Towards automated open source assessment - An empirical study

Sai Pranav Koyyada[1], Denim Deshmukh[1], Deepika Badampudi[1,*], Vida Ahmadi[1,2] and Muhammad Usman[1]

[1]Blekinge Institute of Technology, Sweden
[2]City Network International AB, Sweden

Abstract
The open source software (OSS) assessment has become important given the increased adoption of OSS in commercial product development. Researchers proposed many OSS assessment models. However, little is known about the industrial relevance of the models. In this study, we proposed an automated tool based on the OSS assessment attributes identified together with a European cloud provider company. We analyzed 51 repositories to observe patterns in maintenance activities over their lifetime (from inception to the latest release). Based on the analysis, we propose a novel approach for evaluating the maturity of the OSS project. Finally, we assessed the usefulness of our automated solution in a pilot study.

Keywords
OSS assessment automation, commit classification, software maturity

1. Introduction

Software companies increasingly adopt Open Source Software (OSS), which has become part of the mainstream practice in software engineering [1]. However, the selection of OSS is still challenging [2]. The practitioners in our case company: a European cloud provider, reported similar challenges. They mentioned gathering information from multiple sources and tools as a complex and time-consuming activity. The case company identified the need to automate the OSS assessment to reduce the selection effort.

The automation of the OSS assessment requires the identification of the attributes practitioners consider for OSS selection. In the last two decades, many OSS quality models have been proposed to assist the OSS selection. Lenarduzzi et al. [2] identified discrepancies in the information provided in the evaluation models and the practitioners' information needs. In addition, little is known about how relevant these models are in practice as they have not been validated extensively[2]. The OSS quality assessment models suggest many evaluation attributes. Maintenance is one of the most considered quality attributes in the OSS assessment models proposed in the previous studies [3]. Li et al. [4] conducted a survey to understand the attributes practitioners consider important in OSS selection. Practitioners mention maintenance as an important attribute; however, they did not mention metrics for assessing maintenance [4]. Metrics are important to automate quality assessment. However, practitioners did mention metrics for assessing software maturity, such as the number of forks, number of releases, and number of commits [4]. However, Li et al. [4] suggested that while some practitioners consider the number of commits as a metric to evaluate software maturity, evaluating the prevalence of commits over time and the types of commits may be more useful.

Levin and Yehudai [5] proposed a model to classify the comments based on the maintenance activities. However, their motivation to classify was to improve planning and resource allocation for maintenance. As indicated by Li et al. [4], the prevalence of commits over time and types of commits could be a good measure of maturity. However, they did not find any portals that effectively provide community-related factors to automate OSS project assessment [4]. Therefore, it is interesting to investigate how commit classification based on maintenance activities can help automate OSS assessment.

The study aims to identify commonly considered attributes in OSS selection in the case company and investigate to what extent commit classification based on maintenance activities can help in the OSS assessment. Many models for commit classification exist [6, 7, 5, 8]. Our objective is not to propose the most accurate classification model but to demonstrate the use of commit classification in the OSS assessment. We used Levin's commit classification model [5] to monitor different

ISE22: 1st International Workshop on Intelligent Software Engineering (ISE)
*Corresponding author.
saky19@student.bth.se (S. P. Koyyada); dede19@student.bth.se (D. Deshmukh); deepika.badampudi@bth.se (D. Badampudi); vida.ahmadi.mehri@bth.se (V. Ahmadi); muhammad.usman@bth.se (M. Usman)

