Reclassifying Success and Tragedy in FLOSS Projects

Andrea Wiggins and Kevin Crowston

Abstract This paper presents the results of a replication of English & Schweik's 2007 paper classifying FLOSS projects according to their stage of growth and indicators of success. We recreated their analysis using a comparable data set from 2006. We also expanded upon the original results by analyzing data from an additional point in time and by applying different criteria for evaluating the rate of new software releases for sustainability of project activity. We discuss the points of convergence and divergence from the original work from these extensions of the classification and their implications for studying FLOSS development using archival data. The paper contributes new analysis of operationalizing success in FLOSS projects, with discussion of implications of the findings.

1 Introduction

Much of the empirical analysis of FLOSS has been undertaken using bespoke data sets laboriously created for a single analysis. However, over the last few years, research teams have developed several repositories of FLOSS data that provide reliable curated data about FLOSS projects (4; 7; 9). Use of data from these repositories relieves researchers of the need to spider and parse data from project repository sites, increasing productivity while also avoiding errors from problems in the data collection processes. These repositories of repositories (RoRs, (8)) are seeing increasing use by researchers. Much of the prior research that employed large-scale data sources should be possible to recreate and extend using the data from RoRs, allowing the research community to build more quickly on past work to refine theories and methods in FLOSS research.

Syracuse University School of Information Studies
Hinds Hall, Syracuse NY 13244 USA
e-mail: awiggins@syr.edu, e-mail: crowston@syr.edu

In this paper, we adopt this approach in replicating English & Schweik's (3) classification of project success and failure in open source projects. In this paper, we do not engage in a detailed critique of the classification; rather our goal is methodological development in the area of large-scale analysis of archival data from FLOSS repositories. We note though that objective methods for identifying FLOSS project success and failure is a topic of interest for both researchers and practitioners. Researchers need to identify successful and failed projects to be able to investigate the potential causes of success or failure. Practitioners are interested in being able to evaluate success for several reasons: first, this gives an individual decision information with respect to whether to rely upon or become involved with a given software project; second, it gives software foundation decision-makers useful information for determining whether to admit projects or invest resources into developing them; and third, it provides an assessment of the health of projects in which individuals and larger organizations are currently engaged.

In the following section, we describe the classification developed by English and Schweik (3). We then outline the methodology we adopted to recreate their results using data from the Notre Dame SourceForge Research Data Archive (SRDA) (9). We discuss our results in relation to the original work, examine the outcomes of varying a single classification criterion and look at the changes to project classifications over time. We then reflect on the methodological challenges involved in replicating large-scale analysis of archival data on open source projects. Finally, we conclude with directions for future work.

2 Theory: Assessing project success

Crowston et al. (2) note that for FLOSS projects, success is a multi-dimensional construct that can be assessed from many perspectives. The original classification by English and Schweik presents a set of six classes of FLOSS projects, operationalizations for which are reproduced in Table 1 from Table 1 in (3). English and Schweik developed the original criteria for their classification based on interviews with FLOSS developers and the thresholds for the classification originated with their initial manual coding of a sample of projects. (The original paper provides the full rationale for their definitions.) The classification has two facets: the stage of the project, either initiation (I, first year of the project or up to three releases) or growth (G, subsequent to initiation thresholds); and the outcome, either success (S) or failure, which English and Schweik labeled as "tragedy" (T) in reference to the tragedy of the commons. In addition, projects might be labeled as in an indeterminate state (I) if success or failure cannot yet be determined. Projects were classified based on a number of factors, including age, releases and their timing, and downloads, which serves as a proxy measure for the creation of useful software. Finally, projects were labeled as unclassifiable if there is evidence that they may have distribution channels other than SourceForge, suggesting that the download or release count data are unreliable. The final column of Table 1 shows the operationalization we adopted in our

reimplementation of the classification. In many cases, our criteria are identical, but we discovered a few necessary changes, as discussed below in the methods section.

Table 1 Six FLOSS success/tragedy classes and their methods of operationalization, from English & Schweik (2007).

