Download Patterns and Releases in Open Source Software Projects: a Perfect Symbiosis?

Bruno Rossi, Barbara Russo, Giancarlo Succi

Download Patterns and Releases in Open Source Software Projects: a Perfect Symbiosis?

Bruno Rossi, Barbara Russo, and Giancarlo Succi

CASE – Center for Applied Software Engineering
Free University of Bolzano-Bozen
Piazza Domenicani 3, 39100 Bolzano, Italy
{brrossi, brusso, gsucci}@unibz.it
http://www.case.unibz.it

Abstract. Software usage by end-users is one of the factors used to evaluate the success of software projects. In the context of open source software, there is no single and non-controversial measure of usage, though. Still, one of the most used and readily available measure is data about projects downloads. Nevertheless, download counts and averages do not convey as much information as the patterns in the original downloads time series. In this research, we propose a method to increase the expressiveness of mere download rates by considering download patterns against software releases. We apply experimentally our method to the most downloaded projects of SourceForge's history crawled through the FLOSSMole repository. Findings show that projects with similar usage can have indeed different levels of sensitivity to releases, revealing different behaviors of users. Future research will develop further the pattern recognition approach to automatically categorize open source projects according to their download patterns.

Keywords: Open source software projects, software releases, repository mining.

1 Introduction

Determining the success of software projects is very often non-trivial. There are many aspects to consider, and even the definition of success can depend from multiple point of views. Nevertheless, discerning the success of software projects is useful as researchers can evaluate approaches, methods, and processes that performed well - given a certain context. Furthermore, the availability of large data about open source software projects made such research more appealing. At the same time such task can be more difficult, as we miss some important in-context information that can be gathered only as insider of a development team. Reconstructing such information can be problematic when mining online repositories without strict contact with the original development team.

The definition of success is also not unique. One of the views of software projects' success is directly dependent on the users. Specifically, as reported in [2], in Information System (IS) research the success of a software system has been studied as

directly dependent on system and information quality [4]. According to this view, system quality impacts directly on software usage, and thus on users satisfaction. Deriving the success of a project is thus a question of considering a) the impact of the system quality on the users' usage level and b) the acceptance rate of the user. Once this has been determined, then it is relatively straightforward to associate successful projects to their development practices, development process characteristics, or even product features. Leaving aside software quality, the determination of criteria to measure the usage of software and relative users' satisfaction are relevant research problems.

If we focus on usage, and on commercial software, there is at least one single indicator that can be reliably used as an indication of usage: the number of copies sold on the market. It is difficult that applications bought do not translate in real usage of the application. For open source software, the situation is more fuzzy, as there are no unique indicators of usage. Different proposals have been made, like using the number of downloads [3], adopting software agents to monitor the software usage [2], tracking the inclusion in software distributions [2], using downloads time-series to detect the evolving users' community [10], or even use web search engines results to derive the popularity of projects [11].

Conversely, determining users' satisfaction is probably easier for open source software than for proprietary software. The large number of data available allows mining of repositories to audit mailing lists, forums, bug tracking reports, and so on. Also in this case, there is no single universal indicator of users' satisfaction and different proposals have been made: considering user ratings, opinions on mailing lists, or surveys [2] – among others.

In this work, we focus on software usage and specifically on download rates. Software usage needs to take into account different measures, but we believe that download rates have not been exploited to their full potential. Especially for open source software, downloads have been identified useful as a proxy of software usage. Recently, the interest is more on the patterns that have been detected rather than the mere download indexes (as for example [6] and [10] justify).

Our hypothesis is that it is not true that in all open source software projects download rates are in relation with software releases. More precisely, we believe that there are projects in which download rates are in relation with the application type (e.g. file sharing applications, where users want to have the latest application available, security fixes included) and others where users do not really care about the release date (e.g. applications installed and kept for longer time, like graphical utilities). From these hypotheses, it derives automatically that different patterns and - even more - download numbers can represent very different situations that are difficult to generalize. So, before reconstructing patterns to see whether projects were successful or not, we will need to know how much projects are sensitive to software releases. We have two research questions for this paper.

- RQ1. Are download patterns connected to releases in open source software projects?
- RQ2. If such relation exists, is the relation consistent in the same category of projects?

