OSSARA: Abandonment Risk Assessment for Embedded Open Source Components

Xiaozhou Li
Tampere University

Sergio Moreschini
Tampere University

Fabiano Pecorelli
Tampere University

Davide Taibi
Tampere University

***Abstract*—**
Accompanied by the upsurging adoption of open source software (OSS) in a wide range of complex software-intensive systems, practitioners often raise multiple concerns about their maintainability and sustainability. A system that embeds unmaintained components is fragile towards various risks which could result in dire consequences. To address such concerns, this paper introduces the OSS Abandonment Risk Assessment (OSSARA) model, which evaluates the abandonment risk by synthesizing the prediction of potential abandonment of the embedded OSS components.

SOFTWARE needs to be continuously updated and maintained to keep being useful [1]. This is particularly true for open source software (OSS) components and libraries, which are more and more often integrated into large and complex systems.

For companies developing long-term projects, all the embedded OSS components should guarantee a high life expectancy and continue being maintained as long as the system embedding them is on duty. Embedding abandoned OSSs in a critical system could induce severe risks. For example, new security vulnerabilities could be exploited; bugs and issues could never be resolved; functions could become obsolete and inadequate for new environments. Metaphorically, systems embedding abandoned OSSs are like vehicles with rusted gears or human bodies with malignant tumors. Indeed, the abandonment of OSS components might produce a ”domino effect” in the projects embedding them that shall result in the inoperativeness of all the systems relying on them. The importance of such a statement is in the fact that even if a single embedded software is not available a whole project can be compromised.

In this respect, we were recently asked by a local branch of a global company, working in different domains, and with over 200K employees in over 150 countries, to devise a methodology aimed at identifying those components embedded in their software products that are most likely to be abandoned shortly. To meet their requirements, we designed the **OSS Abandonment Risk Assessment (OSSARA)** model, which we present

in this paper. This model aims to assess the abandonment risk of a software system based on the abandonment prediction for each embedded OSS component and the criticality that each component represents for the system. With the OSSARA model, the practitioners shall be intuitively aware of the overall abandonment risk of the system and the predictive assessment of each component's abandonment. Therefore, the practitioners can monitor the system's risk level and proactively choose either to contribute to the OSS component maintenance and evolution or to plan for the substitutions.

Related Work

Over the last decade, researchers have been giving great attention to software sustainability.

Samoladas et al. [2] successfully exploited survival analysis methods to predict the survivability of software projects.

Businge et al. [3] analyzed the survivability of 1,447 versions of 467 Eclipse third-party plugins and classified them into two categories: the ones relying on stable dependencies and those on at least one potentially unstable dependency. They observed that plugins that only use stable dependencies are more prone to maintain a higher source compatibility rate over time.

Coelho et al. [4] leveraged machine learning to build a model able to identify unmaintained GitHub projects based on a set of 13 process metrics achieving promising results. Afterward, they presented an extended version of the work [5] defining a metric to indicate how risky it would be depending on a given GitHub project.

Valiev et al. [6] assessed open source Python projects' sustainability based on ecosystem-level factors, i.e., factors describing inter-dependencies between packages. They calculated sustainability by the mean of dormancy, i.e., the period of inactivity for a project repository. Results indicated that the number of connections, as well as the position within the dependency network, are significant factors affecting the projects' sustainability.

Later on, Mujahid et al. [7] proposed a scalable approach that relies on the package centrality in the ecosystem to identify packages in decline. Results of an evaluation conducted on the npm ecosystem have shown very good prediction capabilities, thus indicating centrality as a very important factor for predicting project abandonment.

In our previous work [8], we investigated approaches to automate the evaluation of information from OSS Projects, however, we did not propose an assessment and risk model.

In contrast to the related literature, we are proposing a method to calculate the abandonment risk based on the risk that embedded components of a system will be abandoned themselves. Moreover, thanks to the support we have received from our case company, our method is completely suitable for real industrial applications.

Software Composition using OSS

Software composition via the adoption of components off-the-shelf (COTS) has long been considered as an effective practice for software implementation [9]. Despite the disadvantages of COTS in terms of their uneven performance, lack of evolution control, and lack of inter-operating capabilities, their adoption benefits the practitioners by simply preventing them from "reinventing the wheel." OSS can be considered as COTS since most of the embedded components, e.g., libraries or plug-ins, are usually integrated as is.

