Evaluating the Health of Open Source Components

Ivan Zaytsev[a,*], Slinger Jansen[a]

[a]*Department of Information and Computer Sciences, Utrecht University, Utrecht, The Netherlands*

Abstract

Context: Implementing open source components into commercial applications has many advantages for software developers. However, an unforeseen decline in health of the supplying community can lead to a number of complications or large expenses, caused by transition costs to an alternative software component. Successful product managers must be able to assess the health of the open source communities their applications depend on.
Objective: In this paper we present a modular method for software product managers that allows them to assess the health and vitality of open source communities.
Method: The research is founded on a systematic literature review on the topic of open source and software ecosystem health, as well as a case study at a software firm with extensive open source experience.
Results: The main research result is an Open Source Component Health Analysis Method that can be applied and fully customised by software product managers. The method is based on a list of open source vitality indicators, as well as an open source interaction model, including the role of commercial patronage in contemporary open source communities.
Conclusion: Recent appearances of commercial patronage appear to dilute the classical distinction between voluntary private contributions to open source and software development for commercial software firms. The introduced method presents a new and structured approach to open source vitality analysis and can help product managers to increasingly implement open source in their products.

Keywords: open source, vitality analysis, health analysis, software ecosystem

1. Introduction

Contemporary, large software products are frequently developed by an entire ecosystem of organizations and open source communities (OSCs) [1]. Such software products base their code on deliverables produced in so called software ecosystems, which span beyond the influence of a single organization [2]. Hereby, open source communities often provide innovative, user demand driven software, free of charge [3]. Implementing open source components into commercial applications has many advantages for software developers, as they can reduce development costs, improve an applications performance and add functionality without the need to invest into according in-house capabilities [4]. Development resources, freed up by open source integration, can be invested in strengthening a software products core capabilities and improving its market competitiveness [5].

For developing and maintaining a software product that depends on open source code, product managers must be able to assess the health of the communities their products depend on. A lack of understanding of a software component's vitality bears not quantifiable risks for the software firm, as well as for customers relying on the product in question. However, such an assessment can be particularly difficult, as open source communities are loosely structured and use collaboration methods that differ greatly from those in commercial software development [6]. Additionally, an open source communities expertise is not centrally structured, less tangible than with commercial products, and bears no obligations for formal technical support.

Currently, little work on the topic of open source vitality exists and no research known to the authors explicitly focuses on the needs of software product managers, evaluating open source for commercial applications. Wahyudin et al. [7] directly addresses open source community health. Based on an evaluation of scientific literature on OS communities and software project monitoring, the authors constructed a software community interaction model, focusing on the three core quality related perspectives of OSCs: The developer community, the user community and the software product. Consequently, the authors constructed two core health indicators that aggregated previously introduced performance variables: Developer contribution and bug service delay. Weiss [8] takes a different approach to assessing OSC popularity and proposes to look at web search engine results. Based on the assumption that a successful OS application will enjoy broad distribution, he proposes four measures of assessing a communities' popularity, based on the number of matching search results. The work of Izquierdo-Cortazar et al. [9] presents some of the more recent OSC vitality related publications. The authors statistically analyze OSC evolvabil-

*Corresponding author. Address: Department of Information and Computing Sciences, University of Utrecht, P.O. Box 80.089, 3508TB Utrecht, The Netherlands. Tel.: +31 (0)30 253 98 96.

Email addresses: `i.zaytsev@students.uu.nl` (Ivan Zaytsev), `s.jansen@cs.uu.nl` (Slinger Jansen)

ity and robustness of 1400 FLOSS communities. Hereby, the utilized approach divided the main research question into two sub-questions dealing with: Size and regeneration, as well as interactivity and workload adequacy. Opposite to the efforts of Izquierdo-Cortazar et al., Samoladas et al. [10] evaluate probabilities for FLOSS survival. By mining an open source database, funded by the European Union (FLOSSMetrics), the authors review time series data and attempt to predict the continuation of OS projects. Among all reviewed research , one of the most comprehensive approaches to general OSC vitality is the work of Subramaniam et al. [11]. The paper addressed OSC project success by means of a longitudinal data analysis of OS projects on SourceForge over a time period of 5 years. Based on a literature analysis, the authors constructed a model of open source success measures that was divided in developer interest in the project, user interest in the project and project activity. In addition, the model evaluated interrelations between success factors and introduced a segmentation into time dependent and time independent variables, such as e.g. developer interest or operating system language.

