a lot of activity in the last years, because of the trend of *Big Data* processing. The ratio *total_messages/commits* consistently grew until 2008, because the rise in the number of mail messages was not followed with a similar increase in the number of commits (see Figure 4). However, even though it has seen a tremendous increase of traffic from 2009 onwards, the number of commits has been growing at a faster pace, outgrowing mail messages and resulting in a decreasing ratio with values under 10.

D. Apache Turbine

`Turbine`, an early web framework in Java, is part of Apache Jakarta. It is a project that was very active around 2000, but got stagnant due to endless refactoring that broke backwards compatibility and the rising popularity of alternate approaches, such as Ruby on Rails. The latest `Turbine` release had 41 KSLOC.

Figure 5 shows that the total email traffic has gone down since 2001-02. In January 2006 the number of messages was 26, down from 1,865 in January 2002, a two orders

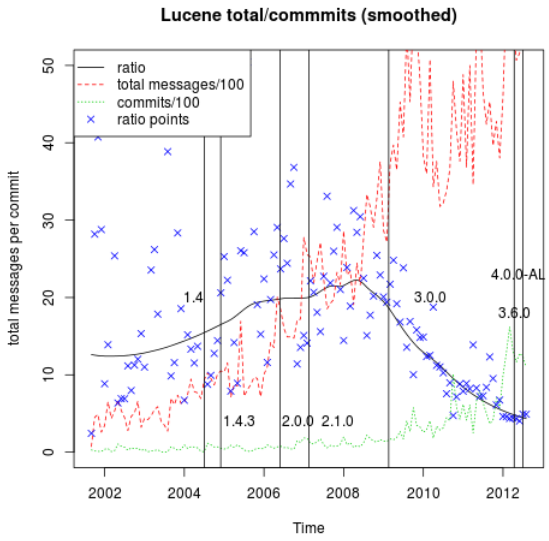

Fig. 4. Apache Lucene measures. Main releases are annotated with vertical lines.

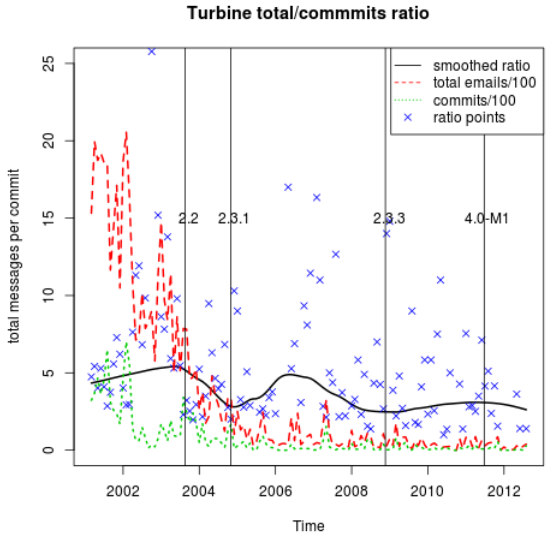

Fig. 5. Apache Turbine measures. Main releases are annotated with vertical lines.

of magnitude of decrease. `Turbine` 3 never got released. Developers moved to other alternatives at the time, and the source code was refactored several times while component functionality and APIs never became stable but keep changing and breaking dependent projects such as `Jetspeed`.

However, the smoothed ratio has remained almost constant during the whole lifetime of the project. 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. The project is mostly stable, and still alive.

E. Apache Tomcat

`Tomcat` was one of the first projects in Apache involving a code donation. In `Jakarta` there was a project called `JServ`, developed to support the Servlet API 2.0. Sun Microsystems, at the time, had a product called `JWSDK`, a reference implementation of the Servlet API. Its current size is 213 KSLOC.