Class/ Abbreviation	Definition	Original Operationalization	Re-operationalization
Success, Initiation (SI)	Developers have produced a first release.	At least 1 release (Note: all projects in the growth stage are SI)	*Not explicitly classified: Sum of IG, SG and TG*
Tragedy, Initiation (TI)	Developers have not produced a first release and the project is abandoned.	0 releases AND $\geq$ 1 year since SF project registration.	Same
Success, Growth (SG)	Project has achieved three meaningful releases of the software and the software is deemed useful for at least a few users.	3 releases AND $\geq$ 6 months between [all] releases AND does not meet the download criteria for tragedy detailed in the TG description below.	$\geq$ 3 releases AND $\geq$ 6 months between most recent and third most recent release AND $>$ 10 downloads
Tragedy, Growth (TG)	Project appears to be abandoned before producing 3 releases of a useful product OR has produced three or more releases in less than 6 months and is abandoned.	1 or 2 releases AND $\geq$ 1 year since the last release at the time of data collection OR $\leq$ 10 downloads during a time period greater than 6 months starting from the date of the first release and ending at the data collection date OR 3 or more releases in less than 6 months AND $\geq$ 1 year since the last release	1 or 2 releases AND $\geq$ 1 year since the most recent release OR 3 or more releases AND $\geq$ 1 year since most recent release OR $\leq$ 10 downloads[1]
Indeterminate, Initiation (II)	Project has yet to reveal a first public release but shows significant developer activity.	0 releases AND $<$ 1 year since SF registration	Same
Indeterminate, Growth (IG)	Project has not yet produced three releases but shows development activity OR has produced 3 releases or more in less than 6 months and it has been less than 1 year since the most recent release	1 or 2 releases AND $<$ 1 year since the most recent release OR 3 releases AND $<$ 6 months between releases AND $<$ 1 year since the most recent release	Same

[1] Note: we used all-time downloads as this was operationally the same when combined with the release rate criterion

2.1 Propositions

Turning to the substantive content of the paper, we present several propositions related to the three main areas of analysis, both methodological and theoretical.

1. First, we expect that our classification drawing on repository data will produce comparable results to those reported in the original work, possibly with some minor variations due to differences in sampling.
2. Second, we expect that the three variations in the classification criterion for the rate of releases, an indicator of project stability, will result in differences in classification, though these differences will be limited to specific classes, as not all classifications rely on this data.
3. Finally, we also expect to see change over time in the classification applied to individual projects, based on predictions about the potential next states of projects based on their current classification. For example, no project which has advanced to a growth stage (SG, TG, IG) can return to an initiation stage classification (TI, II). Projects that are II will not remain in that state for longer than a year, by definition, but may progress to any of the other classifications. In all cases where a project may become either a success or a tragedy in the next classification, we expect to see success less often than tragedy. As a result, as the number of projects grows, most classes should grow proportionally, but not all. We expect that over time, the number of tragedies will increase as a matter of accumulation of failures in the population as a whole, although we expect that the relative proportion of successes will remain stable. Further, we expect that the effects of time will lead to larger proportions of projects identified as being in the initiation stage, as the number of new projects grows, and this will in subsequent time periods lead to a slow increase in tragedies at the initiation stage.

3 Methods

The main features of the large-scale archival data analysis consisted of replication and extension of the original work. We applied an eScience strategy to conducting the analysis with respect to our choices of data sources and tools, and in the process created research artifacts in the form of processed data and analysis workflows that will support further extension of this work.

3.1 Replication

The replication of the original analysis required processing data from approximately the same time period to provide a suitable comparison. The extension of the work was intertwined with the replication, and involved preparing additional data for anal-

ysis of change over time along with the addition of new variations of the classification. As our goal with this research was methodological refinement, we selected and analyzed additional dates and variations to evaluate the performance of the classification, rather than to make a theoretically-informed evaluation of change to the community composition itself.

3.1.1 Data

In the original work by English and Schweik, the authors developed and tested their classification algorithm using FLOSSmole data from 2005, subsequently creating a sampling frame from the FLOSSmole project list of August 2006. They then spidered SourceForge around 16 October of 2006 for release information to augment the FLOSSmole data. However, the data spidered by English and Schweik do not include statistics for over 8,000 projects with incomplete data or that were deleted by SourceForge between August and September. The gap between collecting project data and release data might also invalidate a portion of the original analysis, as the period of up to two months between data collection times affects values for the thresholds for a project's active lifespan, achievement of maturity and release rate. For example, projects that made a first release in August or September could be misclassified as TG instead of IG.

