

OSSARA: Abandonment Risk Assessment for Embedded Open Source Components

Xiaozhou Li, Sergio Moreschini, Fabiano Pecorelli, and
Davide Taibi, Tampere University

// Systems with unmaintained embedded open source software (OSS) components are vulnerable to severe risks. This article introduces the OSS Abandonment Risk Assessment model to help companies avoid potentially dire consequences. //



SOFTWARE NEEDS TO be continuously updated and maintained to continue 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 embedded OSS components should guarantee lengthy life expectancies and be maintained as long as systems are in service. Embedding abandoned OSS in critical systems could expose companies to severe risks. For example, new security vulnerabilities could be exploited, bugs and issues might never be resolved, and 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” that results in the inoperability of full systems. The importance of such a statement is in the fact that even if a single embedded software component is unavailable, a whole project can be compromised.

In this respect, we were recently asked by a local branch of a global company, which operates in different domains and with more than 200,000 employees in 150-plus countries, to devise a methodology aimed at identifying components embedded in its software products that were the most likely to be abandoned soon. To meet the requirements, we designed the OSS Abandonment Risk Assessment (OSSARA) model, which we present in this article. The model aims to assess the abandonment risk of a software system through prediction for every embedded OSS component and the criticality that each component represents. With OSSARA, practitioners can monitor a system’s risk level and choose to maintain or replace OSS components.

Related Work

During the past decade, researchers have been paying great attention

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/deed.ast>.

Digital Object Identifier 10.1109/MS.2022.3163011
Date of current version: 20 June 2022

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 plug-ins and classified them into two categories: those relying on stable dependencies and those with at least one potentially unstable dependency. They observed that plug-ins that use only stable dependencies are likelier to maintain a higher source compatibility rate through time. Coelho et al.⁴ leveraged machine learning to build a model 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 to depend on a given GitHub project.

Valiev et al.⁶ assessed open source Python projects' sustainability based on ecosystem-level factors, i.e., those describing interdependencies among packages. They calculated sustainability by the mean of dormancy, i.e., the period of inactivity for a project repository. The results indicated that the number of connections as well as the dependency network position are significant factors affecting the projects' sustainability. Later, Mujahid et al.⁷ proposed a scalable approach that relies on the package centrality in an ecosystem to identify packages in decline. The results of an evaluation conducted on the Node Package Manager ecosystem showed strong prediction capabilities, thus indicating centrality as an important factor for forecasting project abandonment.

In previous work,⁸ we investigated approaches to automate the evaluation of information from OSS projects, although we did not propose an assessment and risk model. In

contrast to the related literature, we are introducing a method to calculate abandonment risk based on the probability of embedded system components losing maintenance support. Moreover, thanks to the assistance we received from our case company, our method is suitable for real industrial applications.

Software Composition Using OSS

Software composition via the adoption of components off the shelf (COTS) has long been considered an effective practice.⁹ Despite the disadvantages of COTS in terms of uneven performance, a lack of evolution control, and insufficient interoperating capabilities, using the components enables practitioners to avoid “reinventing the wheel.” OSS can be viewed as COTS since most of the embedded components, e.g., libraries and plug-ins, are usually integrated as is. The main advantages of OSS components are the open licenses, which usually permit access to the source code and, eventually, making extensions. Moreover, OSS is often accessible without paying license fees, thus reducing adoption costs.

When developing a software project, the most common practice is to integrate several components and combine them by writing custom code. The amount of custom code is usually minimal compared to the size of the components. Developing all components as custom software might require significant effort, not only for the process itself but also for maintenance. However, creating a system consisting of several OSS components introduces risks since the maintenance of each one is usually delegated to the developer community. There may be cases where the community does not continue the upkeep. Companies with integrated unmaintained OSS

components need to find alternatives, either deciding to maintain the components themselves or replacing them.

OSSARA

We propose OSSARA to assess a system's abandonment risk on the basis of embedded OSS components. The abandonment risk is calculated based on 1) the likelihood of each component losing maintenance support during a certain period and 2) the importance 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 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.

