Carnegie Mellon University
Research Showcase @ CMU

Institute for Software Research | School of Computer Science

2009

# Heartbeat: Measuring active user base and potential user interest in FLOSS projects

Andrea Wiggins

James Howison
*Carnegie Mellon University*

Kevin Crowston

Follow this and additional works at: http://repository.cmu.edu/isr

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Institute for Software Research by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

# Heartbeat: Measuring Active User Base and Potential User Interest in FLOSS Projects

Andrea Wiggins, James Howison, and Kevin Crowston

**Abstract** This paper presents a novel method and algorithm to measure the size of an open source project's user base and the level of potential user interest that it generates. Previously unavailable download data at a daily resolution confirms hypothesized patterns related to release cycles. In short, regular users rapidly download the software after a new release giving a way to measure the active user base. In contrast, potential new users download the application independently of the release cycle, and the daily download figures tend to plateau at this rate when a release has not been made for some time. An algorithm for estimating these measures from download time series is demonstrated and the measures are examined over time in two open source projects.

## 1 Introduction

Measuring FLOSS project success has been a subject of intense research interest; it is the most common dependent variable in FLOSS research [1]. One of the most desirable measures of external success is a measurement of software use, representing the volume of active users of the software. Unfortunately, usage is also one of the hardest measures to acquire, since software use occurs in a local computing context and rarely leaves evidence in public archives. This paper draws on user and developer experiences to design an algorithm to process download counts to estimate the change over time in both the general level of interest, drawn from experimental first-time downloads of the software, and the size of the active user base.

Andrea Wiggins
Syracuse University, Hinds Hall, Syracuse, NY 13244 USA e-mail: awiggins@syr.edu

James Howison
Syracuse University, Hinds Hall, Syracuse, NY 13244 USA e-mail: jhowison@syr.edu

Kevin Crowston
Syracuse University, Hinds Hall, Syracuse, NY 13244 USA e-mail: crowston@syr.edu

## 2 Measuring Software Use

Software use is a component of all software success frameworks [3, 6], including those which focus specifically on the FLOSS environment, as discussed in detail in recent review articles [5, 2]. Unfortunately, usage is very difficult to measure in the FLOSS context. There are two basic techniques that are commonly suggested: surveys of users and analysis of public archives.

Surveys of users would perhaps provide the best data because it would be possible to ask which applications were tried, which were selected for regular use (and why) as well as measuring frequency of use. Surveying users of FLOSS projects poses significant challenges, however, because of the lack of an available sampling frame and the large number of FLOSS projects for which comparative information would be desired. As well, responses to such a survey would be affected by limitations of user recall. Automatic surveys are a possible source for these data, and there are limited efforts to conduct automatic measurement by installing agents on user's computers which report usage statistics. For example, the "Debian Popularity Contest" project[1] reports usage in this fashion for packages on the Debian (and Ubuntu) Linux distributions: users opt-in to install a special reporting package which provides the project with aggregated and anonymized statistics regarding packages installed and packages actually used. Assuming there is little systematic bias amongst those that choose to install the reporting package, the data is excellent, but it remains limited to Linux users and measures usage at the Debian package level, rather than the individual project level. Individual FLOSS projects sometimes gauge usage through software agents installed in the application or through automated crash reporting software, although the practice is somewhat controversial and the data rarely available to researchers.

The second technique is to analyze online evidence of application use. This is relatively easy for applications that regularly connect to the web. For example, the Netcraft league tables measure the "market share" of various web servers by using spiders that crawl websites and headers set by the products. Other groups provide data on web browser usage, based on the evidence from the user agent strings logged at different websites. These techniques are also feasible for applications delivered as web services. Unfortunately, most FLOSS applications are still used in a local context where application usage does not leave evidence in public archives, making these approach inapplicable.

Because of the lack of direct measures of use, many researchers turn to other available data that serves as a proxy for usage. Download counts are the most commonly used success measure, and are often argued to be a viable proxy for software use and an analogy to market share. The choice of downloads as a proxy for usage rests on the fact that obtaining software is antecedent to using it, and in the FLOSS world the most common method for obtaining software is downloading it, often from a project forge site, which leaves evidence in public archives that is readily

[1] `http://popcon.debian.org/`

available to researchers. However, there are numerous problems with this interpretation.