In order to fill the research gap, this article introduces a modular method for OSC vitality analysis, specifically designed to be adjustable to situational requirements of varying communities and project characteristics. The method is built around a pool of method fragments, each aimed at validated OSC vitality indicators and labelled with suitable selection rational. Furthermore, a case study at a large software company with extensive open source expertise was performed. The gained industry insights were used to expand the understanding of OSC vitality indicators, as well as for validation of the presented method.

The following section introduces the utilized research method, covering case study details, as well as a systematic literature review on the topic of OSC and open source related software ecosystem health. Lorem ipsum.....

2. Reserach Method

3. The OSC Health Analysis Method

Built on the developed understanding of OSC vitality, this section introduces a systematic approach for software product managers that enables them to evaluate the vitality of candidate open source communities. Hereby, the foundation for the following method is based on five expert interviews, the introduced systematic literature review, a study of OSC related practices at the case company, as well as a distilled list of OSC vitality indicators.

3.1. Method Fragment Pool

Derived from the domain of method engineering, this section introduces the concept of situational method design, as well as that of a method fragment pool, containing base elements for method assembly. While the general principles of situational method engineering [12] fully apply, this section focuses on a customized adoption towards the specific needs of Software Product Managers and their responsibilities in regard to OSC dependency management. Built upon the work of Harmsen, Brinkkemper and Oei [13], Figure 1 depicts a process flow aimed at creating a situational method for OSC vitality assessment. Furthermore, the diagram includes an iterative cycle, focused on the customization of method fragments towards a community's unique characteristics. The building blocks of a situational method are so called method fragments, which are stored within the method fragment pool. Each element within the pool focuses on assessing one specific OSC vitality aspect and can be either used by itself, or in combination with other fragments. Each fragment is linked to situational factors that make it best suited for assessing either a given community type or determining its applicability for a given Software Product. Fragment selection begins with an assessment of OSC characteristics, necessary for narrowing down the total number of candidate method fragments for the given case. Combined with the requirements and the resources of the respective Software Product, both form the selection rational for choosing suitable method fragments within the existing fragment pool. Following the selection process, the chosen fragments are being assembled into a situational method. Although a particular order is not a requirement, a logical structure or segmentation into fragment groups can contribute to the methods usability. Depending on the relative significance of community aspects, fragment importance can vary. Thus, looking at the overall quality of the software product, individual fragments can receive a greater weighted influence over less significant ones. After method assembly, the individual fragments and the entire method can undergo an iterative process, aimed at improving the accuracy of the assessed OSC vitality. Hereby, historic or present community data can be used to review fragment and method accuracy. If a given fragment is found to generate inaccurate or contradictory vitality data it can be adjusted, its result weighed less or it can be fully removed from the situational method. All community specific adjustments can then be updated and re-inserted into the fragment pool. Should another software product, with varying project characteristics, require assessing this particular OSC, the new and more accurate fragments can be implemented.

3.2. Fragment Categorization and Selection Rationale

The following sub-section introduces a number of criteria that can be used to select situation specific method fragments for a given OS community or software development project. The main reason for this approach is the intention to create a practical heuristic that can be quickly utilized for method assembly. The approach does not enforce the selection of specific fragments, based on the presented rationale, but merely suggests using it as an indicator for assessing fragment suitability. The three introduced criteria have been selected based on their simplicity for the use by software product managers and their universal applicability for varying types of OSCs. Herby, the first two indicators stem from scientific literature and focus on specific community characteristics. The third indicator is based on the authors estimate for required fragment resources and aims at a software product's development scope.