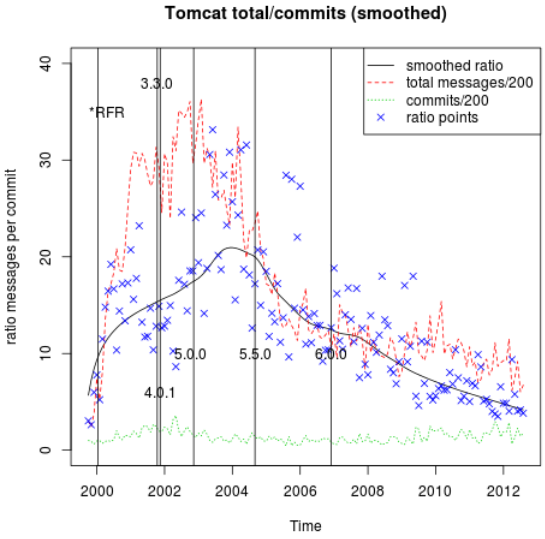

Fig. 6. Apache Tomcat measures. Main releases are annotated with vertical lines.

When `Tomcat` released the 3.2 version, there was a strong discussion about its future evolution[4]. As a consequence, during a relatively long period, `Tomcat` released in parallel 3.2.X, 3.3.X and 4.0.X versions, developed by different developer groups. While this feature can be thought to cause the spike in developer mail 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 Figure 6 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 (codenamed `Catalina`) and not in the approach from 3.3.

Some releases are signaled in Figure 6, including the RFR marker, which outlines the date of the *Rules for Revolutionaries* email message. After this message and in the following big discussion, the number of messages grew up to values around 6,100/month, and later 7,200 around the release of 5.0.0.

It should be noted, that free software developers and users typically install and update packages using the distribution package repository as new versions are released. They are also typically more familiar with cultural issues: where to

[4]Craig R. McClanahan recounts the history in a message, see http://mail-archives.apache.org/mod_mbox/tomcat-dev/200001.mbox/raw/%3C387BF46C.F8302840%40mytownnet.com%3E

find documents, how to proceed when a bug is found, etc. In contrast, typical Windows developers *discovering* Java in the early 2000's would be forced to download, install and upgrade on their own, and being less aware of cultural conventions. These type of developers make more *noise* in lists asking for support. The arrival of Windows developers, attracted 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, that we have not found in the developer list.

F. Apache Jackrabbit

`Jackrabbit` is the reference implementation of JSR-170, a Java Content Repository (a standard for managing web content). This JSR was pushed forward mostly by Day Software, a company who donated two other components that use `Jackrabbit` to the ASF as well: the OSGi service platform, `Felix`, and a web framework, `Sling`. The three projects are closely related and will be studied together in Figure 7.

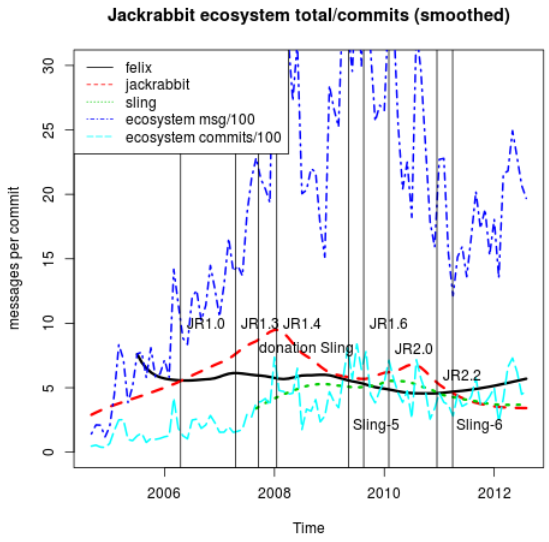

Fig. 7. Apache Jackrabbit measures. Main releases are annotated with vertical lines.

The ecosystem has attained success, with `Jackrabbit` having processed 1,500 messages in July 2009, `Felix` 2,168 in September 2009, and `Sling` 1,688 in July 2008, and 1,616 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 other languages, 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.