The paper is structured as follows. Section 2 presents background on deriving open source software usage and specifically on studies on the evaluation of usage by means

of download rates. Section 3 discusses further the research questions by means of a problem statement. Section 4 presents a method for analyzing downloads time series. Section 5 proposes an experimental evaluation of the method by means of the highest ranked projects in *SourceForge's* history. The section includes experimental design, data collection, data filtering, evaluation of the method, findings and limitations. Section 6 is about conclusions and future works in prospective research.

2 Background

Different techniques have been proposed to derive software usage. If we specifically focus on download rates, this indicator has been found to have several advantages, like the fact that it is relatively easy to gather this kind of information from online repositories. Nevertheless, various disadvantages are also reported, like the issue that a single download may not really translate in software usage [2]. What is very often suggested is to use this measure with care [3].

There has been a move in recent years from considering mere download rates (we can report as an example [6]) towards analyzing time-series and emerging patterns. In this sense, there is a consensus now among all researchers that download averages, or totals, do not convey enough information to be used as independent or dependent variable in success prediction models. Time series of downloads convey a larger set of information. The problem is that in the open source scenario, with large datasets available, some kind of data compression or summarization is needed so that knowledge about single projects can be synthetically represented, summarized, and then visualized. We can report specific studies that are related to the current research (Table 1).

Table 1. Related Studies.

Paper	Study	Results
[6]	Identification of classes of successful and unsuccessful projects according to download patterns	Six patterns of download rates identified. Justification of emergence of such patterns
[7]	Identification of successful open source projects by means of downloads numbers	Categorization of 122,065 projects in super, successful, and struggling. No evidence of Zipf's Law for the number of downloads for projects on *SourceForge*
[10]	Proposing a method to measure size of open source projects and use base based on downloads time series	Different types of users found according to adaptation of downloads to releases

All these studies have in common the analysis of download rates. Research questions are different, as well as the experimental setting. In [6] and [7] the focus is

on deriving successful projects, in [10] the focus is on deriving the use base from download patterns.

3 Problem Statement

To present the problem statement, we give a practical example by means of two well known software. Namely, we consider the most downloaded project of *SourceForge's* history - *eMule*[1] - and the *TCL*[2] application. This selection is not random, as those applications were selected in the categorization made in [6], so the interested reader can find the motivation more compelling. As can be seen, the two time series of downloads for the two projects are quite different (Fig. 1 and 2).

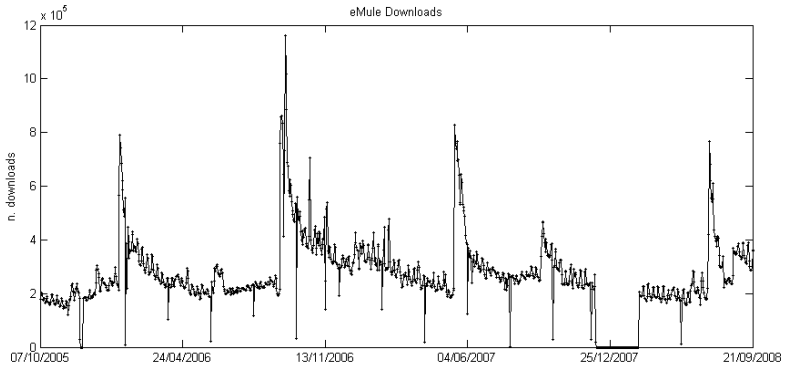


Fig. 1. eMule project downloads.

What we see from the figures is that there are some short-time cyclic patterns in the download rates (maybe weekly) in both cases, but in the eMule case there are evident longer term cyclic patterns. What we ask ourselves is whether the latter type of patterns are related to software releases.

A simple approach would be to simply plot release dates on the same time series and evaluate manually the situation case by case. Instead, what we want to derive is an approach that allows to detect automatically - and without visual inspection - whether a time series is dependent on release dates. In the specific, if there are strong increases in download rates in coincidence with a software releases. Such method must also remove less important cyclic patterns in download rates. Considering the example in Fig. 2, we do not want to consider relevant low cyclic patterns that repeat weekly.