First, downloads tell us relatively little about the size or growth of an active user base because the measure can not address how many downloads convert into actual use of the software. Download counts may be an inflated measure of usage due to a process of experimentation, whereby the application is downloaded but rapidly discarded as unusable. Second, it is problematic to assume that a single data source such as a count of downloads tells the full story, since software is distributed through multiple channels (physical media, Linux distributions, etc). This limitation means that downloads will misrepresent more mature projects that are most likely to be part of a default installation.

Finally, a further confound, particularly for the common use of aggregate lifetime download figures, is that FLOSS projects release updated versions of their software relatively frequently and downloads of these new releases by current users will inflate the total download counts. More problematic though, projects release at different rates. A project making more frequent releases may have more downloads, even though the actual number of regular users may not be significantly different from a project that makes fewer releases and therefore has fewer downloads. Again, mature projects suffer in these comparisons because they typically make fewer releases, even though they might have a very satisfied user base.

The remainder of this paper uses contextualized knowledge of patterns of use and downloads to design an algorithm to process download measure over time to remove some of these confounds. The result is a more nuanced and useful understanding of download figures and measures of both regular users, experimentation rates and the conversion between them.

## 3 Hypothesis development

In this section, we draw on personal experience to develop a set of hypothesis about download behaviour and its relation to software use. The authors are all regular users of many FLOSS applications but as a basis for theorizing, we consider two in particular, BibDesk and Skim-app, as one of the authors has participated as a developer in these two projects for several years. BibDesk is a reference manager for BibTeX, while Skim-app is a PDF reader and annotator optimized for academic papers. Both applications run exclusively on the Mac OS X platform and are almost exclusively distributed through regular downloads from their Sourceforge sites[2].

In considering these projects, we made two important observations regarding the relationship between downloads and software use. First, the software was initially obtained by the authors while exploring alternative reference managers. The initial

[2] We say 'almost' because both applications are sometimes included in packaged distributions for scientists, such as MacTeX. These distributions almost always lag behind the current versions, however, and observation on the mailing lists shows that they are not a large source of users for the applications.

download was driven by the private work cycles of the potential users, but once adopted, the application is used regularly, usually daily.

Second, these projects are both active and periodically release new versions that both fix bugs and extend features. Releases vary in their frequency, sometimes coming 'often and early', sometimes at a slower pace, depending on the development activity of the project team, although rarely with longer than a few months between releases. Both projects also release more than one package each. In addition to the BibDesk package, the BibDesk project provides an Input Manager for LaTeX programs, and Skim-app makes available a library for extending the use of user annotations. In both cases, however, the primary package is several orders of magnitude more popular than the secondary packages.

When projects make a new release, users are informed through two methods. The first is an announcement through the project's mailing lists, including the "ANN" (for announcement) lists which are specifically intended for this purpose. The second, a more recent addition, is through a "nag" screen in the software itself, which checks for new versions at specified intervals. For these applications, users may be motivated to download new versions relatively quickly. This is in part because the software is improving in desirable ways, but it is also driven by the fact that user support responses usually begin by insisting that the user first update to the most recent version. This practice is common in FLOSS projects, since the latest version has often fixed the issue in question. In our experience we have usually updated the application within days of a new release, and almost always with a matter of weeks, noting that at the individual level, these patterns are still affected by private work cycles.

These common experiences suggest a number of hypotheses regarding the relationship between releases and download counts. First, we expect there to be a relatively constant level of downloads that is effectively independent of new releases, as potential new users are driven by their own work cycles to try the software. Second, regular users will respond relatively quickly to new releases, causing significant increases in downloads during the days just following a new release. If a project is successfully converting experimenters into users, the response to releases will increase over time. Assuming the application does not achieve market saturation, the experimentation rate will remain constant or potentially increase over time if the application receives good publicity or word of mouth.

Figure 1 shows an idealized depiction of the trends expected in the downloads. The grey area shows the experimentation rate, depicted as basically regular but growing over time, perhaps as the result of increased publicity, or word of mouth. The white area under the curve shows the size of the installed base that regularly updates the software, which is also depicted as growing over time in an idealized successful project.

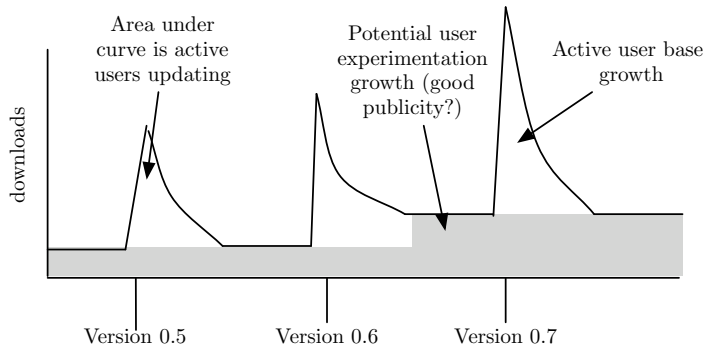


**Fig. 1** An idealized depiction of download and release trends. The grey area is potential user downloads, the white area under the curve is the active user base.