G. Apache Hadoop

The `Hadoop` ecosystem is seeing development at a frantic pace since 2007. With time, a number of subprojects spawned out of Hadoop, which already amounts for 1.27 MSLOC, being the most significant:

- `Avro`, a data serialization system,
- `HBase`, the Hadoop distributed, scalable, big data store database,
- `Hive`, a data warehouse system for `Hadoop`, and
- `Pig`, a platform for analyzing large data sets.

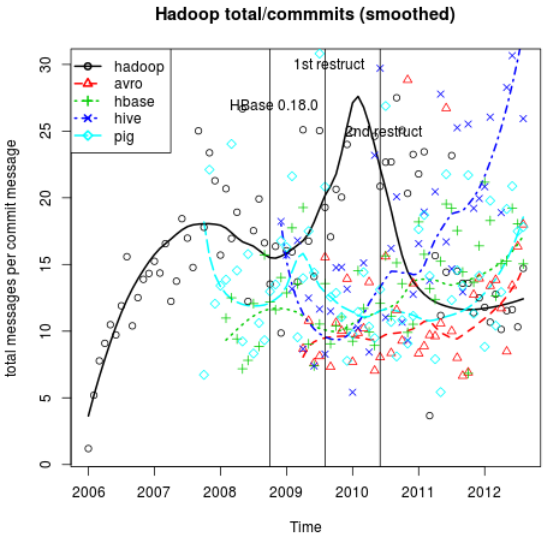

Fig. 8. Hadoop and subprojects measures. Main releases are annotated with vertical lines.

All the projects are seeing very high ratios of messages to commits, above 10 most of the time (see Figure 8). When inspecting the traffic of the mailing lists in detail, we have found that most of the communication, almost half of it, appears to be related to the issue tracking system, while the developer traffic share is very low. This makes us assume that development is coordinated, with users as well, through the issue tracker rather than using the developer list.

H. Apache Geronimo

`Geronimo` is a certified JavaEE Application Server. It was born when a group of developers involved in several JavaEE components got in a conflict with `JBoss` and asked the ASF to release a certified JavaEE Application Server. It was adopted by IBM as a community edition of their `WebSphere Application Server`. The current size of `Geronimo` is 370 KSLOC.

As can be seen from Figure 9, `Geronimo` shows a uniform ratio of traffic to commits. As the project matures, there is less developer activity.

I. Apache Spamassassin

Written in Perl, it is a preexisting community that joined the ASF in 2004. While most Java developers are involved in

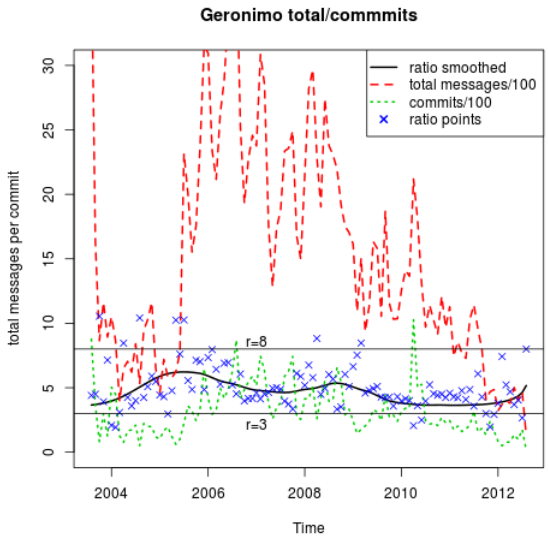

Fig. 9. Geronimo measures.

several projects, and some of them even in `httpd` or `apr`, `Spamassassin` developers were more isolated in the initial phases of the project.