The main advantages of OSS components are the open licenses, which usually enable access to the source code, and eventually to make extensions. Moreover, OSS is often accessible without paying a license fee, thus reducing adoption costs.

When developing a new software project, the most common practice is to integrate several components and combine them by writing custom code. The portion of the custom code is usually minimal compared to the total size of such components. The development of all the components as custom software might require a large amount of effort, not only for the development itself but also for the maintenance side.

Creating a system consisting of several components also introduces risks since the maintenance of each OSS component is usually delegated to the OSS community. However, there might be cases where the community does not keep on maintaining them; and therefore companies integrating these unmaintained OSS components need to find alternatives, either deciding to maintain the components themselves or replacing them with alternatives.

OSSARA Model

Herein, we propose the OSSARA model to assess the abandonment risk of a system based on its embedded OSS components. Given a time, the abandonment risk is calculated based on (i) the likelihood of each component to be abandoned in the considered period, and (ii) the weight that each component has for the main system, following the classic risk assessment notion $Risk = Prob(Loss) \times Size(Loss)$ [10]

Figure 1 depicts the OSSARA process. Starting from a software system that embeds several OSS components (14 in the provided example), we first calculate for each component the abandonment probability in the given time and the weight (abandonment probability and weight are represented by colors and box sizes respectively). Then, we combine these two pieces of information to calculate the risk that the main system will be abandoned within the considered period.

More formally, the overall abandonment risk R_a for a system ($R_a \in [0, 1]$) that integrates k OSS components is calculated as follows:

$$R_a = \sum_{m=1}^{k} w(O_m) * r(O_m).$$

where $w(O_m) \in [0, 1]$ represents the weight of the OSS component O_m and $r(O_m) \in [0, 1]$ the risk that O_m will be abandoned. The weight of a component $w(O_m)$ can be quantified by counting the number of invocations (e.g. number of imports in the code).

Predicting OSS Abandonment

Identifying inactive or abandoned OSSs could be easily performed by directly checking for the presence of specific tags on SourceForge[1]. However, noticing such labeling would be too late for a company to find proper alternatives. Thus, it is necessary to find a way to foresee the potential abandonment proactively.

Predicting the abandonment risk for a software component is a multi-concern assurance problem since it could depend on several aspects, such as low performance, scarce maintainability, etc. Commonly, an OSS is considered abandoned just based on the number of commits performed on the system repository in a given time interval [2], [11], [12]. Therefore, one could trivially think to use this information as a sole predictor to foresee the abandonment of an OSS component; i.e., if the number of commits on a certain project repository goes below a pre-defined threshold in a certain period, then the component is considered abandoned. However, determining either the threshold or the period length shall vary amongst different practitioners.

Our case company considers an OSS project as abandoned if it does not have any releases or commits within the last six months, which is comparatively a stricter threshold than that suggested by Khondhu et al. [11]. Furthermore, it is also possible that when an OSS community does not focus on committing, the contributors are still active in handling pull requests or discussing the relevant issues. In this case, the suggestion from our case company is that the measures regarding committing (e.g., the daily number of commits), communication (e.g., the daily issue comments), and issue handling activities (e.g., daily closed pull requests) shall all be taken into account. For the above reasons, we propose to apply supervised techniques to predict the abandonment likelihood for a given OSS component based on the aforementioned key activities (e.g., commits, issues, pull requests, etc.). This will overcome the problem of subjective thresholds: rather than relying on pre-defined strict thresholds, these techniques can adapt the prediction to the component under analysis.

In detail, to predict the abandonment of an OSS component, we propose the following four-step pipeline:

- **Step 1. Data Crawling:** We gather data from all the 125486232 GitHub projects using the GHTorrent dataset[2].The selected metrics are: #commits, #commit comments, #unique committers, #issues, #issue comments, #watchers, #Open and closed Pull-Requests.
- **Step 2. Data Preprocessing:** We created training data for each OSS project that fulfill the criteria specified by our case company, labeling as *active* projects including 1) Commits $>$ 2000, 2) Days of activity (from the created day to the last commit day) $>$ 1000, 3) At