## 4 Data

The hypotheses proposed above can be validated using package release and download data from Sourceforge. These data are readily available on the Sourceforge site, but historical data on downloads is only available in monthly aggregates. Our hypotheses require daily data, however, which at any given time is only available for the past 60 days.

In order to build the daily time series data required for analyses like this one, the FLOSSmole project[4][3] has been spidering and storing the 60-day historical data as an aggregate measure for all packages of a project, and has accumulated daily download data covering approximately three years. Release data are available from the Notre Dame Sourceforge Research Archive (SRDA[4]) and release lists are available for each package produced by a project.

## 5 Descriptive Results

Figure 2 shows a daily time-series of BibDesk downloads plotted against releases of the primary package. There are four characteristics that are immediately apparent:

1. Pronounced download spikes following new releases,
2. Cyclic weekly effect causing regular saw-like patterns,
3. Relatively flat downloads in periods without new releases, and
4. Growth over time in both the post-release spikes and the flat periods.

[3] `http://ossmole.sf.net`
[4] `http://zerlot.cse.nd.edu/`

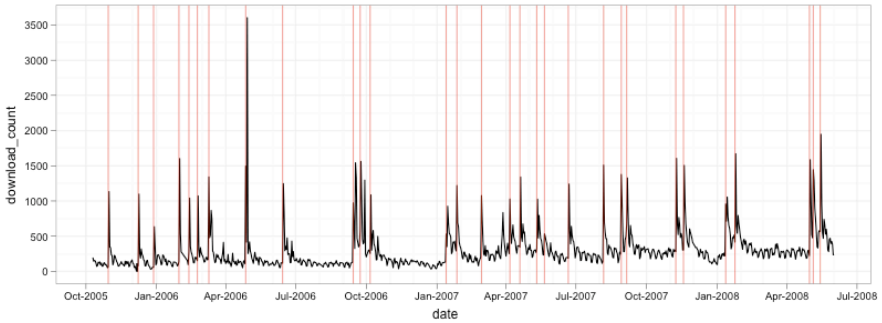


**Fig. 2** A daily downloads time-series for Bibdesk. Release dates for the primary package are marked with vertical lines.

The first notable pattern is a pronounced spike in downloads of several orders of magnitude, clearly associated with new releases of the software, as hypothesized. These appear to fall off predictably, assuming another release is not made within a short period of time. The pattern of decay appears to be near-exponential, with the bulk of downloads occurring within a matter of days, and the effect decaying completely within a week or two. When the releases are clustered in quick succession, the second release appears to restart the pattern.

The second pattern is a weekly cycle that generates a saw-like effect where downloads rise and fall within a relatively small amplitude. This effect is independent of releases, as demonstrated by release decay curves which also feature these saw-like patterns. Closer inspection of the data reveals that download rates fall while Sunday works its way around the planet, creating a lull in activity of approximately 48 absolute hours centered on Sunday evening GMT. This "Sabbath Effect" is so powerful that releases made during this lull appear to have slightly delayed spikes.

The third pattern we observed is the suppression of download rates when a release has not been made for some time, creating a plateau effect. Download rates in such periods show little change and are relatively flat, especially if the cyclic weekly effect is smoothed, e.g., by taking a moving average. A good example is the period near January 2007 and July 2007.

The fourth observation is that over longer periods of time, the absolute size of both the release spikes and the flat periods are subject to change, rising and falling dynamically. Overall, however, both the size of post-release spikes and the plateaus appear to grow in magnitude.

These patterns appear to confirm the hypotheses developed through participation in the BibDesk project. The combination of post-release spike-and-decay pattern, together with the flat downloads when no release has been made for some time, appears to be best explained through the understandings developed above. This spike represents regular, active users of the software updating their installed copies, with the decay showing that not all users update immediately, but that most do within a week or two. The flattening of the download series during the lull between re-

leases also supports this interpretation, and the levels at which the series flattens provide a theoretical baseline experimentation rate of first-time downloaders, which is hypothesized to be relatively independent of effects from releases.