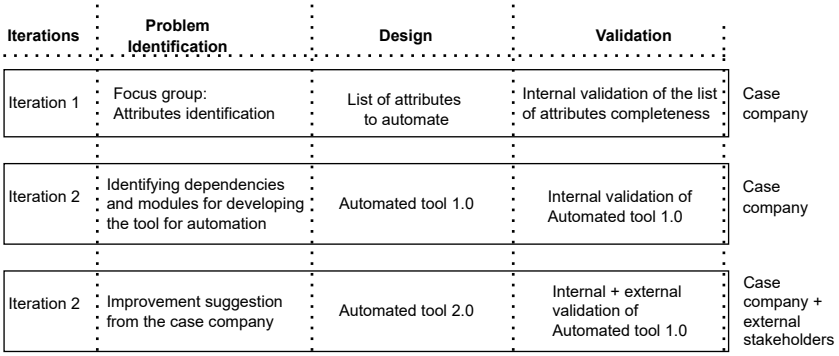

Figure 1: Research design followed in the study

maintenance activities carried out in OSS projects. We analyzed 51 OSS projects, including frameworks, APIs, libraries, databases, and applications, to collect attributes that can help facilitate the OSS assessment. Finally, we conducted a pilot qualitative study to understand practitioners' opinions on the usefulness of the commit classification based on maintenance activities and commonly considered selection attributes in the OSS assessment.

2. Research methodology

Our goal is to answer the following research questions -

- RQ1: What OSS attributes are considered important to automate in the case company?
- RQ2: How can commit classification be used to facilitate OSS assessment?
- RQ3: How do practitioners perceive the usefulness of our automated solution?

We used design science method[9] to answer the above research questions. Figure 1 depicts the iterations in the solution development and validation. The steps carried out in each iteration are described as follows.

Iteration 1: In this iteration our goal is to identify the attributes that the case company considers important for automating OSS assessments. We used focus groups where key stakeholders from the case company and the authors discussed the different attributes. The input to the focus group was the case company's checklist for OSS assessment and the attributes frequently reported in the literature. The goal of the focus group was to identify attributes that can be automated to improve the efficiency and effectiveness of the OSS assessment. The outcome of the focus group was the list of attributes to automate. The first two authors were employed at the case company, providing them easy access to the developers involved in the OSS assessment. For internal validation, the first two authors discussed with three developers on the completeness of the attributes. Overall the developers agreed with the list of attributes.

Iteration 2: We identified the different sources to retrieve the required attributes. We used various API Endpoints such as GitHub REST API, Stack Exchange REST API, python libraries such as pydriller, OWASP Dependency Checker, and other OSS solutions to gather the relevant information. We built an automated solution that could be triggered with a single command and return all results from the assessment in the JSON format which we refer to as Automated tool 1.0 in Figure 1. We discuss the tool in a focus group with the security expert and three developers. Our solution used the JSON files generated output to present the assessment results. The focus group participants from the case company requested a feature that pooled all the results on one page.

Iteration 3: The problem identification for this iteration was the input from the validation in Iteration 2. Therefore, we created a module that could show the results from various JSON files generated from our automated solution on one page which we refer to as Automated tool 2.0 in Figure 1. The source code, modules used in the automated solution, and the documentation of the tool usage are provided online[1]. We verified the functioning of our automated tool by assessing 51 different OSS projects. Our automated solution generated the expected results. We presented the automated solution through a technical demo at the case company. During the technical demo, we assessed two OSS: one framework and one application. The results took under a minute to generate. We finally interviewed 10 developers from three different companies (including the case company) to validate the completeness of the assessment attributes and the usefulness of the automated tool.

[1] https://github.com/SaipranavK/oss-recon

Table 1
OSS assessment attributes considered at the case company

Attributes	Metrics and information
Repository information	Name, description, topics, API URL, programming languages, and Github community health percentage.
Repository activeness	Age, last updated date, average time to release, number of open issues, active/recent releases, The commit activity, commit classification.
Security	Vulnerabilities.
Community interest	Stars, forks and watchers.
Support	Stack overflow QAs.
Legal requirements	License type, permissions, conditions and limitations.

3. Results

This section presents our results from the focus group study, analysis of commit classification, and preliminary validation based on practitioners' perceptions of our automated solution.