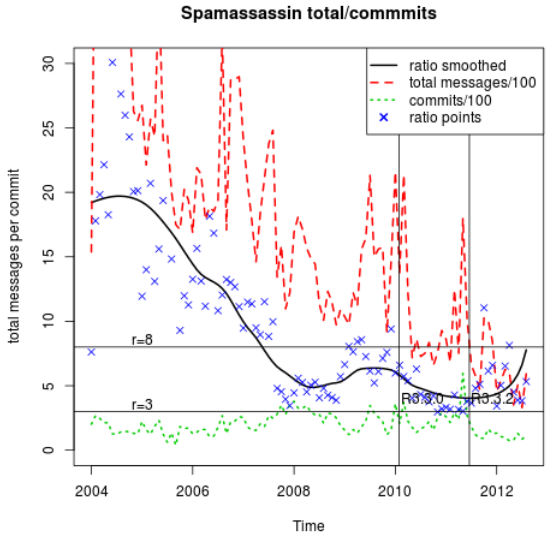

Fig. 10. Spamassassin measures. Main releases are annotated with vertical lines.

Spamassassin is a mature application used in production with 54 KSLOC of code. Figure 10 provides information on the total activity against commits; 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.

J. Apache Portals

`Portals` is an ecosystem of software projects, where `Jetspeed` is its main product with around 202 KSLOC.

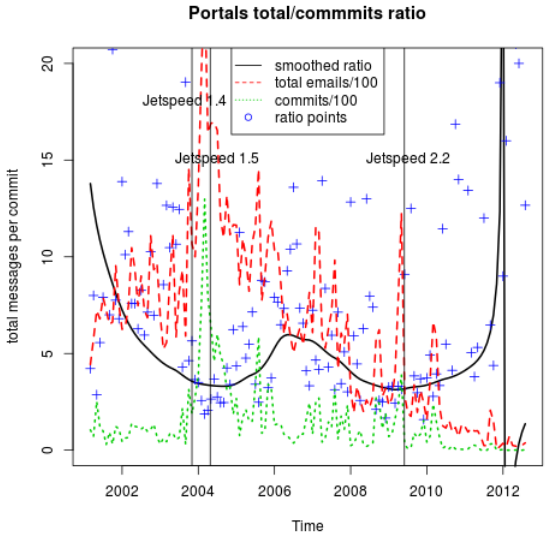

Fig. 11. Jetspeed measures. Main releases are annotated with vertical lines.

According to Figure 11, Jetspeed is stagnating in early 2012: the traffic has fallen consistently and so have the number of commits as well, to the point where the number of commits is very low. The ratio shows a discontinuity when no commits occurred during a month.

K. Apache Beehive

`Beehive` makes J2RR programming easier by building a simple object model on J2EE and Struts. `Beehive` is the first project that got retired into the Attic, where Apache projects get archived. It serves as a good example of what happens to a project that gets abandoned by developers. `Beehive` has a lifespan starting 2004, when it was donated by BEA to Apache for incubation, to about September 2008. After that is was relatively abandoned, and moved to the Attic in January 2010. The latest release had 88 KSLOC.

As it can be seen from Figure 12, the ratio of total_messages/commits is almost constant until release 1.0.2. Then, the project is abandoned and even some months with no commits result in discontinuities in the ratio.

IV. Lessons Learned and Discussion

In this section we summarize and discuss the lessons that we have learnt from measuring the ratio on several projects, with a special focus on assessing the appropriateness of our metric as an intensive metric and what information it provides.

A first observation from the projects under study is that that we can characterize a *healthy* Apache project as one that has a smooth ratio messages/commits. Those projects showing little activity or those where a few developers are responsible of most commits will show more noisy ratios than active projects

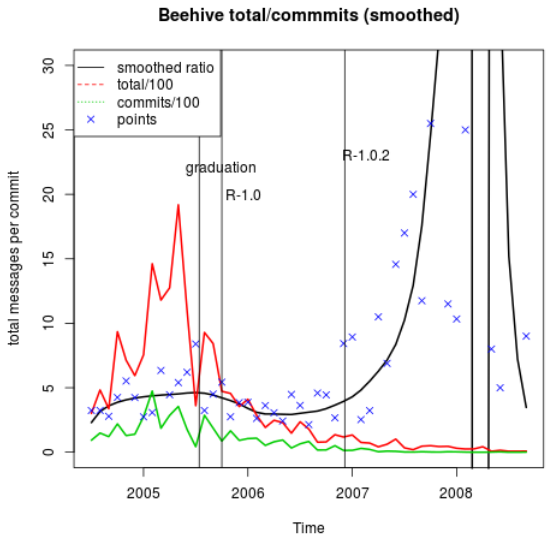

Fig. 12. Beehive measures. Main releases are annotated with vertical lines.