[1] https://sourceforge.net

[2] The GHTorrent Project https://ghtorrent.org

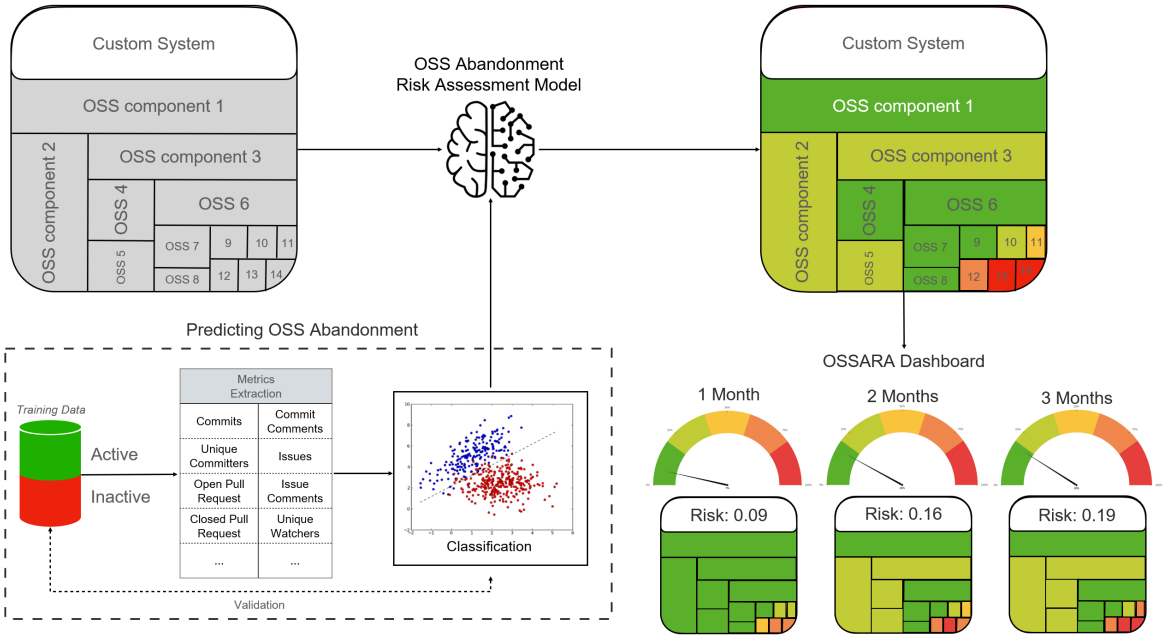

Figure 1. Overview of the OSSARA process.

least 1 commit in the last 6 months, 4) Days with 0 commits $<=$ 50% of days of activity. Furthermore, the labeled training data shall be prepared based on the target prediction period (e.g. one, two, or three months) with the proper dimension of the data determined towards best prediction accuracy.

- **Step 3. Prediction:** Using the labeled and preprocessed data, we train the classifiers of the best performance for the target prediction periods. With the data of the target OSS component as input, the classifier will predict if the component is active or abandoned in the target period. The accuracy of the classifier is used as the probability of the OSS being active or abandoned, as it indicates the probability the prediction is correct.

Validation

To validate the proposed methodology, we conduct a preliminary evaluation on 12208 OSS projects[3], which contain at least 1000 commits from at least five unique contributors and are watched by at least 100 users. Such selection criteria assure the popularity and longevity of the candidates. The dataset is extracted from GHTorrent dataset (till the 2019-06-01 dump) and labeled according to the aforementioned guidelines (see Step 2). Among the four popular classification algorithms we selected, i.e., Decision Tree, Support Vector Machine, Logistic Regression, and Naive Bayes, we find Logistic Regression has the highest accuracy with the dataset. We also apply a 10-fold cross-validation strategy to assess the prediction capabilities of the model. Results of such a validation report an F1 score of $\approx$ 0.86 ($\pm$0.01 estimated error) with the Matthews Correlation Coefficient (MCC) being 0.73; hence, we can conclude that the proposed methodology is reliable enough.

[3]The dataset is shared at https://doi.org/10.6084/m9.figshare.16944001.v1

Working Example

This section presents a working example of the proposed method. For matters related to the non-disclosure agreement with our case company, we can not provide details about a real industrial application of our technique. However, to demonstrate the OSSARA model at work, we apply the approach to an open-source software project case. Herein, we take Keras as an example of software developed in-house that needs to integrate various OSS components. Keras is an OSS project providing a Python-written deep learning API for TensorFlow libraries. The release version

adopted for our analysis is the Release 2.7.0 RC1[4] accessed on 2021-10-26 from Github.