Growth over time in both the spike-and-decay pattern suggests that BibDesk is successful at converting some of the experimenters into regular users, who then update the software after new releases, leading to progressively larger post-release download response. The strong growth in the average daily download rate during flat periods seems to indicate that the project is enjoying an increasing experimentation rate, which may relate to the rapid growth of the Mac OS X operating system during this period of time.

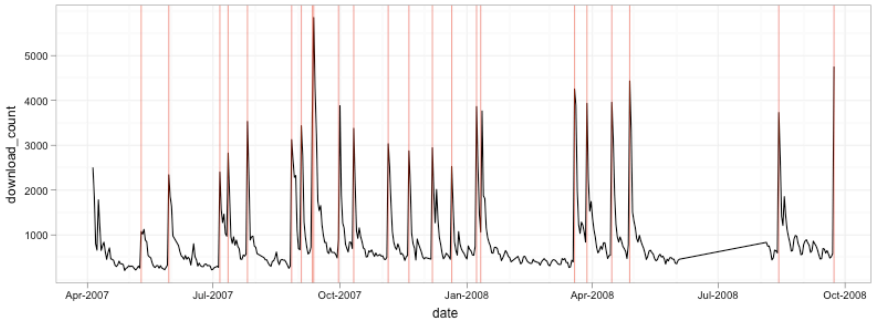


**Fig. 3** Skim-app downloads and releases. Note that Skim-app project began only in early 2007, so the time-period represented here is shorter than BibDesk.

Figure 3 confirms that similar patterns are present in the Skim-app project. The Skim-app project was recently founded in early 2007, so the time period represented in this figure is much shorter than that for BibDesk, and the shorter time period makes the weekly download cycle pattern more apparent. Missing data for July 2008 is also clearly visible in an abnormally straight line segment, which is a result of Sourceforge's difficulties with their statistics server combined with the order in which FLOSSmole spiders the pages. Compared to BibDesk, Skim-app also shows a more rapid release cycle, which often seems to abbreviate the full decay of the release effect. Overall, the project appears to show growth in both the release effect and the average download rates during the flat periods, although these trends are less apparent than with BibDesk.

Finally, it should be noted that both projects show spikes that appear to be larger than the usual weekday cycle effects, but are not directly caused by a release. In Skim-app, one of these anomalous spikes can be seen just prior to July 2007, and in BibDesk two instances are observed just prior to April 2006 and again in April 2007. We expect, but have not confirmed, that these are caused by one-time publicity events (e.g., being favorably reviewed in a blog), temporarily driving up the number of downloads to a rate that varies significantly from the usual experimentation rate.

## 6 Quantifying the user base

Using the model developed above of the patterns observed in time series of daily downloads, we are able to go further and quantify the size, over time, of both the installed base and the experimentation rate. These measures can ultimately be used together to estimate the rate at which these projects convert potential users to actual users. The resulting algorithm for determining the estimated active user base for a FLOSS project is simple to describe, but somewhat more complex to evaluate. We used a scientific workflow analysis tool to automate the process of retrieving data from two repositories, FLOSSmole and SRDA, and then plot releases and downloads together, confirming through inspection that the release-based triggering of user downloads is occurring and that it is therefore appropriate to utilize the user base estimate. A separate workflow calculates the required metrics to produce the measures and display trends as values in a time series plot.

The calculation of the two measures, experimentation rate and user base, requires daily download data for two time periods of equal length, centered around the sampled release. We only sampled releases where there were no other releases made during this entire time window, as we observed that such releases interfered with the decay of download rates to their normal experimentation levels. Unfortunately, this reduces the number of usable data points for each project, but it improves the validity of the results. We had to select the length of the period before and after the release for calculating the measures; we optimized our selection by comparing results for one week and two week periods, and found the single week period to provide the most consistent results.

We therefore considered the week prior to the release date as the baseline level of user interest; the daily download rates are averaged for these 7 days to generate a daily baseline measure. The daily downloads for the week after the release, starting at the release date, provide the data to estimate the active user base. We summed the daily downloads over these 7 days, and then subtracted the sum of downloads over the baseline period to eliminate the effect of normal daily experimental downloads, which continue regardless of the release.