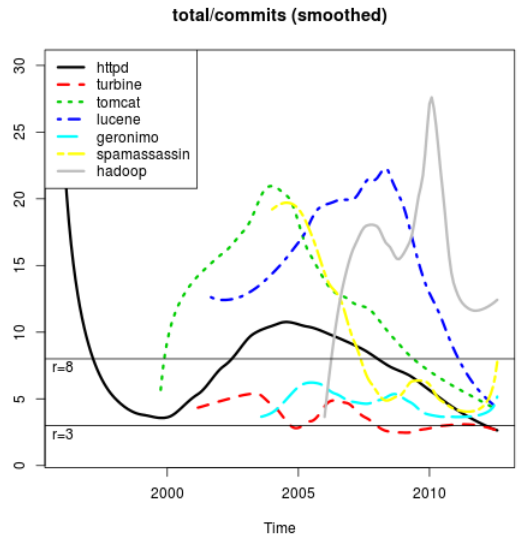

Fig. 13. Smoothed messages/commits ratio for some of the studied projects.

with a balanced groups of active developers. Noisy behavior of the information exchange/commits ratio is typical of projects with low activity or subject to future problems.

When a project stagnates, the metric allows to visually identify this, as it peaks to infinity. `Beehive` is a good example, as after one such period the activity ceased completely, and it was retired to the Attic two years later. 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.

Figure 13 provides an overview of the results shown for those projects that offer a continuous development. We can observe how the metric under study allows us to easily compare the evolution of the projects, even if those span two orders of magnitude in the size of the source code, the activity and the number of contributors differ in one order of magnitude, and they are of different age.

Among these projects, we can identify two groups: those projects that are user-oriented and those that are developer-oriented. For the former, we can see that a typical historical shape of the ratio starts with high values, as brainstorming sessions or discussion about design happen. In the maturity phase, however, the ratio goes down to values between 3 and 8. So, projects show high peaks, but tends to the above mentioned values as the project enters a development and maintenance phase.

Developer-oriented projects, such as `Turbine` and `Geronimo`, however show a different pattern. Having their community a minor size, there are no major spikes of user traffic and the peaks are narrower and do not leave the region where the ratio is between 3 and 8. `Geronimo` shows a maximum of 6.2 in July 2005 and a minimum of 3.7 in February 2011, and varied slowly. `Turbine` has stayed between a minimum of 2.5 in March 2009 and 5.4 in May 2003. `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.

In summary, the total traffic to commit ratio can be considered an intensive metric. However, at least for the projects under study, it seems better suited to user-oriented projects with large communities than for small, developer-only projects.

V. Using Other Ratios

A. Developer List Messages vs. Commits

We can limit our comparison to the developer mailing 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, developer mail traffic is typically created either by internal developers or people trying to join or just modifying the code for their use. Also, while the total traffic typically goes up when a project is used or adopted a lot, developer traffic increases before major releases or refactorings, and when there is brainstorming or discussions about design decisions.

As shown in Figure 14, when a project matures and development is limited to bug fixing and maintenance, the need for coordination is lower and a pattern with less messages on the developer mail list compared to commits can be found. This trend seems to be followed by most of the projects we have been analyzing. So, even when, like `Hadoop`, the total/commit ratio is very high, the developer mail/commit ratio is below one, meaning that developers either don't need to talk or the just "*talk with patches*", as Apache developers often say.