[1] http://www.emule-project.net
[2] http://www.tcl.tk

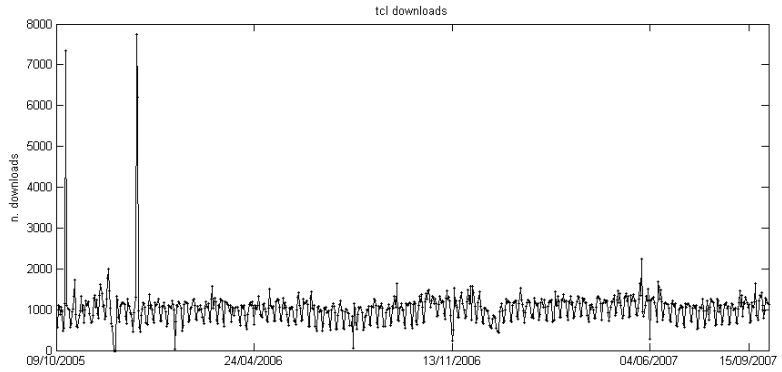


Fig. 2. TCL project downloads.

Thus, we present a proposed automated method that can be used applied to time series to evaluate the sensitivity to software releases. We apply it to downloads time series, but it can be potentially applied also to other types of time series (like the number of commits in time, for example). In the following, we use mostly the *TCL* project to explain the method.

4 Method

To eliminate non-relevant cyclic patterns, we use a technique called *Piecewise Aggregate Approximation (PAA)* that approximates a time series by means of segments. This approach has been used, for example, when handling large amounts of data to reduce the complexity of similarity search space [1]. In our case, segmentation helps not only in reducing the data points to be considered for analysis, but also in reducing short term cyclic patterns in the time series.

In the specific, in this research, we used a specialized form of *PAA* that uses a wavelet transform of the time series to decompose the segments of the time series [1, 8]. The wavelet used is the simplest form of wavelet: the *Haar* wavelet [8]. In Figure 3 we propose an example of *PAA* applied to a synthetic data set. The original time series is approximated by means of a wavelet that maintains similar patterns as the original. As can be seen directly from figures, the data loss is inversely proportional to the number of segments used for *PAA*. Conversely, more segments mean also keeping more short time periodic patterns.

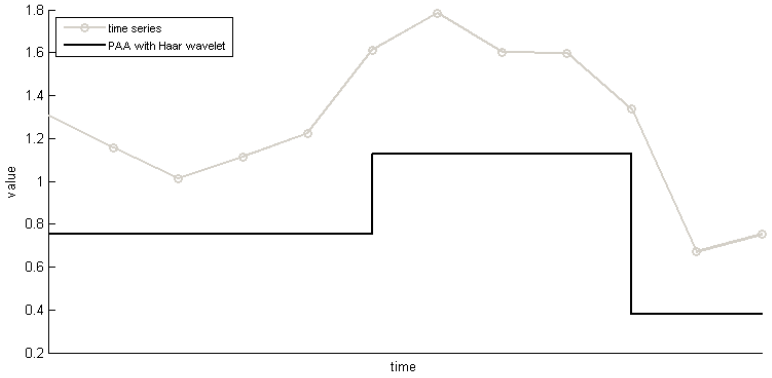


Fig. 3. Example of PAA by using Haar wavelets.

The whole theory of wavelets is far beyond scope for this paper, but an interesting overview can be found in [8]. Alternative and more sophisticate techniques for *PAA* can be found in [1]. To show the results of the technique, we applied it to the *eMule* and *TCL* projects (Fig. 4 and 5). After the application of the method, we have thus eliminated unwanted cyclic patterns from the time series, and got a simpler representation. We will then use this representation to detect the intersection of areas with releases automatically.

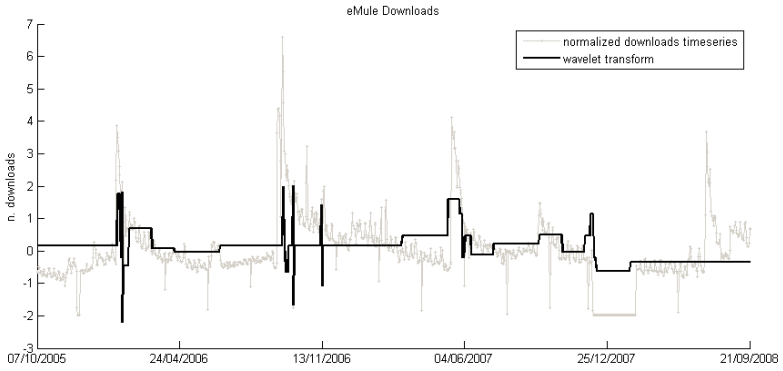


Fig. 4. eMule project wavelet transform.