This algorithm is applied to each selected release, allowing us to track changes over time in both the baseline level of user interest in the software, and in active user base. We express the function as follows: for a release date $R$, the mean $\mu_d$ of daily downloads $d$ over $[R-8, R-1]$, provides the baseline daily download rate. The active user base is then estimated by taking the sum of daily downloads after the release and subtracting the daily downloads prior to the release:

$$\sum_{d_R}^{d_{R+6}} d - \mu_d$$

## 7 Numerical Results

This analysis workflow results in two time series, one for each measure, as shown in Figure 4, for BibDesk, and Figure 5, for Skim-app. The results shown here use a 7 day period for calculating the figures. The baseline rate (the bottom dotted line) is a daily figure, while the user base (the upper solid line) is an estimate of total number of users that update regularly.

The baseline experimentation rate for both projects seems to be quite consistent, with a trend towards growth recently in Skim-app. The mean daily rate for BibDesk is 121 downloads, while for Skim the mean is 480. This makes sense because BibDesk has a smaller potential market than Skim-app, since it is of limited use outside of BiBTeX users, while Skim-app is useful to anyone working with PDFs.

In contrast, the calculation for installed base (the upper solid lines in Figures 4 and 5) displays substantial variance, particular with Skim-app. This result suggests that our algorithm is not completely successful in removing noise, since the installed base would not be expected to change so radically in a short period. Further smoothing of these data might provide more usable results. Nonetheless the overall patterns and sizes appear to be reasonable. The higher experimentation rate of Skim-app, combined with an overall growing active user base suggests that Skim-app is set for faster growth than BibDesk. BibDesk may in fact be experiencing declining numbers of active users, perhaps as a result of competition from other reference managers, such as Zotero, which are better suited to non-LaTeX document preparation.

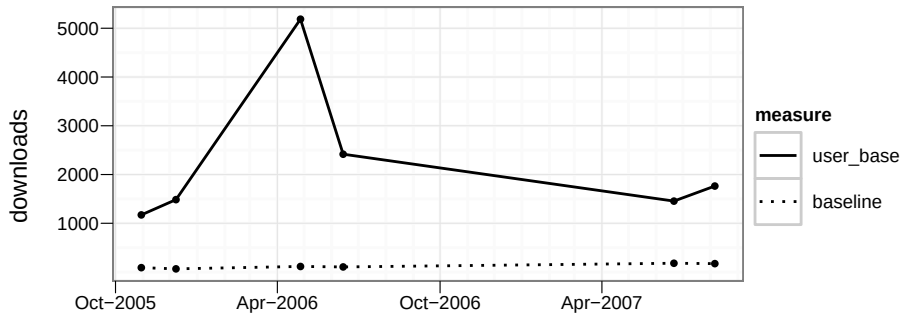


**Fig. 4** BibDesk: Measures of regular updaters (user_base) and potential users (baseline)

## 8 Discussion

The measures and analysis developed in this paper are a promising improvement in measurement of software use for FLOSS software. The measures have substantial face validity over raw download figures, which is increased by the authors' direct experiences with the projects. Nonetheless, the measures are subject to some serious limitations.

The first set of limitations is the reliance on download data as a basis for the measure. While readily available, downloads figures are known to be problematic for a number of reasons [1]. First, the data sources may be unreliable. Sourceforge, for example, has had persistent issues with their statistics server, and has periodically revised their historical figures. In addition, variations in counting mechanisms make it inadvisable to draw comparisons across different project forges. Finally, because the currently available downloads figures are at the project level rather than the package level, our approach will work only when the project has an unambiguous primary package that accounts for a very substantial portion of the downloads. Projects having multiple packages with similar levels of download popularity would certainly confound the method.

A second limitation is in the calibration of the durations used, in this case one week, for measuring both the installed base and the experimentation rate for each period. Varying this period leads to substantial changes in the measures, and would lead to different interpretations of the patterns. At the same time, the rate at which regular users download software after a release seems to vary significantly as well. Skim-app's post-release download fervor was typically over much more quickly than BibDesk's, which sometimes took close to three weeks to return to a relatively stable rate. These differences may reflect more frequent use of Skim-app compared to BibDesk, leading to quicker updates. A more sophisticated approach would at-