For the purpose of our study, the developer traffic to commits ratio seems to be an invariant metric as projects from very different (source code and community) sizes can be easily

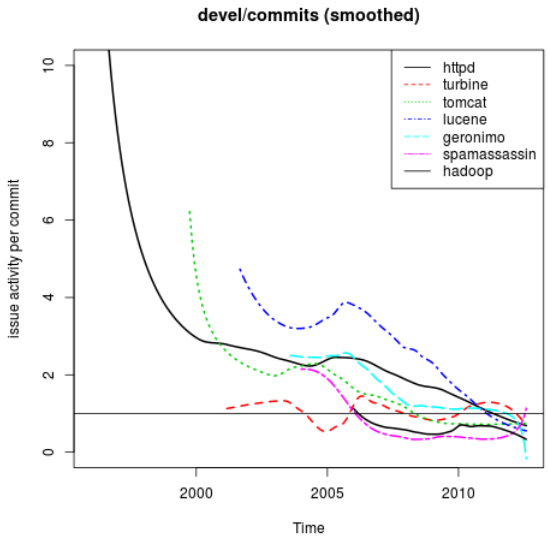

Fig. 14. Developers to commits ratio for all projects.

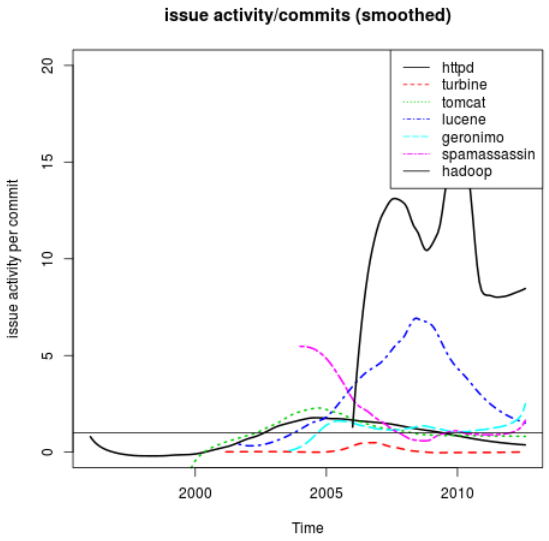

Fig. 15. Bug tracking issues to commits ratio for all projects.

compared. However, this metric is less *expressive* than the one that considers all the traffic in the project, in the sense that from the latter the phases of the projects can be more easily identified as a simple inspection of Figure 13 and Figure 14 shows.

B. Issues vs. Commits

Another information source is the one that is related to issues, i.e., bugs and enhancement requests submitted to the projects. The issue related activity does not show the same convergence with maturity than the previous cases when compared with number of commits (see Figure 15). It gets higher and higher as users are encouraged to report bugs and interact with the project via the issue tracker, and also when a project development is user-driven rather than developer driven.

However, we have found that the issue-related activity correlates with the total project activity in a similar way as with commits. Mature projects have a similar trend to get an issue-related activity of 10-15% of total activity. The Hadoop ecosystem, because of its use of the issue tracking system as it main communication point, shows a much lower total/issue ratio.

Again, the issues to commits ratio may be considered an intensive metric. However, its applicability is not clear to the authors at this time, and the information it provides can be obtained in a more clear way from the previous ratios.

VI. Related Research

The existence of *laws* that allow to explain how software evolves is a matter of study since the early 1970s, starting with seminal work by Lehman [3], [4]. There has been ample research literature on this issue from the mining software repositories point of view [5]. However, to the knowledge of the authors, there is no previous research concerned with finding intensive metrics for software projects, even from the point of view of software evolution. Although Lehman acknowledges the importance of metrics in the process of software evolution [6], the *laws* are written at a high level of abstraction, without referring to specific metrics. Hence, it is difficult to assign the metrics discussed in this paper to any of the laws, although probably the closest are *law* III of Self Regulation, which states that "global *E*-type system evolution is feedback regulated", and *law* VIII of Feedback System, which says that "evolution processes are multi-level, multi-loop, multi-agent feedback systems" [7]. Nevertheless, the fact that there are some factors that are inherent to software development is a central idea of the *laws*, and as such, this paper has much in common with Lehman's views.