3.1. OSS assessment attributes

Identifying the attributes required in the OSS assessment to automate the process is important. We discussed the attributes, the required metrics, and the information to automate the OSS assessment in the focus group. The input to the focus group was the collection of attributes and metrics frequently considered in the literature and the case company. We selected the attributes and metrics based on their importance and the ease of understanding perceived by the case company. The case company preferred using descriptive representation than numerical metrics-based representations. Therefore, the case company did not want information on traditional metrics like code complexity, coupling, cohesion, and other similar metrics. This section presents the attributes for assessing the OSS projects. Table 1 contains the attributes considered important for assessing OSS projects by the case company. We also present the metrics and information gathered to access the attributes (see the second column in Table 1).

Repository information: The company starts assessing the OSS project by reviewing general repository information (see details on repository information in Table 1).

Repository activeness: In addition to the generic information, the company reviews the repository's activeness by reviewing the average time it takes to release a new version, the number of open issues, and the list of active or recent releases of the OSS project. In addition to the company's attributes, we added age, last updated date, commit activities: the number of deletions and additions, and commit classification: the number of corrections, adaptions and perfective activities metrics.

Security: The security expert at the case company identified security as an important criterion for adopting OSS projects.

Community interest: In addition, the company reviews the support availability by considering the number of questions and answers posted with tags associated with the OSS project on StackOverflow.

3.2. Commit classification based on maintenance activities to evaluate OSS projects

We propose using commit classification, among other metrics, to evaluate OSS projects. Each new release of an OSS has additions or deletions compared to the previous version. These additions and deletions are changes that introduce new features, fix bugs, or extend the support of OSS. The changes implemented in a release are called maintenance activities. The Software Engineering Body of Knowledge(SWEBOK) and IEEE14764 categorize software maintenance activities as corrective, adaptive, perfective, and preventative. Preventative and corrective activities are corrections, given their purpose to fix latent and operational faults. Perfective and adaptive activities are modifications to improve the software.

We visualized the commit activeness based on the maintenance activities from the inception to the latest release of the OSS Project. We used the approach proposed by Levin et al.[5] to classify the maintenance activities. Figure 2 shows the commit classification of an example OSS across its different releases. Each release has a distribution of three activities: corrective in red, perfective in yellow, and adaptive in blue. Preventative activities are proactive activities, and there are comparatively difficult to capture for classification. Therefore, our study did not include classification based on preventative activities.

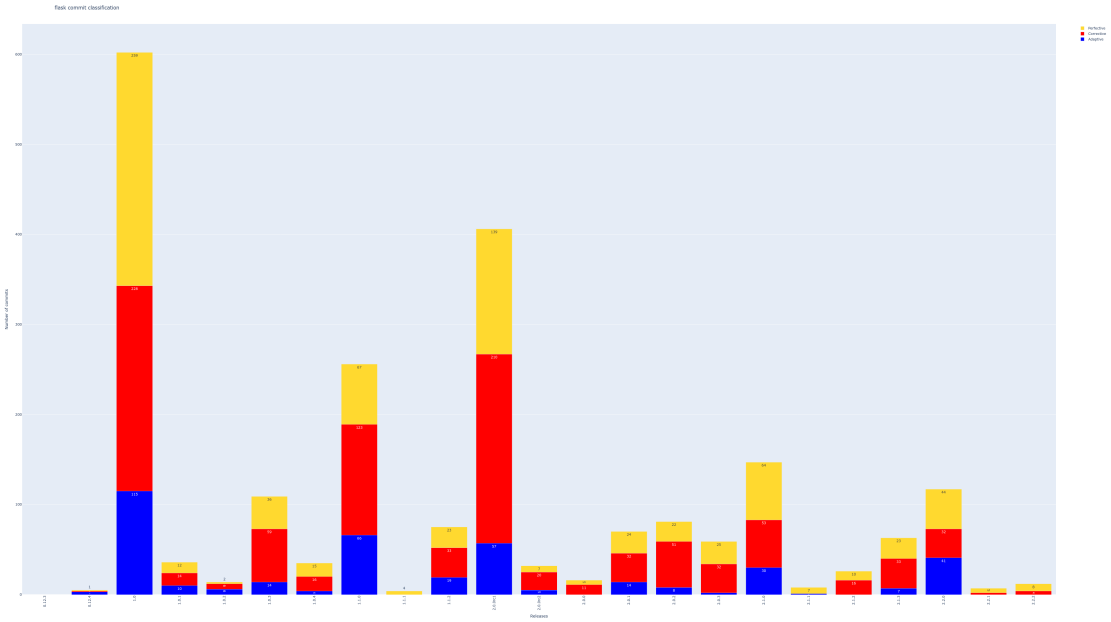

Figure 2: Commit classification of OSS flask.