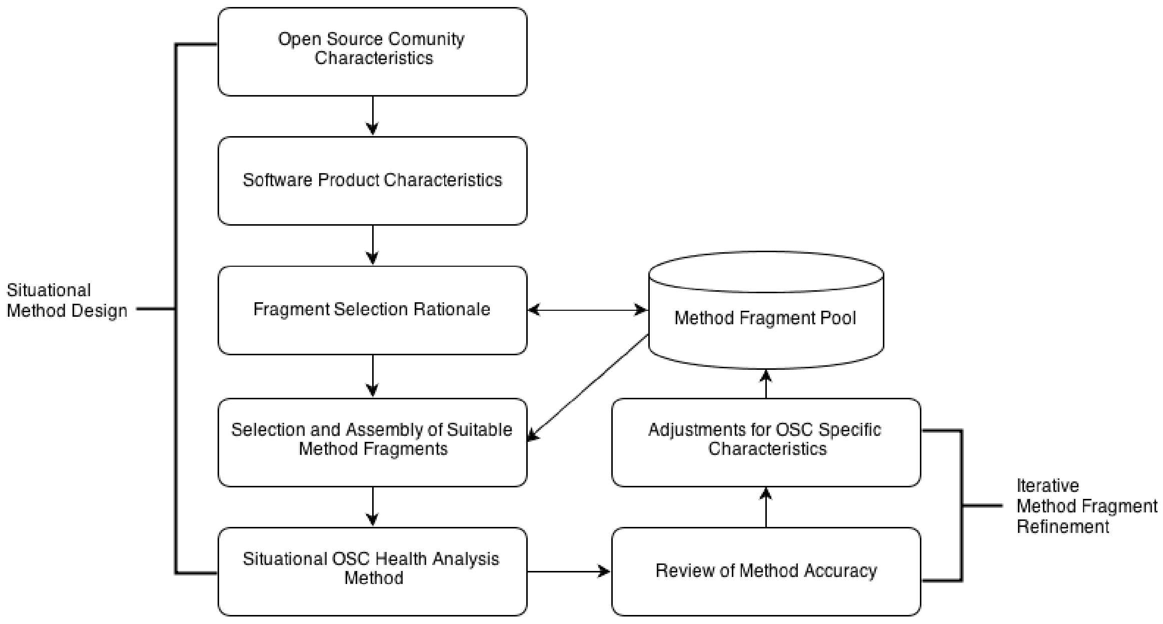

Open Source Comunity Characteristics
Software Product Characteristics
Fragment Selection Rationale
Selection and Assembly of Suitable Method Fragments
Situational OSC Health Analysis Method
Method Fragment Pool
Adjustments for OSC Specific Characteristics
Review of Method Accuracy
Situational Method Design
Iterative Method Fragment Refinement

Figure 1: Design Process: Situational OSC Health Analysis Method

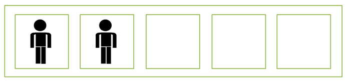


Figure 2: Fragment Selection Indicator - Community size

The most evident indicator for fragment selection is community size. Hereby the size can be measured by the number of OSC developers, the number of core developers or alternatively the total number of registered community members. While this indicator needs to be adjusted for each community's nature or characteristics, it can help to quickly exclude fragments that are only applicable to large or very small OSC types. Each method fragment is labelled with one or many community size indicators, suggesting the most suitable community sizes.

Community age is a direct indicator for the probability of its survival and was found to have a significant positive correlation with the maturity of it software development processes and its governance structure [10]. This in return makes it a suitable indicator for selecting applicable method fragment, as it can indicate which fragments are best suited for mature or young OSCs. Alike to the previous indicator, each fragment can have one or multiple labels describing matching OSC criteria. Looking di-

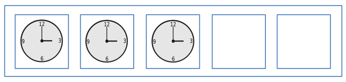


Figure 3: Fragment Selection Indicator - Community maturity

rectly at the perspective of software product managers, disposable time and financial resources can vary greatly with each managed software product. Thus, even if a method fragment for vitality assessment is promising, it yet may not be feasible to use. This can be the case if the given fragment exceeds available project resources or if the overall analysis would benefit from focusing the efforts on less resource intensive fragments. In contrast to the previous indicator, each fragment is labeled with only one cost indicator that expresses the authors' estimate of the necessary financial or time investment, in order to execute the according method fragment.