The goal of our work was to demonstrate the use of a shared data repository for replication of the original research. Because of the limitations on the available data from FLOSSmole, as noted by English and Schweik, we selected SRDA as the data source, as it contained all of the necessary data for the classification at the time. In addition, we note that the SRDA data comes straight from SourceForge as a monthly database dump, which makes it an authoritative data source. FLOSSmole data are parsed from SourceForge HTML pages, and while the repository provides a reliable data source, its contents are one step removed from the original source. We analyzed data for October 2006 to match the data collection for releases in the original work; more specifically, we used the October 2006 release of SRDA data, which was captured on 23 October 2006.

3.1.2 Analysis

The analysis was replicated by careful examination of the original English and Schweik article, from which the requirements for data and processing were derived. A workflow for the data processing was developed in Taverna, a scientific workflow tool (1; 6). This approach made it easy to integrate data coming from several different sources, e.g., from the SRDA and a local cache of release data (previously retrieved from the SRDA), and to modularize the analysis steps. The analysis was implemented in several phases: a first phase to retrieve the basic data needed about project downloads and releases, followed by phase that ran simple tests on criteria

to determine whether thresholds had been met, and finally a stage that classified the projects according to these tests.

For the replication of the classification, we found it challenging to determine the correct data selection or parameters for classification from the text of the English and Schweik article. Fortunately, the workflow implementation allowed us to remain flexible in our data selection and to parameterize the analysis. As we sought to reproduce the classification table in the published article in the format of a truth table to achieve completeness and exhaustiveness in the classification, we discovered that the classification as published was not complete. Some of the negative cases were not included in the published table, which is not to say that the authors did not consider these cases, perhaps because those combinations did not arise in the original data sample. For the most part, we were able to fill in the appropriate classifications for negative cases based on inheritance from other criteria for each class.

We also faced slight differences in the data available from the SRDA that required some changes to the operationalizations. For example, the SRDA did not have a convenient source for downloads within 6 months, though in the cases where that count was required, it was functionally equivalent to all-time downloads, which we used instead. The final column of Table 1 shows the operationalizations we developed and implemented in the workflows.

After debugging and verification of the workflow's performance with a subset of data, the final analysis was run with a full replicated data sample and compared to the original published research. In production, we eventually substituted SQL queries for the final two phases of the classification workflow due to significant economies in processing time. The analysis of the resulting data was conducted with R on the classified data stored in a SQL database. To encourage reuse of these data and analysis approaches, the workflows, SQL scripts and classification data will be made available to other researchers from our website, `http://floss.syr.edu/`.

As a result of using the SRDA data, the time required to retrieve and pre-process data were much lower than reported in original article. English and Schweik spent 22 days to spider the data for the analysis that were not already available in FLOSSmole. After optimizing our processes, we found that retrieving and preparing the data for classification requires less than 30 hours for data sets approaching 150,000 projects. In future efforts, this reduction in processing time will allow us to develop more granular analysis of changes to project classification over time by generating analysis-ready data for each monthly release of data from the SRDA.

3.2 Extending Analysis

In addition to replication, we wished to extend the prior work by examining stability of classification status over time, and evaluating the performance of two alternate means of operationalizing the rate of release.

3.2.1 Additional dates

Our data processing times were significantly lower than reported for the original work, so we were able to analyze data for more than one date. Preparing data for analysis is the most time consuming part of the process, but by storing the prepared data, we are able to reuse it more readily with alternate classification schemes. In addition to the original date, we analyzed data from April 2006 to provide a short-term comparison that would help evaluate the stability of the classification over a relatively short period of time. We selected this time period because over a period of six months, there is opportunity for many projects to change status on several of the indicators, thereby affecting their classification status.

3.2.2 Implementing English & Schweik's future work

One of the more complex aspects of the original analysis is the qualification of release rate as an indicator of the sustainability of project activity. The assumption here is that projects which make releases too quickly cannot maintain the pace of activity; what this fails to take into account, however, is the wide diversity of release strategies employed by FLOSS projects. The original measure evaluates release rate by whether the project has had at least six months elapsed between first and last releases, which automatically privileges older projects rather than more stable projects. One solution, inspired by English and Schweik's discussion of these issues, is setting a threshold for the amount of the time between the most recent series of releases, rather than for all releases. A second option is to evaluate the average time between each release against a threshold, which applies the lifetime perspective of the original method, but seems likely to be more stable than the alternative that evaluates the time between only the most recent releases. We have implemented both of these variations, along with the original version, to evaluate the influence of this factor on project classification.