We conduct our analysis only focusing on the 536 Python files from the repository and detect the OSS components (i.e., packages) imported within each file. Furthermore, to ease the computation as well as the explanation of the results, we only consider the top 20 packages most frequently imported in Keras. Herein, we consider only the five most commonly imported OSS components in Keras, namely, *TensorFlow*, *CPython*, *Numpy*, *abseil-py*, and *h5py*. Packages, e.g., *re*, *random*, *collections*, etc. all belong to the *Cpython* component and will be considered together. The weight of each component is thus calculated by the percentage of files importing it.

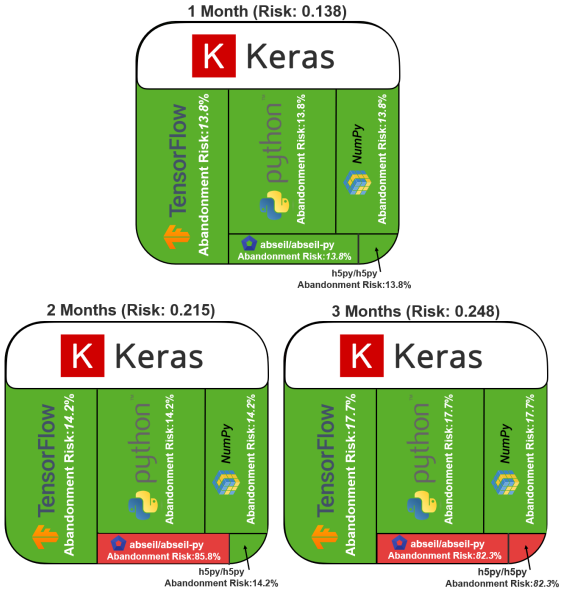

Figure 2. Abandonment Risk Assessment for Keras

We conduct our analysis using three different time frames to predict the abandonment risk of embedded OSS components in one, two, and three months. The same approach can also be applied to longer time-frames. To simplify the process, we quantify the weight of each imported Python package counting the number of imports for such package over all the project files. Figure 2 summarizes the results obtained in the three considered time frames.

The results show that within one month all components are safe from abandonment. When considering a two-month time frame, the *abseil-py* package appears to have a high risk of being abandoned (i.e., 85.8%). Finally, as for the three-month analysis, the risk that the *h5py* package will be abandoned arises. The three key components, namely *TensorFlow*, *Cpython*, and *Numpy*, remain active throughout the period. Based on our calculation, the overall abandonment risk for Keras grows from 0.138 in one month to 0.215 in two months and 0.248 in three months. From the repository history of *abseil-py* and *h5py*, we can observe since 2020 both projects have had a very low committing and issue-handling rate from a small group of contributors, which legitimizes the existence of such risks.

This demonstration example shows that the OSSARA model can provide an intuitive prediction of the abandonment risk of software systems that embed OSS components. However, this oversimplified example aims only to explain how our method works when the probability-based conclusion may not reflect reality. Please note that in this simplified example we did not consider hierarchical relations. For example, the selected 20 packages might, in turn, use other components that might have a high abandonment risk. Meanwhile, though the compliance to real industrial needs from the case company is the main strength of the model, the generalizability can be limited. Furthermore, the potential application of the risk prediction towards component replacement and integration requires the support of assessment methods on software engineering decision making [13]. Last, other metrics might have different prediction power for the abandonment risk.

Conclusion

Integrating abandoned OSSs in software-intensive systems is hazardous and could result in severe consequences, which evokes the concerns of practitioners. Especially when the functions from abandoned components are integrated into the highly critical modules, the consequences caused by abandoned OSSs that are lacking maintenance can be unbearable. To foresee such risks to raise awareness, we propose the OSSARA model to provide an assessment and prediction pipeline detecting such risks to the integrated OSS components. The model also provides potentials for continuous adaptation and

[4] https://github.com/keras-team/keras/releases/tag/v2.7.0-rc1

customization by which the practitioners can conduct continuously optimized prediction via up-to-date OSS activity data with selectively effective and even customized algorithms. The model has been positively acknowledged by our case company which is currently adopting and integrating it into their CI/CD pipeline, to highlight the potential abandonment risk of embedded OSS components and to take actions proactively by notifying developers.