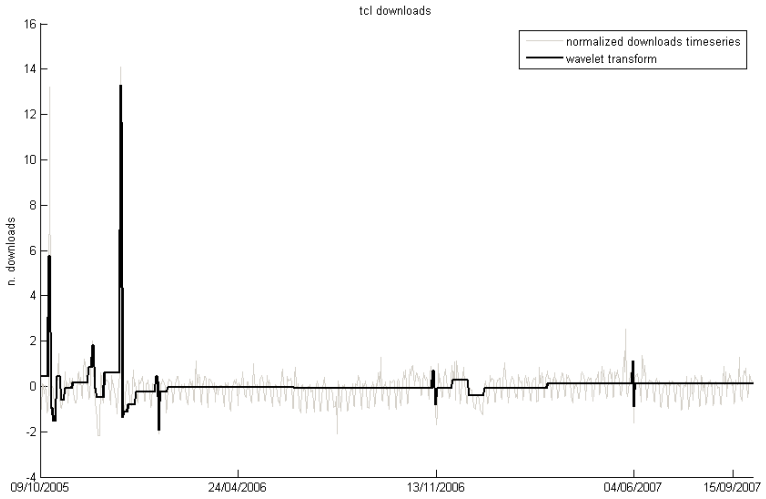


Fig. 5. TCL project wavelet transform.

In detail, the whole approach is the following:

a. Represent projects as time series of downloads;
b. Filter out projects that have too many missing values;
c. Perform linear interpolation of missing values. This is needed as the original data set might have missing data points from the data collection process. Linear interpolation helps in reducing the impact of such data points;
d. Perform *PAA* on each time series;
e. Discriminate in the wavelets the areas of different levels of activities as determined by *PAA*;
f. Plot release information into the time series;
g. Evaluate releases in the different intervals identified, summarizing the result with two metrics;

So, once the transform has been applied, we need a way to summarize the patterns of the original time series. For this, we divide the wavelet into different intervals according to the level of burstiness, identifying bursty intervals and more constant intervals. The reason is that we want to codify our time series in such a way that it is more easy to automatically derive intersections with release dates. This is similar to what has been proposed in [9] to analyze development iterations. After such identification, we introduce the dates of releases and we evaluate the intersection of release dates with different areas.

To divide areas of wavelets, we consider periods where activity is frenetic (*A*) and others where periods of activity are more constant (*B*). For this, we will introduce the following notation for the remaining of the paper: we define tw_i as the i^{th} point in the wavelet time series, $I_{k,\varepsilon}$ as an interval in the time series, where tw_k is the starting point

and a positive integer ε is the length of the interval $I_{k,\varepsilon}$. To detect automatically areas, we use the following discriminating rule:

$given\ \varepsilon > 0$
$I_{k,\varepsilon} = \left[tw_{k\varepsilon+k}, tw_{(k+1)\varepsilon+k}\right]$

$I_{k,\varepsilon} \rightarrow A\ iff\ \exists i \in tw_i \mid tw_i \in I_{k,\varepsilon} \wedge (tw_i \geq (\mu(tw) + \sigma(tw)) \vee (tw_i \leq (\mu(tw) - \sigma(tw))$
$I_{k,\varepsilon} \rightarrow B\ otherwise$

We use this rule to classify between A, and B periods. For convenience, we also define J_A as each connected set of intervals $I_{k,\varepsilon}$ of type A. The same represents J_B for B periods. Fig. 6 shows the results of area mapping for the *TCL* project.

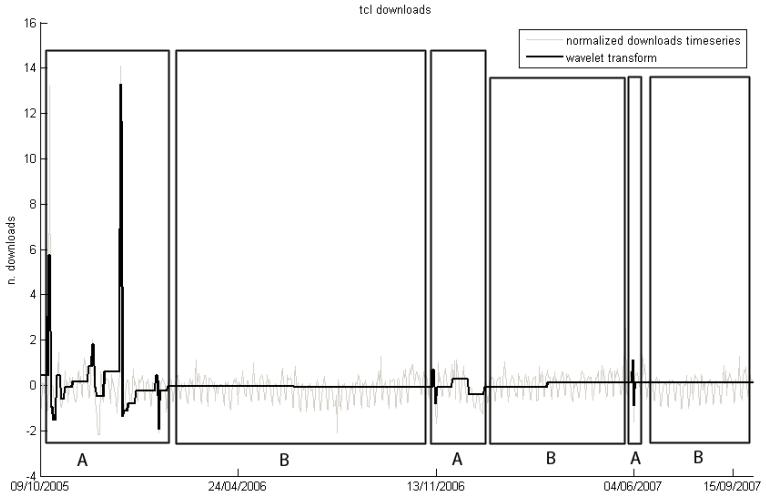


Fig. 6. TCL project with areas identified (spaces between areas are just to ease the interpretation of figure).