4 Results

We discuss the results of our analysis with three comparisons: comparison to the original published results, comparison of results from varying one classification criterion, and comparison of classification over time.

4.1 Comparison to Original Published Results

Our results for data from October 2006, using the same default values for the classification thresholds, are compared with the original results for the same time period

in Table 2. We note that as percentages of the classified projects, our results are similar for the classes of II, IG, TG, and SG. They are also remarkably close in values for the number of "unclassifiable" projects.

Potential causes for variation in results could be discrepancies in the release and download data, which English and Schweik retrieved from different sources and at different times, as previously discussed. Variations in release data in particular would be problematic, as this could affect the determinations of whether or not the project is active, whether it has had enough releases, and whether or not the releases have occurred too quickly to be considered sustainable.

Table 2 Comparison of classification results to original results from English and Schweik.

Class	Original results	Replication Results	Difference
unclassifiable	3,186	3,296	+110
II	13,342 (12%)	16,252 (14%)	+2,910 (+2%)
IG	10,711 (10 %)	12,991 (11%)	+2,280 (+1%)
TI	37,320 (35%)	36,507 (31%)	-813 (-4%)
TG	30,592 (28 %)	32,642 (28%)	+2,050 (+0%)
SG	15,782 (15%)	16,045 (14%)	+263 (-1%)
other	8,422	—	
Total	119,355	117,733 (+ 9.6%)	

Another variation between these results is that we produced no "other" classifications. We did not run into problems with differences between sampling frames and actual data that we were able to collect, as there was no delay between sampling and data collection. More specifically, we have not sampled so much as taken a census, as we have used all of the available data for each time period. The discrepancies in total numbers of projects, approximately 1,600 fewer in our sample, could also result from the deletions of inactive projects that the authors cited as a cause for the "other" projects; however, we were able to classify a larger number of projects overall. The date for the SourceForge dump upon which our analysis is based is slightly later in the month of October than the original analysis data collection time period (and two months later than the collection of project statistics), but in our case, we have no record for projects that were deleted from the SourceForge system. Overall, we consider the replication successful, as the greatest variation in classification by proportion is in the TI category, with a relative difference of just 4%.

4.2 Comparison of Release Rate Criteria

As discussed previously, we implemented three different variations for judging the sustainability of the rate at which a project is making releases. The original article called for a period of at least 6 months between the most recent release and the

first in the window of three releases. English and Schweik mentioned examining the time between each release, and based on this idea, we implemented a density-based calculation of the time elapsed over the most recent three releases (Method Two). The final variation compares the average time between all releases in a project against a threshold (Method Three); notably, this is a more strict definition than the original and may merit a different threshold value. The original implementation allowed an average of three months between releases in the case of the minimum qualifying number of releases for evaluating success; the results are reported for a six-month threshold in Table 3.

Table 3 Classification outcomes from varying release lag measures for each time period, using a six month threshold.

2006-10-23	Method One	Method Two	Method Three
IG	12,991 (11%)	14,310 (12%)	19,235 (16%)
II	16,252 (14%)	16,252 (14%)	16,252 (14%)
SG	16,045 (13%)	15,426 (13%)	3,143 (3%)
TG	32,642 (28%)	31,942 (27%)	39,300 (33%)
TI	36,507 (32%)	36,507 (32%)	36,507 (32%)

It is clear that variations in this criterion result in a reclassification of SG projects as IG or TG projects. This occurs when the release lag evaluation comes out as "too fast" to be considered sustainable (IG), or the project has not made a release in the last year (TG). In all other ways, these projects are judged successful according to the other classification criteria—they have achieved "enough" releases and the software has been downloaded. This reclassification effect is exaggerated in the comparison between Method One and Three because the difference in calculation method suggests that a different threshold value should have been used.