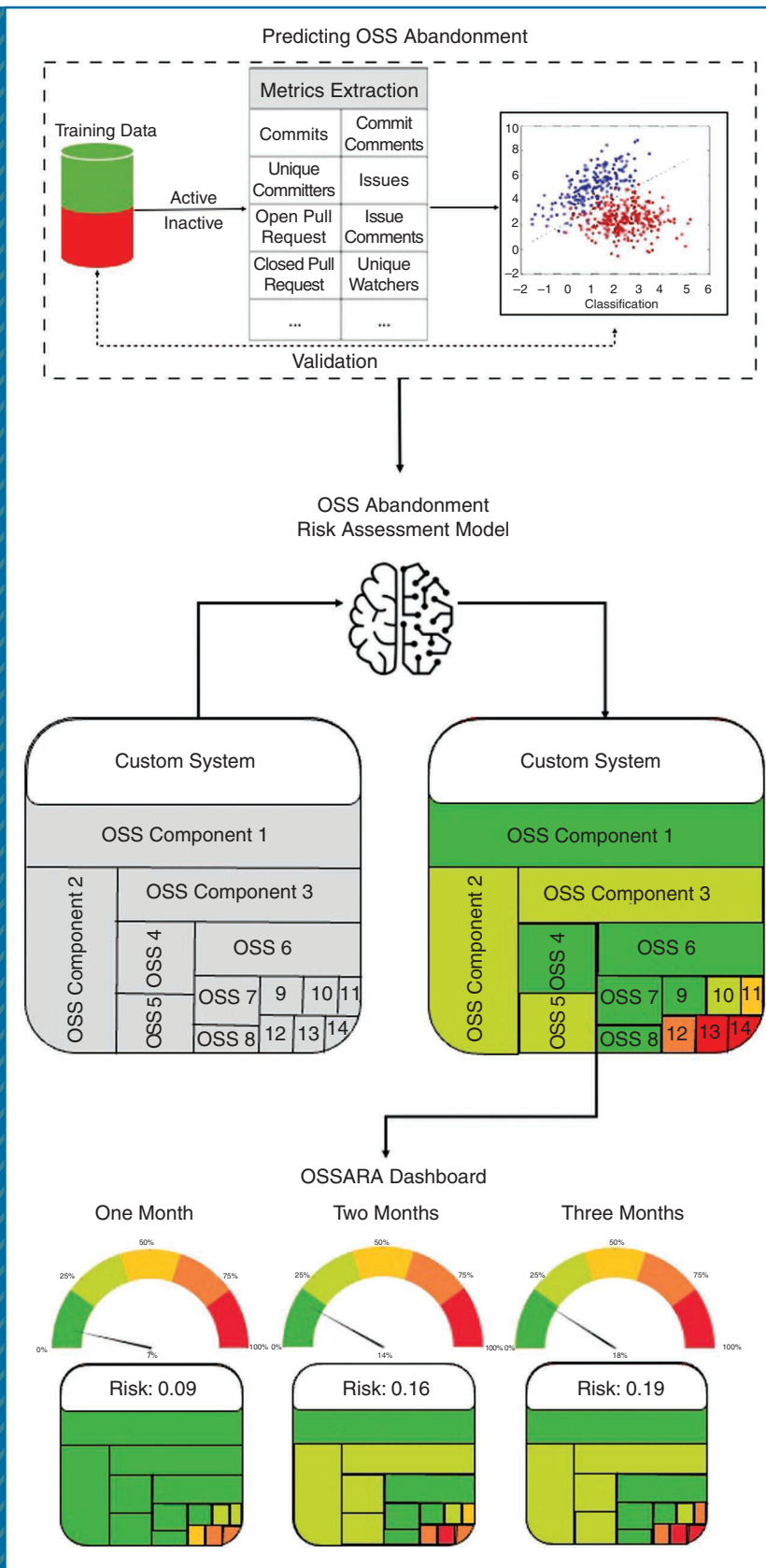
More formally, the overall abandonment risk R_a for a system ($R_a \in [0,1]$) that integrates k OSS components is calculated as follows:

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

where $w(O_m) \in [0,1]$ represents the weight of the OSS component O_m and $r(O_m) \in [0,1]$ conveys 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., the number of imports in the code).