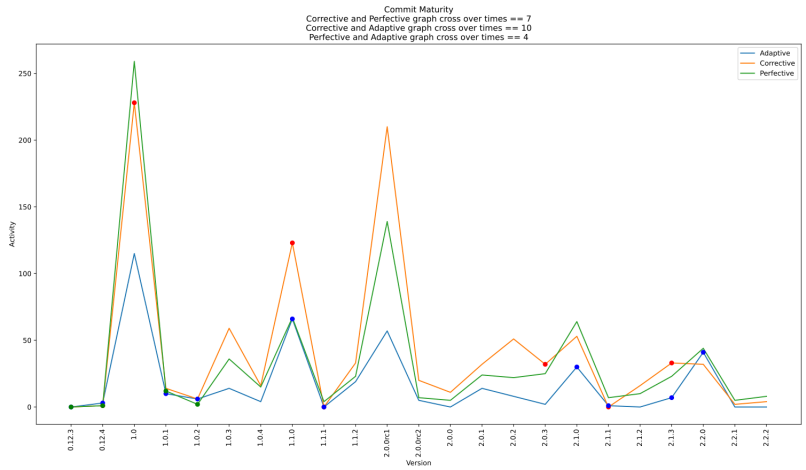

Figure 3: Commit maturity of Flask repository.

3.2.1. Commit maturity: A novel perspective on commits classification

We analyzed 51 OSS repositories, including frameworks, APIs, libraries, databases, and applications. The visualizations on commit classification on 51 open source repositories used for analysis are provided online[2]. We observed the maintenance activities of the repositories from their inception to the latest release. The analyzed 51 OSS included very popular repositories with more than 10000 stars, popular repositories with over 5000 stars, and some growing OSS with over 500 stars on GitHub.

We observed adaptive activity is the least performed activity in each release for all the OSS repositories. Corrective and perfective activities have the most variance, i.e., the number of corrective activities may be higher than perfective in some commits and lower in others.

[2]https://doi.org/10.5281/zenodo.7053198

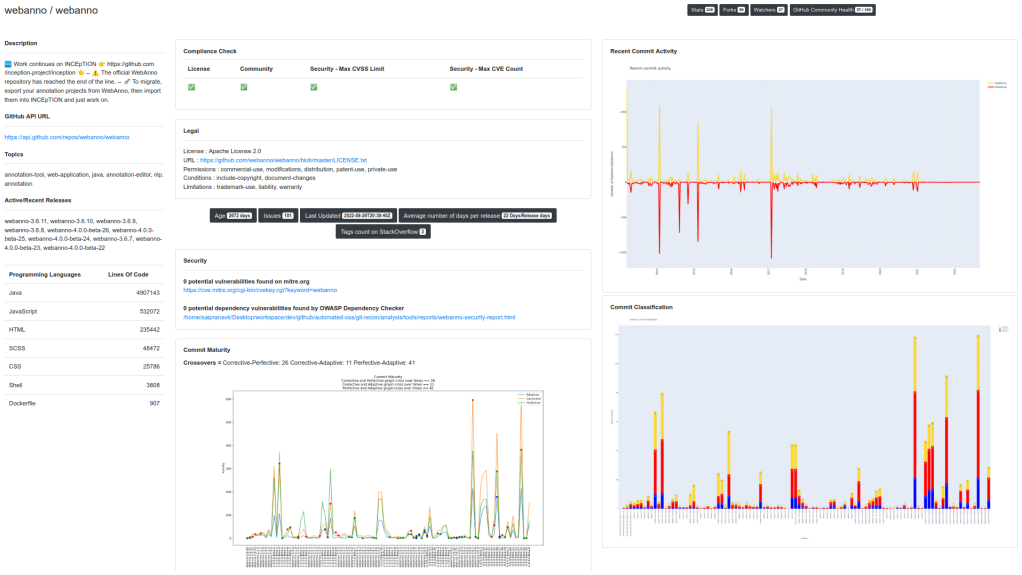

Figure 4: Sample report from our automated solution for OSS webanno.