While this change to the release rate criterion seems to have a small effect with the recent release density function (Method Two) and a larger effect with the averaging over all releases function (Method Three), comparison of the project-level classifications tells another story. We find that even at the six-month threshold, Methods One and Three are most consistent with respect to results; in every case, the changes to a project's classification is a transition from SG to IG or TG. However, applying Method Two yields changes from SG to IG as well as from IG to SG, and likewise with TG. In addition, more classifications are changed with Method Two than with Method Three, so the apparently smaller change in summary statistics masks a larger change in classification at the project level.

4.3 Comparison Over Time

The comparison of classification results over time suggests interesting directions for future analysis at a more granular level. Table 4 compares the counts of projects at

two points in time, April and October of 2006. The consistency in SG classifications over time demonstrates underlying regularity with respect to the proportion of projects which can meet the classification criteria for success. We also see relative stability in the IG and II classes. We also observe growth of the TG class over time as it gradually accumulates failures, as we would in fact expect.

Interestingly, the changes between these two periods show small increases in the proportions of most classifications, with a notable decrease in the frequency of the TI classification. This suggests that more projects are successful at making at least one release than odds would suggest. All other things equal, we might expect to small net increases across all categories, but this would not take into consideration the compound effects of change over time; the decrease in TI projects would suggest that there is a substantial number of projects which require more than one year to make their first release. Logically, these projects are most likely to become TG or IG projects.

Table 4 Classification outcomes from different time periods, using the original release rate criteria.

Class	2006-04-21	2006-10-23
IG	12,166 (10.8%)	12,991 (11.0%)
II	13,592 (12.4%)	16,252 (13.8%)
SG	14,244 (12.7%)	16,045 (13.6%)
TG	28,777 (25.6%)	32,642 (27.7%)
TI	39,948 (35.5%)	36,507 (31.0%)
unclassifiable	3,343 (3.0%)	3,296 (2.8%)
Total	112,430	117,733

To describe the changes in project state between two points in time, we present the Markov model shown in Figure 1 that shows the percentage of projects that shift from one classification in April 2006 to a different one in October 2006. New projects are not included. Omitted from the diagram are the rates for projects remaining in the same classification: IG at 54%, II at 56%, SG at 100%, TG at 98%, and TI at 98%.

Initial observations from the model include the fact that once a project is classified a tragedy, it has a very low likelihood of escaping that classification. A TI project has a 1% likelihood of salvaging itself, while a TG project has a 2% chance– in both cases, these are not very good odds for survival. Likewise, once a project is labeled SG, there is no rescinding this title; this is inherent in the operationalization, because once the thresholds have been reached (adequate releases, downloads, an appropriate amount of time between releases), there is no criterion that will reverse them.

It comes as little surprise to see that the most common path from II, the default classification for a newly founded project, is to TI; for a new project, tragedy is four times more likely than moving on to growth in the IG class, with the potential for success. This confirms the common assertion that many projects are stillborn

and never make a release, failing nearly immediately. IG projects are three times more likely to become tragedies than successes, but notably, this is the only route to success, and no project is an overnight success.

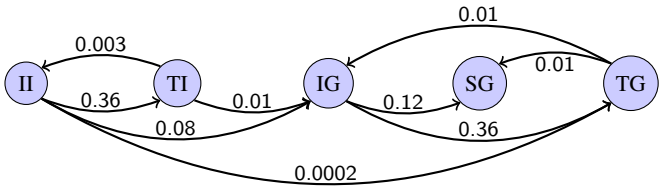

Fig. 1 Markov model showing changes in project classification over a six month period in 2006.

5 Discussion

In this section, we discuss the implications of our findings and areas for future work. As the focus of this paper was primarily methodological, so are the implications. These are related to both the nature of the task for large-scale archival research, and to the substance of the task, classifying the successes and tragedies among FLOSS projects.

Large-scale analysis of FLOSS repositories sounds simple in conversation, but is never so straightforward in practice. This effort built on prior work that eased the development of data handling scripts and functionality, and yet demonstrated time and again that this style of analysis requires thorough knowledge of the data sources and analysis operations. Exceptions in the data source can wreak havoc with automated processing; while the data can simply be processed again, this can be very inefficient. Although it may only require 30 hours to analyze a snapshot of the entire SourceForge population, unexpected variations in source data realistically double or treble the time required to prepare data, once data verification procedures and troubleshooting are included, so permitting adequate time for development of the data handling is essential to this type of analysis. In the process of managing the flow of data between multiple workflows, or between workflows and other semi-automated processes, we also found that it is particularly helpful to use exactly the same names for variables in every location where they are used, from the database fields to the workflows and R scripts.