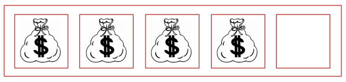
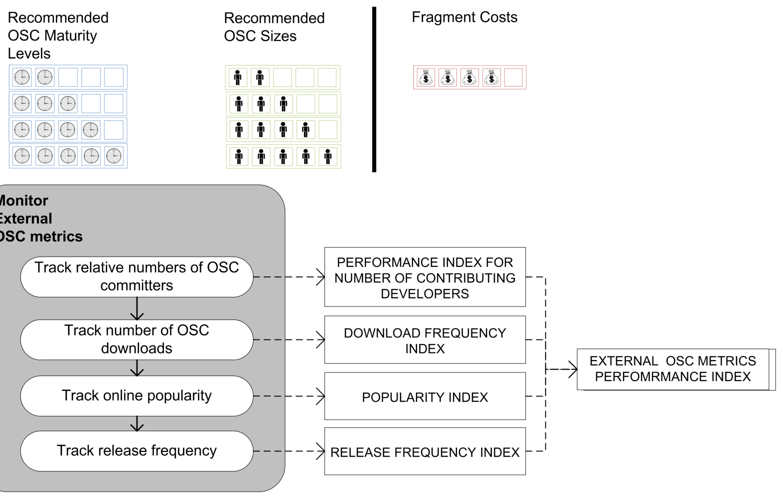


Figure 4: Fragment Selection Indicator - Fragment cost

3.3. Sample Fragment and Method Utilization

By combining the introduced selection rationale and a list of validated vitality indicators, this sub-section describes a representative OSC health method fragments that can be used by software product managers for OSC assessment. The complete method is comprised of ten method fragments, segmented into five fragment categories. Three categories are designed to group fragments that operationalize and process similar data types. The remaining two categories aim at establishing a common method base for standardized data extraction and the optimization of monitoring resource with other OSC stakeholders.

The sample fragment, depicted in Figure 5, aims at external OSC metrics and is most suitable for measuring outward facing OSC activities. This includes actions, such as extracting the number of OSC downloads over time, tracking online search popularity or monitoring release frequency. All activities produce individual results, as well as an aggregate index,



Recommended OSC Maturity Levels
Recommended OSC Sizes
Fragment Costs
Monitor External OSC metrics
Track relative numbers of OSC committers
Track number of OSC downloads
Track online popularity
Track release frequency
PERFORMANCE INDEX FOR NUMBER OF CONTRIBUTING DEVELOPERS
DOWNLOAD FREQUENCY INDEX
POPULARITY INDEX
RELEASE FREQUENCY INDEX
EXTERNAL OSC METRICS PERFOMRMANCE INDEX

Figure 5: OSC Health Method Fragment: Monitor external metrics

representing the overall metrics performance. The fragment is suitable for a broad range of OSCs with varying degrees of maturity and size. However, due to the necessity to establish and track changes to the indices, the fragment is relatively expensive and has the second highest cost rating. In order to present a practical example of the methods application, a representative OSC was selected for data analysis. The main reason for choosing a single OSC was to introduce a set of steps that can be taken in order to apply the introduced method fragments to a realistic assessment scenario. Furthermore, the aim was to present an analysis of a broadly known OSC and thus make the example as relatable as possible for project managers, familiarizing themselves with the new method.