We next evaluate the relation between different area types and releases. We map thus how many releases happen for each project in specific areas. More releases in areas of type A mean that the project is more subject to a relation between releases and download rates. The simple approach that we used in current research is to use the intersection of releases and areas. If we apply this to *TCL* project, we can see that the project is scarcely sensitive to release dates (Fig. 7).

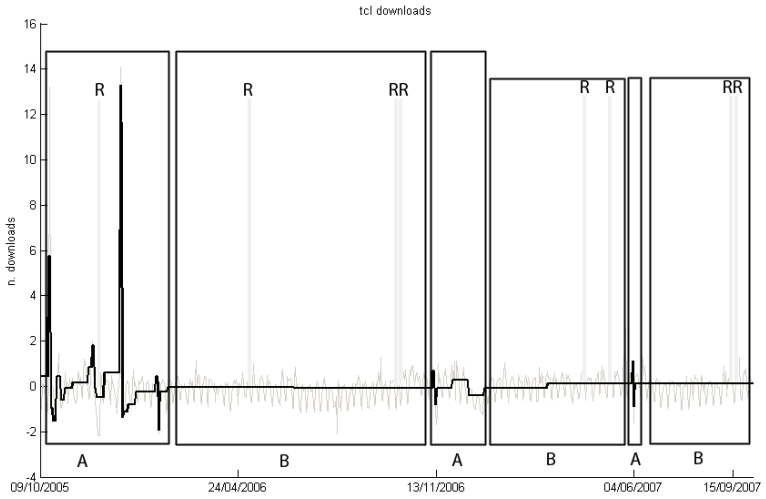


Fig. 7. TCL project with areas and release dates.

To automate the process, and to allow for automatic use of the approach without examining the figures, we defined two metrics. One metric is about the sensitivity to releases, the other about the coverage of different areas. This information is needed as we can have the same level of sensitivity to releases but different level of burstiness of the time series. Without visual inspection, we will miss this relevant information. First we define the set of all releases of a project as:

$$R = \{r_1, r_2, \ldots, r_n\} .$$

Then we define the first metric - that we call *s* - as the sensitivity to releases:

$$s = \frac{\sum[R \in J_A]}{\sum R} . \quad (1)$$

This metric is a weighted ratio of how many releases happen in one period of larger activity in the downloads time series. An index of 1.0 means that all the releases of a software project happen when the downloads time series are active the most. Conversely, a metric of 0.0 shows no reaction to releases: users' download of software is completely separated from release dates.

We then define the second metric as the amount of burstiness of different time series. We refer to this metric as *b*.

$$b = \frac{\sum |J_A|}{\sum (|J_A| \cup |J_B|)} . \quad (2)$$

An hypothetic index towards *1.0* means a time series of downloads completely bursty. An index towards *0.0*, inversely represents a time series of downloads where rates are almost constant. Thus in the example of *TCL*, we have a download rate that is not very sensitive to releases (*s=0.10*) and with a trend of downloads not so bursty (*b=0.31*). This result is consistent with what reported in [6] where the application is reported as an application with regular downloads not related to releases. As can be seen, these two metrics give us more information than the mere average download rates.

5 Experimental Results

We provide an experimental evaluation of the method by applying it to several projects out of the *FLOSSMole* repository [5] and with data about releases gathered through *SourceForge*. The strategy of selection of the sample of projects was to select the most downloaded projects from the whole *SourceForge's* history (Table 2). This choice had the aim of limiting missing data points in the time series on one side, and providing a first evaluation that can then be replicated on less downloaded projects. In this way, this first evaluation has the aim to gather findings from a well-known set of applications where downloads totals are quite consistent.

For each project, we gathered the time series of downloads, limiting the analysis to *1.000* data points. For the selection of the parameters, based on the sensitivity analysis reported in the next section, we selected the length of each interval $I_{k,\varepsilon}$ with ε=30. After collecting the data, we applied the whole approach to the dataset of each project.

Table 2. Most downloaded projects in Sourceforge's history.