It is apparent that the challenges of working with the data sources have not changed significantly since the original classification by English and Schweik, although our tools and strategies are somewhat improved. We found that workflows were particularly good for dealing with retrieving and combining data from diverse sources when a single SQL query is not feasible. When preparing the data involve frequent and repeated format conversions (e.g., epoch times to SQL times and back

again), sophisticated selection criteria (e.g., the most recent three releases), and careful handling of missing values, workflows are a better solution than most alternate procedures, particularly as they can be applied in precisely the same ways many times over, despite the complexity of the task.

One complication to analysis, as others have observed (5), is that the repository data structures represented in the SRDA can and do change over time. For example, all-time downloads data (used in the classification) are no longer available from the most recent monthly SRDA dumps, so a single analysis script will not work on data from different periods, complicating longitudinal analysis.

In addition to allotting more time for data management than may seem reasonable to expect for an automated process, we recommend having a set of test data for testing scripts as they are developed. While this may seem obvious, running a complex analysis on the same data set makes it much more straightforward to identify causes of unexpected results or inconsistencies. In particular, a manufactured data set containing a full range of edge cases can be useful in detecting errors in the script before scaling up analysis.

Although we have been tried to ensure that our algorithm matches the logic of the original paper as precisely as possible, we noted a number of cases that do not logically fit into the classifications that they are assigned. This kind of effect is difficult to trace back to its causes, which may include any of several issues such as bad data, incorrect implementation of the classification or a missing case in the classification. We suspect that some of the exceptions we have observed are cases that are illogical combinations of criteria that were not explicitly addressed in the original definition of the classifications. For example, the original classification does not cover projects that have downloads without releases (seemingly impossible), or how to classify a once-successful project that has long since fallen inactive. There would be no reason to expect that these configurations exist until they emerge as anomalies in the analysis, noticeable only through comparison to the expected results.

Finally, while our focus has been on methodological issues, our analysis does suggest some possible improvements to the English and Schweik classification. First, the influence of release rate and count thresholds is significant, and these are relatively dynamic measures compared to the other criteria, as our evaluation of variations in the release rate criterion demonstrated. Successes were reclassified as tragedies or as indeterminate; those which had not made a release in a year became tragedies, while those whose release rate was too swift became indeterminate. This suggests that additional testing and refinement of release-based criteria is primary task for improving the classification.

Second, as Figure 1 shows, a project classified as a success remains a success, even if it becomes inactive. This is a conservative classification choice, reflecting the reality that a successful project may not continue releasing new software indefinitely, but may enter a third stage of "retirement" in which it is still useful but is no longer under active development. This is a different state than one-time successes which are later abandoned and fall into disrepair; finding a way to distinguish between these cases would add significant nuance to our understanding of the lifecycle of open source project development.

5.1 Limitations

The results reported here are limited by the data sources and analysis methods, which are conversely also strengths in this analysis. This study is limited in generalizability in the same ways as the original work: neither result can be generalized to other repositories beyond SourceForge, and both are subject to flaws in the data sources. Both apply reductionist methods for operationalizing heuristics that are expected to indicate the development and success of FLOSS projects. While this mode of analysis has the advantage that it can be applied broadly, it also loses some face validity in broad application due to the existence of numerous individual examples (which may or may not be edge cases) that defy the assumptions embedded in the categorization of projects.

Limitations specific to this analysis include the change in data sources from the original, which introduces potential sources of error; however, we believe the SRDA data to be no less authoritative than combined FLOSSmole and bespoke data. Although we did not test additional variations on most of the parameters with the type of sensitivity analysis that large-scale analysis permits, doing so is simply a matter of choosing parameters and allotting processing time.

Finally, any classification of project success and failure will be challenging to validate empirically. English and Schweik developed their criteria for classification based on interviews with developers. However, once this classification has been applied to thousands of FLOSS projects, empirical validation becomes particularly challenging, as there is no established success metric against which to objectively evaluate the results. Feedback from the developer community on the results of the classification would provide a measure of validation; however, this method does not scale effectively. We note this limitation to both our work and the original classification as an opportunity for further development of the research on success in FLOSS projects.