The approach that has been presented in this paper has tight links with literature on open systems. In this regard, 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 [8]. Rong et al. [9] characterize "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. In [10], 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. Many authors have already thought that free software development should be viewed from such a perspective, being it an open ecology [11]. Some authors have even suggested that self-organization that occurs in the animal world (as the well-known stigmergy [12]), may

explain how software development occurs in free software projects [13].

Several studies on how the free software communities organize and structure themselves. The most prominent analyses are those by Mockus et al. on Apache [14], [15] and by Koch and Germán on GNOME [16], [17].

The role mailing lists play have been already the main focus of many empirical studies [18], even to the point where literature exists that highlights the concerns that should be considered for such type of analyses [19]. Of special interest to this research is a paper on when a contribution to a mailing list is meaningful [20]. There are some publications that provide further information specifically on the Apache mailing lists [21], [22].

VII. Threats to Validity

Being our approach an empirical research, some threats to its validity can be identified. In this section, we follow the usual classification of threats to construct, internal and external validity.

A. Threats to Construct Validity

- In our approach the unity of information is the message to the mailing list. Many issues regarding this can be found, as messages may have different values of content and redundancy.
- Although, as described in the methodology, we have tried to filter artificial peaks out, this is a manual task that probably has been done in an incomplete way.
- Messages might have gotten lost. As an example, the archives for `turbine` from 1998 to 2001 were not in Apache and got lost.
- Different information flows are not considered: press, documentation and other communication exchange in *downstream* projects such as the Linux distributions that package ASF software.
- For projects implementing standards, such as `Jackrabbit`, `Felix`, `Tomcat`, and 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 project. We are, thus, losing some of the information exchanges related with the project.

B. Threats to Internal Validity

- Extraneous effects: the use of new tools may produce changes in the ratio. So, the use of the Jira issue-tracking system has shown to produce more messages, and the introduction of the git versioning system results in more commits.

C. Threats to External Validity

- Non-representative sampling: We have worked in this paper at the granularity of the ASF subproject, starting from the fact that the ratio can also be observed at the global ASF project. The open question that arises is in how far our results and conclusions are valid when looking at smaller units. In projects dominated by a company with internal work-flows and where code is *dumped* periodically as an open source projects, lots of information is not publicly traceable and the methodology could not be applied. Such a case would be Android or MySQL.
- Non-representative research context: We have only studied a sample of projects from the ASF. Even if they all have their own goals, culture and sometimes rules, future research should see if the findings may be generalized to other free software projects, and to evaluate how the number of volunteers and professionals, the open communication exchange vs. internal/other means may affect the results. So, in projects making extensive use of videoconferencing or IRC channels, our method would not provide good results. A good example of this behavior is the Cyanogenmod project, where developers 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. In order to study such projects, different analysis techniques will have to be developed, for instance parsing IRC log activity or forum threads.

VIII. Reproduction Package

This study has been performed using shell, Python and R scripts. The data source used has been the official mailing list archives of the ASF projects under study. The scripts used to process and analyze the information, and other complementary information can be found in the companion web site with reproduceability elements and methodological details[5], built following the criteria presented in [23].

IX. Conclusions and Further Research

This paper presents the concept of invariant metrics in the domain of software projects, in particular with special attention to software evolution. we have shown three simple metrics that have the property of being scale invariant regarding to project size, community size, global activity and project age, thus being good candidates for an intensive metrics.