OSC selection was performed based on the popularity index of one of the most widespread public source code repositories, namely Github. The platform is built around the distributed revision control system Git that provides an additional source of data. Due to Git's fully distributed architecture, revision data can be directly extracted from every cloned copy of a repository. Among the highest rated community projects on Github, the jQuery JavaScript library was selected for analysis. Although ranked third in the index (8. May 2013), the first two projects Twitter Bootstrap and Node.js were excluded due to either their direct ties to a commercial entity or the scope of available data and community maturity.The fragment's analysis is based on community data available on Github, the jQery Git repository, Ohloh (a public directory of open source software owned by Black Duck Software, Inc.), as well as community pages, blog entries and similar documentation of the jQery project. The jQery Git repository was mined with the open source tool GitStats[1].

As suggested by the method fragment, all measured indices are aggregated into one external performance index. For this purpose, each metric has been first recorded with the according activity for each month between Aug. 07 and Apr. 13. Due to the different measuring units, the performance was adjusted to changes in percent, relative to a standardized base value. The index for the number of active contributors per month is operationalized by changes in percent, compared to an average of 8 contributors per month. This value is based on the actual project average for its entire lifespan and has been rounded to an integer value. Release frequency is recorded in relative frequency compared to a release frequency of 2 releases per year or 0.16 releases per month. As online popularity is already measured in percent values, relative to an established base value, the index has been included in the analysis as is. Although the most recent Github API does offer the opportunity to extract the overall number of downloads, this data attribute is not tracked over time and does not take downloads from other mirrors into account. Thus, this vitality indicator will not be included in this analysis.

By combining the external performance indicators with available data for commit frequency, an aggregate view can be generated that incorporates all long term community activity into a single performance index. Figure 6 depicts a unified index that combines all measured external vitality indicators with OSC commit frequency. Hereby, the commit frequency is expressed in a relative percent value based on an average commit frequency of 60 commits per month.

It is noteworthy that when evaluating the graph, measurement consistency is most important while exact performance values lack significance. All index activity values are measured in relation to an established base value and could indicate stronger or weaker growth if the base is modified.

3.4. Method Assembly and Calibration

Depending on fragment selection, the final situational method can contain any combination of index or scale based elements, as well as binary results. Additionally, two fragments with the purpose of either correcting activity indexes for cyclical trends or for optimizing OSC monitoring efforts can be included.