Future works shall be conducted towards the exploration of other analysis techniques, the inclusion of other metrics in the prediction model, the integration of the other types of OSS risk assessment, e.g., security assessment, license compliance assessment, etc., the dependency and hierarchy analysis, and towards implementation and potential industrialization.

REFERENCES

1. M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.
2. I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Information and Software Technology*, vol. 52, no. 9, pp. 902–922, 2010.
3. J. Businge, A. Serebrenik, and M. van den Brand, "Survival of eclipse third-party plug-ins," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 368–377.
4. J. Coelho, M. T. Valente, L. L. Silva, and E. Shihab, "Identifying unmaintained projects in github," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.
5. J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this github project maintained? measuring the level of maintenance activity of open-source projects," *Information and Software Technology*, vol. 122, p. 106274, 2020.
6. M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 644–655.
7. S. Mujahid, D. E. Costa, R. Abdalkareem, E. Shihab, M. A. Saied, and B. Adams, "Towards using package centrality trend to identify packages in decline," *arXiv preprint arXiv:2107.10168*, 2021.
8. X. Li, S. Moreschini, Z. Zhang, and D. Taibi, "Exploring factors and metrics to select open source software components for integration: An empirical study," *Journal of Systems and Software*, vol. 188, p. 111255, 2022.
9. B. Boehm and C. Abts, "Cots integration: Plug and pray?" *Computer*, vol. 32, no. 1, pp. 135–138, 1999.
10. B. Boehm, "Software risk management," in *European Software Engineering Conference*. Springer, 1989, pp. 1–19.
11. J. Khondhu, A. Capiluppi, and K.-J. Stol, "Is it all lost? a study of inactive open source projects," in *IFIP international conference on open source systems*. Springer, 2013, pp. 61–79.
12. J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 2017, pp. 186–196.
13. R. A. Ribeiro, A. M. Moreira, P. Van den Broek, and A. Pimentel, "Hybrid assessment method for software engineering decisions," *Decision Support Systems*, vol. 51, no. 1, pp. 208–219, 2011.

Xiaozhou Li is a Ph.D. candidate in the Faculty of Information Technology and Communication Sciences, Tampere University, Finland. He is a researcher at Cloud Software Evolution and Assessment (CloudSEA) research group. His research interests include open-source software quality, software maintenance and evolution, user review opinion mining, data-driven empirical software engineering, computational game studies, gamification design, etc.

Sergio Moreschini is a Ph.D. candidate in the Faculty of Information Technology and Commu-

nication Sciences, Tampere University, Finland. He is a researcher at Cloud Software Evolution and Assessment (CloudSEA) research group. His main research interest focuses on extended light field reconstruction for continuous parallax content. He also contributes actively to the domains of empirical software engineering, open-source software quality, data-driven software engineering, etc.

Fabiano Pecorelli is a researcher at Cloud Software Evolution and Assessment (CloudSEA) research group at Tampere University. He received a bachelor's and master's degree in computer science from the University of Salerno, Italy. He is about to defend his Ph.D. dissertation at the Department of Computer Science, University of Salerno, under the supervision of Prof. Andrea De Lucia. His research interests include software code and test quality, predictive analytics, mining software repositories, software maintenance and evolution, and empirical software engineering. To this aim, he applies several techniques such as machine learning, search-based algorithms, and mining of software repositories. He serves and had served as a referee for various international journals in the field of software engineering (e.g., TOSEM, EMSE, JSS).

Davide Taibi is Associate Professor at the Tampere University (Finland) where he head the Cloud Software Evolution and Assessment (CloudSEA) research group. His research is mainly focused on Empirical Software Engineering applied to cloud-native systems, with a special focus on the migration from monolithic to cloud-native applications. He is investigating processes, and techniques for developing Cloud Native applications, identifying cloud-native specific patterns and anti-patterns. He is member of the International Software Engineering Network (ISERN) from 2018. Before moving to Finland, he has been Assistant Professor at the Free University of Bozen/Bolzano (2015-2017), post-doctoral research fellow at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering - IESE (2013-2014) and research fellow at the University of Insubria (2007-2011).