When a certain maintenance activity's frequency goes down, and another maintenance activity's frequency goes up, we call it a crossover.

We counted the number of crossovers for each combination of the maintenance activities and mapped them with the type of the OSS. We noticed that all OSS except the ones of type frameworks had a similar number of crossovers for each combination of maintenance activities.

Lehman's law [10] suggests that a system should continuously change to remain useful. A change can be measured by the number of counts of corrective, adaptive, and perfective requests over a certain period [11]. Any software project should have good features with minimum bugs and extensive support. Whenever a crossover happens, it indicates that the focus of the community shifts from one maintenance activity to another to make sustainable progress. If the community only focuses on adding new features, then there may be many bugs making the OSS unusable. The same applies when the community is only resolving bugs in the OSS. It means that the OSS has many bugs to be addressed and does not introduce new features to improve its value. The balance between the maintenance activities should be maintained. Frameworks are unique in this regard because of the scale of features and platforms they support. As we saw from our analysis, it may not be possible to maintain the balance between the maintenance activities for frameworks. We hypothesize that the total number of crossovers of each combination of maintenance activities should ideally be similar to the total number of releases. Crossover between Adaptive and Corrective activities =

$$(Ai - 1 > Ci - 1) \cdot (Ai < Ci) \quad (1)$$

Where (Ai) is a list of Adaptive Activities and (Ci) is a list of Corrective Activities for each version of the software. Similarly, we can calculate three types of intersections. The number of intersections between each pair of activities, i.e., Adaptive, Corrective, and Perfective, defines commit maturity.

Commit maturity is the number of times each maintenance activity crosses other maintenance activities over the project life cycle. It will allow practitioners to see if the OSS project maintains the balance between the maintenance activities. Figure 3 shows the commit maturity and the number of crossovers for each crossover pair. Each dot represents an occurrence of a crossover in a release. In this example, in release 1.0, the corrective activities decreased, and the perfective activities increased, resulting in a crossover. The flask project has a total crossover count of 21 out of 23 releases. Based on our hypothesis, it is a good maturity indication, and the popularity of the flask project is a testimony to it.

The case company requested all the information needed to evaluate the OSS repository on one page. Figure 4 provides a screenshot of our solution. The left panel includes information on the repository, while the middle panel provides information on the metrics to assess security, support, and legal requirements. The middle panel also includes metrics to evaluate repository activeness: age, last updated date, the average time to release, and the number of issues. The commit activity, commit

classification, and commit maturity are represented in graphical format. Finally, the community's interest: stars, forks, and watchers are presented in the top right corner.

3.3. Practitioners perception of the automated OSS assessment and commit activeness

Completeness of the evaluation attributes: we aimed to evaluate if the practitioners found our automated solution useful for automatically assessing OSS. All the participants agreed that the tool could help in the OSS assessments. Some of the participants wanted to see more attributes. For example, one of the interviewees mentioned *"I want to gather more information like its compatibility with different operating systems and the tutorials sources."*. Since we designed our solution primarily for use in the case company, it is not surprising that interviewees wished for additional attributes. One solution could be to create a configurable solution where the stakeholders can select the important attributes in the assessment.
Ease of understanding the information on OSS assessment attributes: We explained the attributes to the interviewees, particularly the attributes such as commit maturity. We asked the interviewees if the attributes we used were easy to understand. All the participants unanimously agreed that our attributes were easy to understand. The interviewees added that *"The attributes were easy to understand. It was simply like a GitHub page but with more information."*. In addition, the interviewees were positive about the new attributes such as commit maturity: *"The commit evolution and maturity was something new but were still very easy to understand along with other attributes."*.
Commit classification and maturity: We asked the interviewees if commit classification and commit maturity are good visualizations to support the OSS adoption decision. All the interviewees agreed that the visualization was useful once we explained the attributes to them. One of the interviewees mentioned *"Managers would love such a visualization because it not only is simple to understand but also will help a non-technical person easily comment on the OSS community."*.