Predicting OSS Abandonment

Identifying inactive and abandoned OSS could be easily performed by



directly checking for the presence of specific tags on SourceForge (<https://sourceforge.net>). However, by then, it would be too late for a company to find proper alternatives. Thus, it is necessary to find a way to foresee potential abandonment.

Predicting the abandonment risk for a software component is a multiconcern assurance problem since it could depend on several aspects, such as poor performance, insufficient maintainability, and so on. Commonly, an OSS component is considered abandoned based only 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 abandonment; i.e., if the number of commits on a certain project repository falls below a predefined threshold within a certain period, the component is considered abandoned. However, the process for determining the threshold or period length will vary among practitioners.

Our case company considers an OSS project abandoned if the software has not had any releases or commits within the previous six months, which is a comparatively 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 remain active in handling pull requests and discussing relevant issues. In this case, the suggestion from our case company is that measures regarding committing (e.g., the daily number of commits), communication (e.g., the daily issue comments), and issue handling activities (e.g., the daily closed pull requests) be taken into account. For these reasons, we propose to apply supervised techniques

FIGURE 1. The OSSARA process.

to predict the abandonment likelihood for an OSS component on the basis of key activities (e.g., commits, issues, pull requests, and so on). This would overcome the problem of subjective thresholds: rather than relying on predefined thresholds, prediction can be adapted to a component.

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

- *Step 1—Data crawling:* We gather data from all 125,486,232 GitHub projects by using the GHTorrent data set (<https://ghtorrent.org>). The selected metrics consist of the number of commits, commit comments, unique committers, issues, issue comments, watchers, and open and closed pull requests.
- *Step 2—Data preprocessing:* We create training data for each OSS project that fulfill the criteria specified by our case company, labeling projects active on the basis of 1) having more than 2,000 commits, 2) having more than 1,000 days of activity (from the day of creation to the final commit day), 3) having at least one commit in the past six months, and 4) having days with zero commits equal less than 50% of the days of activity. Furthermore, labeled training data are prepared based on the target prediction period (e.g., one, two, or three months), with their dimensions set to provide the best accuracy.
- *Step 3—Prediction:* Using the labeled and preprocessed data, we train the classifiers with the best performance for the target prediction periods. With the target OSS component data as input, the classifier predicts

whether a component is active or abandoned. Its accuracy is used as the probability of the OSS being active or abandoned, as it indicates the likelihood that the prediction is correct.

Validation

To validate the proposed methodology, we conduct a preliminary evaluation on 12,208 OSS projects that contain at least 1,000 commits from at least five unique contributors and are watched by at least 100 users (the data set is shared at <https://doi.org/10.6084/m9.figshare.16944001.v1>). Such criteria ensure the popularity and longevity of the candidates. The data set is extracted from GHTorrent (until the 1 June 2019 dump) and labeled according to the preceding guidelines (see step 2). Among the four classification algorithms we selected, i.e., decision tree, support vector machine, logistic regression, and naive Bayes, we find that logistic regression has the highest accuracy with the data set. We also apply a 10-fold cross-validation strategy to assess the prediction capabilities of the model. The results of the validation indicate an F1 score of ≈ 0.86 (a ± 0.01 estimated error), with the Matthews correlation coefficient being 0.73; hence, we conclude that the proposed methodology is reliable enough.

Working Example

This section presents a working example of the proposed method. Due to a nondisclosure agreement with our case company, we cannot provide details about the real industrial application of our technique. However, to demonstrate OSSARA at work, we apply it to an open source software project. 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 application programming interface for TensorFlow libraries. For our analysis, we chose release 2.7.0 RC1 (<https://github.com/keras-team/keras/releases/tag/v2.7.0-rc1>), accessed on 26 October 2021.


We conduct our analysis by focusing only 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 consider the 20 packages most frequently imported in Keras. We examine only the five most commonly imported OSS components in Keras, namely, TensorFlow, CPython, Numpy, abseil-py, and h5py. The packages, e.g., re, random, collections, and so on, belong to the CPython component and will be considered together. The weight of each component is calculated by the percentage of files importing it.