Rank	Project Name	Type	Total Downloads	% Zeros TS 1000
1	eMule	P2P Client	510,493,881	0,00%
2	Azureus / Vuze	P2P Client	455,284,828	0,30%
3	Ares Galaxy	P2P Client	209,066,979	0,40%
4	7-Zip	Compression Software	76,806,020	12,30%
5	FileZilla	Client FTP	71,295,059	34,40%
6	GTK+ and GIMP installers for Windows	Graphical tool	64,212,148	24,30%
7	Audacity	Audio Editor	64,051,083	4,40%
8	DC++	P2P Client	56,488,262	4,30%
9	PortableApps.com: Portable Software/USB	Utility	55,713,765	18,90%
10	BitTorrent	File Sharing	52,031,664	81,08%
11	Shareaza	P2P Client	49,799,296	0,20%
12	VirtualDub	Video Editing	47,835,670	10,20%
13	CDex	Digital Audio Ripper	39,738,454	6,30%

14	Pidgin	Instant Messaging	32,309,818	24,10%
15	aMSN	Instant Messaging	31,175,716	6,30%
16	WinSCP	File Transfer Client	29,681,313	6,10%

5.1 Filtering

A first problem we had with the data set was how to handle zero counts in the time series. For this set of largely downloaded projects, our interpretation is that a zero in the download totals for one day means a missing point in the data collection process. The evolution of the download counts for such projects also justifies this view, with zeros interleaved to medium-high level download counts. Since our method supposes the application of interpolation, we wanted in any case to avoid its excessive application. For this reason, we filtered out from the sample the projects that had more than 10% zeros in the time series. Additionally, two projects were excluded from the sample. The *DC++* and the *CDex* projects were removed as we didn't have enough information about release dates.

5.2 Interpolation

For projects that were included in the sample, we interpolated linearly the missing points from previous and subsequent values in the time series. In this way, even with an approximation, we limited the impact of missing values from the data collection phase that could lead to an erroneous generation of different areas in the time series. In this experiment, we used a simple linear interpolation.

5.3 Sensitivity Analysis

There is one subjective choice when applying *PAA*: the selection of the number of segments to use. In our research, we considered monthly segments (segments of length *30*) and we believe this is as an appropriate number of segments according to our knowledge of the dataset, as we wanted to avoid weekly cyclic patterns. To support this decision, we performed a sensitivity analysis (Fig. 8), by calculating the Euclidean distance between the original time series and the wavelet. The analysis shows how *PAA* fits the original time-series according to different number of segments. With our selection of the parameters, we do not compress excessively the original representation of the dataset.

If we consider a higher number of segments, the fitting will improve going from monthly, to weekly segments, for example. By doing this, we will also introduce more cyclic patterns into the time series. So there is a trade-off in this sense. Our heuristic of selection for the estimation of the segments preferred to use monthly segments to reduce effects of weekly patterns in the time series.

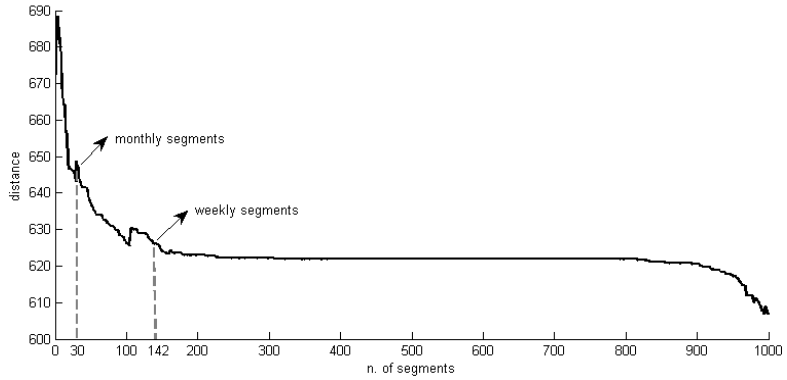


Fig. 8. TCL Project, distance between the wavelet and the time series according to number of segments (the lower value the better the fitting).

5.4 Results

After filtering and interpolation, we considered out of the initial 16 projects, just 7 projects (Table 3). The following categories were included: a) P2P Clients, b) Audio applications, c) Instant messaging, d) File Transfer Clients.