[1]http://gitstats.sourceforge.net

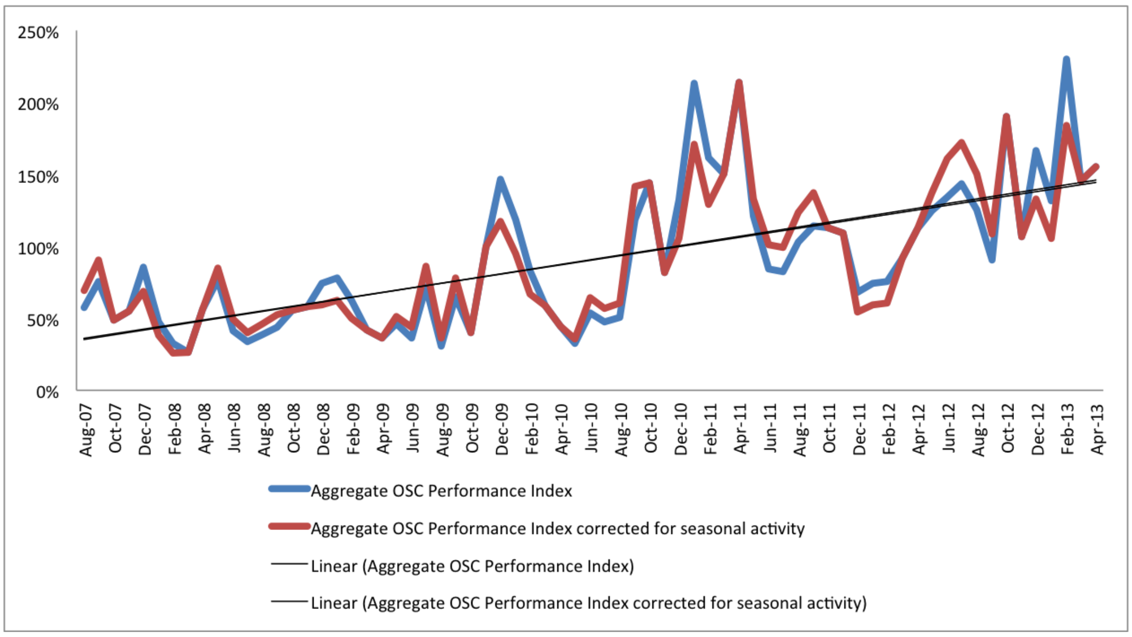


Figure 6: Aggregate OSC Performance Index: jQuery Core

The only fragment required for method assembly is the method base. Consequently, any number or combination of fragments can be utilized within the situational method. Based on the chosen selection rationale, software product managers can group fragments and are free to either aggregate method results or compare them on an individual basis.

Specifically looking at index driven methods and the produced performance indices, they can be aggregated to create a unified OSC performance index. Next to the overall OSC health, individual vitality indicators can be monitored separately in order to identify trends within vitality areas. Furthermore, the approach supports index adjustment by correcting the individual weight of vitality indices according to their importance for a given OSC. Thus, a situational method cannot just be tuned by means of fragment selection but also by means of adjusting fragments to local OSC realities.

Method artifacts generated by non-index fragments lack a distinct scale; however they can be used to register changes in vitality. By individually tracking variations in the likert ratings, each method fragment can be used as an indicator for changes in OSC health. The same applies to the specific case of monitoring software development practices. Despite the binary nature of the results, each can be compared to past practices of the OSC. Thus, added practices can be seen as an improvement to community health or code quality, while the abandonment of practices can be seen as a decline in community performance.

4. Discussion and Conclusion

References

[1] S. Jansen, A. Finkelstein, S. Brinkkemper, A sense of community: A research agenda for software ecosystems, 31st International Conference on Software Engineering (ICSE 2009) (2009) 2–5.

[2] J. Bosch, From software product lines to software ecosystems, SPLC '09 Proceedings of the 13th International Software Product Line Conference (2009) 1–10.

[3] E. Von Hippel, Learning from open-source software, MIT Sloan management review 42 (2001) 82–86.

[4] J. E. Bessen, Open Source Software: Free Provision Of Complex Public Goods, Social Science Research Network (2001) 57–81.

[5] R. E. Hawkins, The economics of open source software for a competitive firm, NETNOMICS: Economic Research and Electronic Networking 6 (2004) 103–117.

[6] K. Crowston, J. Howison, The social structure of free and open source software development, First Monday 10 (2005) 2–7.

[7] D. Wahyudin, K. Mustofa, A. Schatten, S. Biffl, A. M. Tjoa, Monitoring the "health" status of open source web-engineering projects, International Journal of Web Information Systems - IJWIS 3 (2007) 116–139.

[8] D. Weiss, Measuring success of open source projects using web search engines, Technology (2005) 93–99 TS – BeitraginSammelband.

[9] D. Izquierdo-Cortazar, J. González-Barahona, G. Robles, J. Deprez, V. Auvray, FLOSS Communities: Analyzing Evolvability and Robustness from an Industrial Perspective, Open Source Software New Horizons 319 (2010) 336–341.

[10] I. Samoladas, L. Angelis, I. Stamelos, Survival analysis on the duration of open source projects, Information and Software Technology 52 (2010) 902–922.

[11] C. Subramaniam, R. Sen, M. L. Nelson, Determinants of open source software project success: A longitudinal study, Decision Support Systems 46 (2009) 576–585.

[12] I. Van De Weerd, S. Brinkkemper, Meta-modeling for situational analysis and design methods, Handbook of Research on Modern Systems Analysis and Design Technologies and Applications (2008) 35.

[13] F. Harmsen, S. Brinkkemper, H. Oei, Situational Method Engineering for Information System Project Approaches, in: A. A. Verrijn-Stuart, W. T. Olle (Eds.), Methods and Associated Tools for the Information Systems Life Cycle, September, Elsevier Science Inc., 1994, pp. 169–194.