We conduct our analysis using three time frames to predict the abandonment risk of embedded OSS components in one, two, and three months. The same approach can be applied to longer periods. To simplify the process, we quantify the weight of each imported Python package by counting the number of imports across all the project files. Figure 2 summarizes the results. They show that for 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, for the three-month analysis, the risk that the h5py package

will be abandoned rises. The three key components, namely, TensorFlow, Cpython, and Numpy, remain active throughout. 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 observe that since 2020, both projects have had very low committing and issue handling rates from a small group of contributors, which legitimizes the risks assessment.

This example shows that OSSARA predicts the abandonment risk to software systems that embed OSS components. However, this oversimplified

demonstration aims only to explain how our method works when probability-based conclusions might not reflect reality. Please note that we did not consider hierarchical relations. For example, the 20 selected packages might use other components that have a high abandonment risk. Meanwhile, although the model meets real industrial needs—its main strength—its generalizability can be limited. Furthermore, the application of risk prediction toward component replacement and integration requires the support of software engineering assessment and decision making.¹³ Finally, other metrics might have different prediction power for abandonment risk.

Integrating abandoned OSS in software-intensive systems is hazardous and could result in severe consequences, which concerns practitioners. Especially when functions from abandoned components are integrated into highly critical modules, the consequences from abandoned OSS that lacks maintenance can be unbearable. To foresee such risks, we proposed OSSARA to provide an assessment and prediction pipeline. The model also supports continuous adaptation and customization through which practitioners can conduct optimized prediction via up-to-date OSS activity data, with selectively effective and even customized algorithms. The model was positively received by our case company, which is adopting and integrating it into its continuous integration/continuous deployment pipeline. Future work will explore other analysis techniques, the inclusion of different metrics in the prediction model, and the integration of various types of OSS risk assessment, e.g., security assessment, license compliance assessment, and so on, as well as dependency and hierarchy analysis. 

References

1. M. M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.
2. I. Samoladas, L. Angelis, and I. Stamelos, “Survival analysis on the duration of open source projects,” *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 902–922, 2010, doi: 10.1016/j.infsof.2010.05.001.
3. J. Businge, A. Serebrenik, and M. van den Brand, “Survival of eclipse third-party plug-ins,” in *Proc. 2012 28th IEEE Int. Conf. Softw. Maintenance*