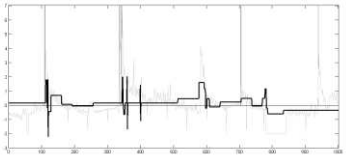
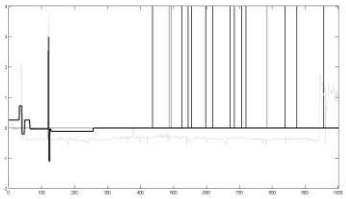
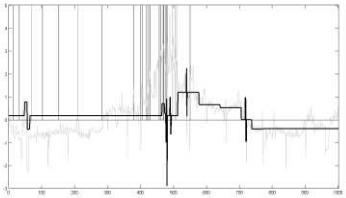
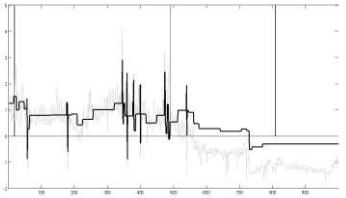
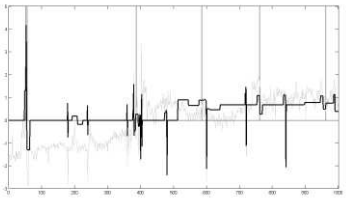
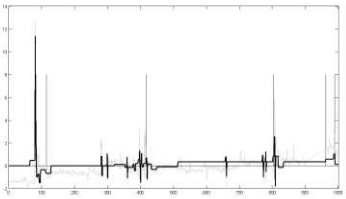
Table 3. Selected Projects.

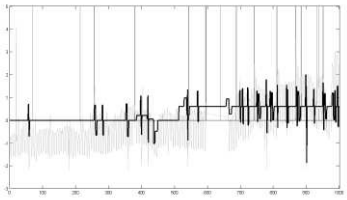
Project Name	Type	Total Downloads
eMule	P2P Client	510,493,881
Azureus / Vuze	P2P Client	455,284,828
Ares Galaxy	P2P Client	209,066,979
Shareaza	P2P Client	49,799,296
Audacity	Audio(Audio Editor)	64,051,083
aMSN	Instant Messaging	31,175,716
WinSCP	File Transfer Client	29,681,313

Then we applied to all projects the *PAA* technique, the derivation of areas in the time series, and the calculation of the metrics for sensitivity to releases and the level of burstiness. We report in the following the results.

For each project, we present the project name, the figure of the wavelet against releases, the parameters for sensitivity to releases, and burstiness of the wavelet as calculated by our approach (Table 4). The reader can see that in some cases, a high level of sensitivity to releases (*s* parameter) can even be enforced by the fact that there are shorter areas of burstiness (*b* parameter) .

Table 4. Analysis of the Projects.

Project Name	Original time series, PAA and releases	*s*	*b*
eMule		0,80	0,39
Azureus / Vuze		0,06	0,15
Ares Galaxy		0,48	0,32
Shareaza		0,66	0,48
Audacity		0,83	0,33
aMSN		0,75	0,38

WinSCP		0,69	0,57

If we look at the results, we can observe the following interesting phenomena. For almost all projects, there is a relation among releases and download rates. The only project where this doesn't happen is the *Azureus/Vuze* project. This goes against our assumption that a user of a P2P application always wants to get the latest release as soon as possible, for example to get security fixes that are particularly important for this category of application or improvements like greater download speeds. If this doesn't happen for this particular application, it could mean that there specific characteristics of the application, or in the modality of distribution of the application that can be different. It can also be an indication that users – differently from the other cases – received the updates mostly from updates inside their Linux operating system distribution and not via software downloads. So this can also be in fact an indication that download rates for that application have to be taken with care.

5.5 Findings

Popular open source software projects follow different patterns of downloads according to the release of a software version. Mostly projects downloads follow the dates of releases with typical increases, but this is not always the case. It is thus interesting to examine the reasons of projects that do not strictly follow this rule. We summarize the findings deriving from the research questions in Table 5.

Table 5. Summary of the Findings.

Research Question	Finding(s)
RQ1. Are download patterns connected to releases in open source software projects?	We found that - in the majority of the projects analyzed - releases lead to an increase in download rates. In some cases, such behavior is less evident or even absent (e.g. Azureus). The explanation for this can be in the characteristics of users or the project features, but can also be an indication that download totals are not completely reliable for that specific application.

RQ2. If such relation exists, is the relation consistent in the same category of projects?	We found that the behavior is not consistent across all categories. Even in the limited set of categories we used, users respond in different ways to software releases even in the same category of applications. For example, in our sample, it is not true that users of P2P applications are more interested than other users in getting the latest release of the software.