5.2 Future Work

Replicating this classification using methods specifically oriented to further reuse of the data and analysis makes it feasible to consider a wide range of potential extensions to this work. More exhaustive testing of the threshold values is the most apparent direction for further refinement of the classification. In addition, taking advantage of the infrastructure to evaluate alternate measures of the classification criteria would permit the development of more sophisticated measurements. Just as we have tested variations on the release rate criterion, another possible variation could implement a function to adjust the threshold for downloads based on project lifespan or number of releases, which might better account for the current usefulness of the software.

While the operationalizations vary by the degree to which they capture the definitions, it is a nontrivial challenge to acquire and prepare data that would permit more

accurate operationalization. For example, the definition of Indeterminate Initiation (II) is that the project has no public release but has significant developer activity; the operationalization is that the project has no releases and was founded less than a year before the data collection date. A more ideal operationalization would include explicit evidence of developer activity, such as communications or CVS activity. However, neither the original analysis nor our replication makes use of such data. The complexity of integrating additional data sources to produce such an analysis has been a barrier to developing more nuanced analysis, but our methodology and research artifacts can provide an extensible foundation for future work with more sophisticated measures.

As suggested by English and Schweik, incorporating data from CVS, email lists, and fora would create new potential for evaluating project activity. Rather than classifying projects as active or inactive based on having made a release within the last year, they could be classified based on the relative level of activity across a variety of channels for participation. While these data are available, incorporating them is tricky, and reshaping the classification criteria to make use of a greater variety of data would pose an interesting challenge. Refining a classification scheme such as this has the inherent problem that there is no objective way to determine what criteria are "best." We suggest that changes should be made based on improved congruence between definition and operationalization, and robustness of the measures to perturbation.

Finally, future work could more closely examine the shifts in classification over time. This effort would serve two goals: first, to further optimize the classification by identifying criteria that are more dynamic and potentially less representative, and second, to identify common developmental trajectories of FLOSS projects by charting their classifications across time. While the second goal is in many ways more attractive for researchers than the first, we note that refinement of the classification is key to generating valid results that can help us understand the implications of changes to project status over time.

6 Conclusions

In this paper, we replicated a classification of FLOSS project success and tragedy for all projects hosted on SourceForge at two points in time. The contributions of the work include the extension of the analysis, both in methods and in data analyzed. We extended the analysis to test three different methods of evaluating release rates; we tentatively suggest that the method that applies the average time between releases is the most stable and consistent with the intent of the analysis. We analyzed the data for two time periods, finding that the proportion of successful projects remained steady, while the number of tragedies appears to slowly increase over time in greater proportions than overall growth in sample size would predict. The implications of the work include recommendations for best practices for conducting

large-scale analysis of archival FLOSS project data, and several suggestions for future work to develop the classification into a robust tool for research and practice.

Acknowledgements The authors gratefully acknowledge James Howison's extensive contributions to the data and analysis infrastructure employed in this research. This research was partially supported by the United States National Science Foundation CRI Grant 07–08437.

References

[1] Taverna project website. URL `http://taverna.sourceforge.net/`

[2] 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)

[3] English, R., Schweik, C.: Identifying success and tragedy of FLOSS commons: A preliminary classification of Sourceforge. net projects. In: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development, p. 11. IEEE Computer Society (2007)

[4] 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)

[5] Howison, J., Crowston, K.: The perils and pitfalls of mining SourceForge. In: Proceedings of the International Workshop on Mining Software Repositories (MSR 2004), pp. 7–11 (2004)

[6] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., Oinn, T.: Taverna: A tool for building and running workflows of services. Nucleic Acids Research **34**(suppl 2), W729–732 (2006)

[7] Robles, G., Koch, S., Gonzalez-Barahona, J.: Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), Edinburg, Scotland, UK, pp. 51–55 (2004)

[8] Sowe, S., Angelis, L., Stamelos, I., Manolopoulos, Y.: Using Repository of Repositories (RoRs) to study the growth of F/OSS projects: A meta-analysis research approach. International Federation for Information Processing **234**, 147 (2007)

[9] Van Antwerp, M., Madey, G.: Advances in the SourceForge Research Data Archive (SRDA). In: Fourth International Conference on Open Source Systems, IFIP 2.13 (WoPDaSD 2008). Milan, Italy (2008)