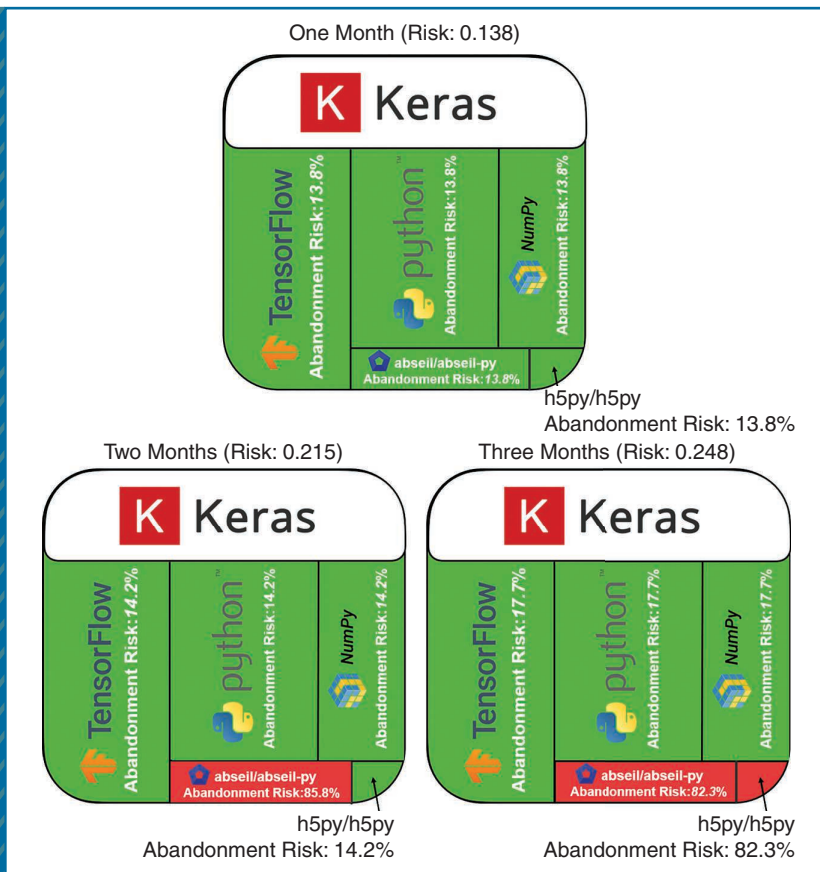


FIGURE 2. The abandonment risk assessment for Keras.

- (ICSM), pp. 368–377, doi: 10.1109/ICSM.2012.6405295.
4. J. Coelho, M. T. Valente, L. L. Silva, and E. Shihab, “Identifying unmaintained projects in GitHub,” in *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2018, pp. 1–10, doi: 10.1145/3239235.3240501.
 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,” *Inf. Softw. Technol.*, vol. 122, p. 106,274, Jun. 2020, doi: 10.1016/j.infsof.2020.106274.
 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 *Proc. 2018 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 644–655, doi: 10.1145/3236024.3236062.
 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,” 2021, *arXiv:2107.10168*.
 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,” *J. Syst. Softw.*, vol. 188, p. 111,255, Jun. 2022, doi: 10.1016/j.jss.2022.111255.
 9. B. Boehm and C. Abts, “COTS integration: Plug and pray?” *Computer*, vol. 32, no. 1, pp. 135–138, 1999, doi: 10.1109/2.738311.
 10. B. Boehm, “Software risk management,” in *Proc. Eur. Softw. Eng. Conf.*, Springer-Verlag, 1989, pp. 1–19, doi: 10.1007/3-540-51635-2_29.
 11. J. Khondhu, A. Capiluppi, and K.-J. Stol, “Is it all lost? A study

ABOUT THE AUTHORS



XIAOZHOU LI is a postdoctoral researcher in the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, 33720, Finland, where he is a researcher in the Cloud Software Evolution and Assessment group. Li received his Ph.D. in computer science from Tampere University. His research interests include open source software quality, software maintenance and evolution, and user review opinion mining. Contact him at xiaozhou.li@tuni.fi.



SERGIO MORESCHINI is a Ph.D. candidate in the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, 33720, Finland, where he is a researcher in the Cloud Software Evolution and Assessment group. His research interests include extended light field reconstruction for continuous parallax content. Contact him at sergio.moreschini@tuni.fi.



FABIANO PECORELLI is a postdoctoral researcher in the Cloud Software Evolution and Assessment group, Tampere University, Tampere, 33720, Finland. His research interests include software code and test quality, predictive analytics, and mining software repositories. Pecorelli received his M.S. in computer science from the University of Salerno, Italy. Contact him at fabiano.pecorelli@tuni.fi.



DAVIDE TAIBI is an associate professor at Tampere University, Tampere, 33720, Finland, where he heads the Cloud Software Evolution and Assessment research group. His research interests include empirical software engineering applied to cloud-native systems. Taibi received his Ph.D. in computer science from University of Insubria. He is a member of the International Software Engineering Network. Contact him at davide.taibi@tuni.fi.

- of inactive open source projects,” in *Proc. IFIP Int. Conf. Open Source Syst.*, Springer-Verlag, 2013, pp. 61–79, doi: 10.1007/978-3-642-38928-3_5.
12. J. Coelho and M. T. Valente, “Why modern open source projects fail,” in *Proc. 2017 11th Joint Meeting Found. Softw.*

- Eng.*, 2017, pp. 186–196, doi: 10.1145/3106237.3106246.
13. R. A. Ribeiro, A. M. Moreira, P. Van den Broek, and A. Pimentel, “Hybrid assessment method for software engineering decisions,” *Decis. Support Syst.*, vol. 51, no. 1, pp. 208–219, 2011, doi: 10.1016/j.dss.2010.12.009.