We suspect that for projects where download patterns are not strictly in relation to releases there are two distinct explanations:

a. users really do not care about the latest release of the application. This can also happen because the update of the application requires much effort compared to the advantages of the update, so the user may decide to postpone the update to a later time;
b. users are interested in updates and are actually updating the software as a new version appears. In this case, downloads time series do not capture this behavior, maybe because users are getting the updates by means of alternative sources (other websites than *SourceForge* or through the mechanism of updates in their own Linux distribution);

We argue thus that if we are in the a) case, downloads time-series can still be used as a somewhat reliable indicator of project's success in combination with other measures of usage and users' satisfaction. Conversely, if we are in the b) case, the evaluation of download rates must be complemented with additional information deriving – as an example – from projects' websites traffic, and/or search engines queries, like has been proposed in [11].

5.6 Limitations

The main limitation of the approach is about the definition of the parameters of *PAA* segmentation and areas definition. Although we provided the heuristic of selection and sensitivity of the model to the parameters when explaining the approach, it is clear that different parameters can lead to slightly different results. Specifically, the choice of the length of the interval $I_{k,\varepsilon}$ can give as result areas of different size to be used then in the metrics for calculation. Sensitivity analysis has been performed to reduce and limit this effect.

6 Conclusions and Future Works

We proposed a method to augment the expressiveness of downloads time series of open source software projects. We added information about the relation of projects'

downloads to releases and defined two metrics. The metrics defined can give information about the responsiveness of the users to releases. This is a first step in research of automatic detection of patterns in downloads time series. Information from such patterns can then be used in models to detect projects' success.

We applied experimentally the method to a subset of projects in the *SourceForge* repository. We showed that codifying the downloads time series as two metrics conveys more information than using global metrics like average download rates or total download counts. As we have seen experimentally, even if projects have similar total download rates and counts, they can follow completely different download patterns. As such considering just those numbers can lead to wrong or biased conclusions. Furthermore, project downloads can be more or less related to software releases showing different behaviors from the point of view of users that can depend – and this will need to be validated in future research - on projects characteristics, application type or even modality of distribution.

Future research goes into two directions. One direction is to extend the approach to a larger data set, specifically focusing on projects' categories. The second direction is to investigate successful projects with an extension of the methodology developed in this paper.

Acknowledgments. We thank the creators and maintainers of the *FLOSSMole* repository for granting access and for their constant effort in providing a useful source of information about open source projects.

References

1. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans. Database Syst. 27, 2, 188-228 (2002)
2. Crowston, K., Annabi, H., Howison, J.: Defining Open Source Software Project Success, in proceedings of the 24th International Conference on Information Systems (ICIS), pp. 327-340 (2003)
3. Crowston, K., Annabi, H., Howison, J., Masango, C.: Towards a portfolio of FLOSS project success measures, the 4th workshop on Open Source Software engineering, International Conference on Software Engineering (2004)
4. Delone, W.H., McLean, E.R.: The DeLone and McLean Model of Information Systems Success: A Ten-Year Update, J. Management of Information Systems, vol. 19, pp. 9-30 (2003)
5. Howison, J., Conklin, M., Crowston, K.: FLOSSmole: A collaborative repository for FLOSS research data and analyses. International Journal of Information Technology and Web Engineering, 1(3), 17–26 (2006)
6. Israeli, A., Feitelson, D. G.: Success of Open Source Projects: Patterns of Downloads and Releases with Time. In IEEE International Conference Software Science, Technology, & Engineering, pp. 87-94, (2007)
7. Feitelson, D. G., Heller, G. Z., Schach, S. R.: An Empirically-Based Criterion for Determining the Success of an Open-Source Project. Proceedings of Australian Software Engineering Conference, pp. 363-368 (2006)
8. Li, T., Li, Q., Zhu, S., Ogihara, M.: A Survey on Wavelet Applications in Data Mining. SIGKDD Explor. Newsl. 4, 2, 49-68 (2002)

9. Rossi, B., Russo, B., Succi, G.: Analysis of Open Source Software Development Iterations by means of Burst Detection Techniques, Proceedings of the 5th International Conference on Open Source Systems, pp.83-93, Springer, Boston (2009)
10. Wiggins, A., Howison J., Crowston, K.: Measuring Potential User Interest and Active User Base in FLOSS Projects, in proceedings of the 5th International Conference on Open Source Systems, pp.94-104 (2009)
11. Weiss, D.: Measuring Success of Open Source Projects using Web Search Engines, in Scotto M., Giancarlo S. (Eds.): Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy, pp.93-99 (2005)