Of these, the ratio of total messages to the mailing lists to total number of commits in the versioning system has shown to be the most *expressive* metric, providing a good overview of the state of the projects under study. In addition, and because of its invariance, we are able to compare the projects regarding their state even if their sizes are very disperse. This metric has shown to serve to identify stagnant projects, and more importantly, projects in danger of stagnation. However, at least for the population of projects under study, its use is better suited towards user-oriented projects with a large community of users than to developer-oriented projects. Much of the

[5]http://gsyc.urjc.es/~grex/repro/2013-apache-intensive

information that can be easily visualized in the former is not available in the latter, at least with the current approach.

Our methodology is a simplification of our first idea, which was to compare the total communication flow against the development information. It would be tempting 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.

Further research should be put on verifying our results and lessons learned on other free software projects to see if they can be generalized and if the proposed metrics are really a) intensive and b) of any usefulness.

ACKNOWLEDGMENT

The authors would like to thank the Apache community, for their support and comments on some questions raised related to this paper. The work of G. Robles and J.M. González-Barahona has been funded in part by the European Commission under project ALERT (FP7-IST-25809) and by the Spanish Gov. under project SobreSale (TIN2011-28110).

REFERENCES

[1] K. Narayanan, *A Textbook of Chemical Engineering Thermodynamics*. PHI Learning Pvt. Ltd., 2004.

[2] W. G. Jacoby, "Loess:: a nonparametric, graphical tool for depicting relationships between variables," *Electoral Studies*, vol. 19, no. 4, pp. 577–613, 2000.

[3] M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," in *Proceedings of the IEEE*, vol. 68, no. 9, Sep. 1980, pp. 1060–1076.

[4] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution - the nineties view," in *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, nov 1997, p. 20.

[5] H. H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance*, vol. 19, no. 2, pp. 77–131, 2007.

[6] M. M. Lehman, D. E. Perry, and J. F. Ramil, "Implications of evolution metrics on software maintenance," in *Proceedings of International Conference on Software Maintenance*, 1998, pp. 208–217.

[7] M. M. Lehman, "Laws of software evolution revisited," in *Proceedings of the European Workshop on Software Process Technology*. London, UK: Springer-Verlag, 1996, pp. 108–124.

[8] G. Nicolis and I. Prigogine, *Self-organization in nonequilibrium systems*. Wiley, 1977.

[9] J. Rong, L. Hongzhi, Y. Jiankun, Z. Dehai, C. Lihua, and S. Yafei, "An approach to analysis and measurement of the stability of web community based on dissipative structure theory and information entropy," in *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, vol. 6. IEEE, 2009, pp. 326–329.

[10] J. Rong, L. Hongzhi, Y. Jiankun, F. Tao, Z. Chenggui, and L. Junlin, "A model based on information entropy to measure developer turnover risk on software project," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. IEEE, 2009, pp. 419–422.

[11] K. Healy and A. Schussman, "The ecology of open-source software development," University of Arizona, USA, Tech. Rep., Jan. 2003, http://opensource.mit.edu/papers/healyschussman.pdf.

[12] P.-P. Grassé, "La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interpretation du comportement des termites constructeurs." *Insectes Sociaux*, no. 6, pp. 41–81, 1959.

[13] G. Robles, J. J. Merelo, and J. M. González-Barahona, "Self-organized development in libre software: a model based on the stigmergy concept," in *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.

[14] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 24–35.

[15] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of Open Source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[16] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

[17] D. M. Germán, "The GNOME project: a case study of open source, global software development," *Journal of Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2004.

[18] E. Shihab, N. Bettenburg, B. Adams, and A. Hassan, "On the central role of mailing lists in open source projects: An exploratory study," *New frontiers in artificial intelligence*, pp. 91–103, 2010.

[19] N. Bettenburg, E. Shihab, and A. E. Hassan, "An empirical study on the risks of using off-the-shelf techniques for processing mailing list data," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 539–542.

[20] W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan, "Should i contribute to this discussion?" in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 181–190.

[21] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 137–143.

[22] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 23.

[23] J. M. González-Barahona and G. Robles, "On the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 75–89, 2012.