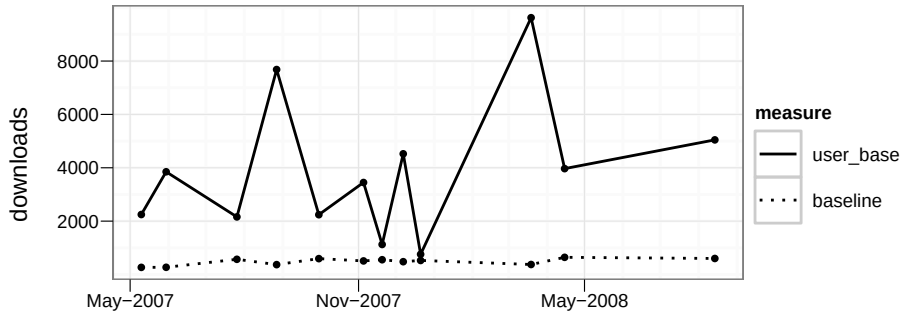


**Fig. 5** Skim: Measures of regular updaters (user_base) and potential users (baseline)

tempt to model the download time series with parameters for weekly, release and update intensity effects.

There are also unresolved questions about the generalizability of this method. The conceptualization of the underlying processes relies on the assumption that users will move quickly to update their software. This seems likely true only for software that has everyday use, and where extension, rather than stability, is the emphasis of the project's community efforts. Projects which produce irregularly used software, such as conference organization systems, or software where stability is a strong concern, such as Ruby on Rails or Python, would not be expected to produce the same patterns. Similarly, server software might be upgraded much more deliberately than end user software. It is also important to note that one must be careful in comparing downloads between different types of software products, since their potential market sizes are very different. Web server software, such as the Apache HTTP Server, has a much larger audience than bibliographic software like BibDesk.

For these reasons, we do not recommend using this analysis approach without first inspecting the download and release time series for evidence of consistent release responses. Alternately, assuming that the software is expected to be regularly downloaded, a lack of this pattern could be taken as prima facie evidence that the project makes its software available in different ways. In this respect, the method would therefore provide a way to check the validity of using development forge download figures alone for estimating active user base for that project.

## 9 Conclusions and Future Work

In conclusion, this paper makes methodological contributions by developing new measures, and reports empirical results from evaluating the measures:

- We introduce and evaluate a new measure for estimating baseline user interest in FLOSS projects.
- We introduce and evaluate a new measure for estimating active user base for FLOSS projects.
- We apply these measures in time series analysis of two FLOSS projects, finding that the overall levels of both measures show good face validity. As a longitudinal time series, the baseline measure also shows good face validity, but the active user base measure displays surprising variance.

The measures developed in this paper can be used wherever a dependent variable of project popularity is called for, such as cases where popularity is incorporated with other metrics into an overall measure of project success. Future work could evaluate these findings against a more dynamically selected time range for these measures, as there may be room for improvement in the selection of time periods for the baseline download rate and the post-release period, as the time windows that we selected are somewhat arbitrary, based on an heuristic from evaluation of a small sample. A slightly more complex function might determine the length of

the post-release period based on the time required for daily downloads to return to a rate within a standard deviation of the baseline rate. This method might offer a more precise measure, but would potentially also be more sensitive to variations in software release patterns by project.

A limitation of our current study is that we do not have an independent estimate of the user base for the two projects studied against which to calibrate our measure. Future research should apply our measure to more projects and in particular, to projects for which other estimates of users can be made in order to assess the validity of the measure.

Finally, this method may also apply to non-FLOSS downloaded software which is regularly updated, such as shareware or other regularly updated software, including iPhone applications. Applying this estimation method is limited only by the reliance high-resolution download figures and release dates, as well as the assumptions discussed above.

## References

1. Crowston, K., Howison, J., Annabi, H.: Information systems success in free and open source software development: Theory and measures. Software Process: Improvement and Practice **11**(2), 123–148 (2006). DOI 10.1002/spip.259
2. Crowston, K., Wei, K., Howison, J., Wiggins, A.: Free/libre open source software development: What we know and what we do not know (2008). Under Review
3. Delone, W., Mclean, E.: Information systems success: The quest for the dependent variable. Information Systems Research **3**(1), 60–95 (1992)
4. Howison, J., Conklin, M., Crowston, K.: FLOSSmole: A collaborative repository for FLOSS research data and analysis. International Journal of Information Technology and Web Engineering **1**(3), 17–26 (2006)
5. Scacchi, W.: Free and open source software development: Recent research results and methods. In: M. Zelkowitz (ed.) Advances in Computers, vol. 69, pp. 243–295. Elsevier Press (2007). DOI 10.1016/S0065-2458(06)69005-0
6. Seddon, P., Staples, S., Patnayakuni, R., Bowtell, M.: Dimensions of information systems success. Communications of the Association for Information Systems **20**(2), 61 (1999)