4. Conclusion and future work

We reported initial findings from an empirical study to support the practitioners in the OSS assessment process. With the help of the practitioners in the case company, we first identified the attributes that could be automated in performing the OSS assessment. Our tool automatically collects and presents the data about the identified attributes in one place to facilitate the practitioners in performing the OSS assessment. We also investigated how commit classification based on different maintenance activities can be used in OSS assessment. We introduce the use of commit maturity to see if an OSS project is balanced in feature enhancements and bug fixes or overly focused on only one type of maintenance activity (e.g., only fixing bugs in case of corrective maintenance). We used our tool to analyze 51 OSS projects. We also shared the results with the company practitioners, who found our tool helpful in performing the OSS assessments. In the future, we plan to study commit maturity as a metric to assess OSS maintainability through extensive validation and application and standardize it for wider adoption. Additionally, we will continue to enhance our automated OSS assessment tool by improving the range of supported attributes and desired metric outputs. We also wish to employ repository mining techniques to identify and correlate community activities with the OSS engagement and growth trends to comment on its popularity and support. Another interesting direction of research would be correlating maintenance activities for an OSS with its traditional maintainability metrics that can help evaluate the relationship between maintainability and maintenance activities, if any, and thus result in more branching paths of research in software metrics and the maintainability domain.

Acknowledgment

The Knowledge Foundation supports this work through the OSIR project (reference number 20190081) at Blekinge Institute of Technology, Sweden.

References

[1] G. Robles, I. Steinmacher, P. Adams, C. Treude, Twenty years of open source software: From skepticism to mainstream, IEEE Software 36 (2019) 12–15. doi:10.1109/MS.2019.2933672.
[2] V. Lenarduzzi, D. Taibi, D. Tosi, L. Lavazza, S. Morasca, Open source software evaluation, selection, and adoption: a systematic literature review, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2020, pp. 437–444.
[3] N. Yılmaz, A. Kolukısa Tarhan, Quality evaluation models or frameworks for open source software: A systematic literature review, Journal of Software: Evolution and Process (2022) e2458.
[4] X. Li, S. Moreschini, Z. Zhang, D. Taibi, Exploring factors and metrics to select open source software components for integration: An empirical study, Journal of Systems and Software 188 (2022) 111255.
[5] S. Levin, A. Yehudai, Boosting automatic commit classification into maintenance activities by utiliz-

ing source code changes, in: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, 2017, pp. 97–106.

[6] A. Hindle, D. M. German, M. W. Godfrey, R. C. Holt, Automatic classication of large changes into maintenance categories, in: 2009 IEEE 17th International Conference on Program Comprehension, IEEE, 2009, pp. 30–39.

[7] S. Gharbi, M. W. Mkaouer, I. Jenhani, M. B. Messaoud, On the classification of software change messages using multi-label active learning, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1760–1767.

[8] L. Ghadhab, I. Jenhani, M. W. Mkaouer, M. B. Messaoud, Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model, Information and Software Technology 135 (2021) 106566.

[9] R. J. Wieringa, Design science methodology for information systems and software engineering, Springer, 2014.

[10] M. M. Lehman, Programs, life cycles, and laws of software evolution, Proceedings of the IEEE 68 (1980) 1060–1076.

[11] E. J. Barry, C. F. Kemerer, S. A. Slaughter, How software process automation affects software evolution: a longitudinal empirical analysis, Journal of Software Maintenance and Evolution: Research and Practice 19 (2007) 1–31.