The Journal of Systems & Software 188 (2022) 111255

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Exploring factors and metrics to select open source software components for integration: An empirical study☆

Xiaozhou Li[1], Sergio Moreschini[1], Zheying Zhang, Davide Taibi*

Tampere University, Tampere, Finland

ARTICLE INFO

Article history:
Received 9 January 2021
Received in revised form 6 December 2021
Accepted 1 February 2022
Available online 11 February 2022

Keywords:
Open source
Software selection
Open source adoption

ABSTRACT

Context: Open Source Software (OSS) is nowadays used and integrated in most of the commercial products. However, the selection of OSS projects for integration is not a simple process, mainly due to a of lack of clear selection models and lack of information from the OSS portals.
Objective: We investigate the factors and metrics that practitioners currently consider when selecting OSS. We also investigate the source of information and portals that can be used to assess the factors, as well as the possibility to automatically extract such information with APIs.
Method: We elicited the factors and the metrics adopted to assess and compare OSS performing a survey among 23 experienced developers who often integrate OSS in the software they develop. Moreover, we investigated the APIs of the portals adopted to assess OSS extracting information for the most starred 100K projects in GitHub.
Result: We identified a set consisting of 8 main factors and 74 sub-factors, together with 170 related metrics that companies can use to select OSS to be integrated in their software projects. Unexpectedly, only a small part of the factors can be evaluated automatically, and out of 170 metrics, only 40 are available, of which only 22 returned information for all the 100K projects. Therefore, we recommend project maintainers and project repositories to pay attention to provide information for the project they are hosting, so as to increase the likelihood of being adopted.
Conclusion: OSS selection can be partially automated, by extracting the information needed for the selection from portal APIs. OSS producers can benefit from our results by checking if they are providing all the information commonly required by potential adopters. Developers can benefit from our results, using the list of factors we selected as a checklist during the selection of OSS, or using the APIs we developed to automatically extract the data from OSS projects.

1. Introduction

Open Source Software (OSS) has become mainstream in the software industry, and different OSS projects are now considered as good as closed source ones (Robles et al., 2019; Kilamo et al., 2020). However, selecting a new OSS project requires special attention, and companies are still struggling to understand how to better select them (Lenarduzzi et al., 2020).

One of the main issues during the selection of OSS projects, is the lack of clear information provided by OSS providers about the software quality assessment, and in particular the lack of automated tools that help the selection (Lenarduzzi et al., 2020).

A local company hired our research group to ease and standardize the OSS selection process and to automate it as much as possible, to reduce the subjectivity and the effort needed for the evaluation phase. Currently, the company does not prescribe any selection model, and reported us that their developers commonly struggle to understand what they need to consider when comparing OSS projects.

In this paper, we investigate the first steps towards the definition of a semi-automated OSS evaluation model. Therefore, we extend our previous work (Lenarduzzi et al., 2020) by conducting a survey investigating the factors commonly considered by the companies when selecting OSS, the source of information that can be used to analyze these factors, and the availability of such information on the portals.

The goal of our work is to investigate and determine the factors that practitioners are currently considering when selecting OSS, to identify the sources (portals) that can be used to evaluate such factors mentioned by the practitioners, and to validate the public APIs that can be accessed to automatically evaluate those factors from the sources and portals.

☆ Editor: Raffaela Mirandola.

* Corresponding author.
E-mail addresses: xiaozhou.li@tuni.fi (X. Li), sergio.moreschini@tuni.fi (S. Moreschini), zheying.zhang@tuni.fi (Z. Zhang), davide.taibi@tuni.fi (D. Taibi).

[1] The two authors equally contributed to the paper.

https://doi.org/10.1016/j.jss.2022.111255

The research community has been studying OSS selection and evaluation from different perspectives.

Researchers developed methods to evaluate, compare, and select OSS projects. Such methods and tools exploit different types of approaches, including manual extraction of data from OSS portals (e.g. OMM Petrinja et al., 2009, OpenBQR Taibi et al., 2007a, PSS-PAL Wasserman et al., 2017).

Researchers also proposed platforms for mining data from OSS repositories, that can also be used as the sources of information for the evaluation and comparison of OSS (e.g., The SourceForge Research Data Archive (SRDA) Madey, 2008, FLOSSmole Anon, 2020a, FLOSSMetrics Anon, 2020b, tools to provide dump of existing OSS portals (e.g. GHArchive Anon, 2020c, GitTorrent Anon, 2020d, …), and tools to extract information from OSS portals (e.g. PyDriller Spadini et al., 2018, CVSAnaly Robles et al., 2004...).

Moreover, different approaches to evaluate software quality, often applied to OSS, have been proposed in research, including software metrics (e.g. Chidamber and Kemerer's metrics suite Chidamber and Kemerer, 1994, Cyclomatic Complexity McCabe, 1976), tools to detect technical debt Avgeriou et al., 2020 or to measure other quality aspects (e.g. Software Quality Index Dixon, 2016, Architectural Debt Index Roveda et al., 2018).

Though the previous work provided a significant amount of results on OSS quality evaluation, OSS development data crawling, and OSS selection and adoption models, such works are still limited and not easy to apply in industry for selecting OSS because of various reasons:

- OSS selection models
  - Limited application in industry of the previous OSS selection models Lenarduzzi et al., 2020. The vast majority of models have never been adopted massively by industries with neither case studies nor success stories on the usage of these models therein. One of the potential reasons is that it is nearly not possible to have a generally accepted set of OSS selection criteria to use. The companies must adopt the criteria for their specific needs and constraints to achieve their business objectives. Another reason for such limited adoption of the selection models can be related to the lack of maturity of the models. The models lack clarity and guides about which metrics would offer the most relevant insights into the selection criteria.
- OSS Mining Platforms such as The SourceForge Research Data Archive (SRDA) (Madey, 2008), GHArchive, GitTorrent (Gousios, 2013a)
  - They are designed for research purposes and are complex to use and often have different dependencies for developers that simply need to get data for an OSS project. As an example, GitArchive (Anon, 2020c) does not allow to directly query the data with an API, but needs to be accessed through Google Big Query or dumping the files. Moreover, the collection of all the information needed by the users to evaluate an OSS project requires to use several platforms. This study aims to provide an overview on what information is important to the companies and from what platforms to extract it when evaluating OSS projects.
  - They are often not maintained in the long term. As an example, The SourceForge Research Data Archive (SRDA) (Madey, 2008) is not available anymore. The GHTorrent (Gousios, 2013a) was created in 2013 with the last activity reported in 2019. More information on these platforms is available in Table 1.
- Tools to evaluate software quality
  - They are complex to use and often require effort for manual configuration and analysis on the target software.
  - Most tools require expertise to understand which metrics should be used in which context, and how to interpret the evaluation results. They often provide an overload of information, but not always useful for every context. As an example, tools for assessing the quality and technical debt of software, such as SonarQube,[2] include more than 500 different rules to validate the source code, but only a limited amount of the rules that are commonly associated to specific qualities.
  - When existing tools focus on a specific set of quality metrics to do the evaluation, there is a lack of a tool that can aggregate the factors commonly considered during the selection of OSS.
  - The existence of an OSS project community and of a health ecosystem is an informative indicator of the maturity of a software and its propensity for growth. Even though there are many community-related factors and metrics identified in the OSS selection model, some metrics cannot be accessed directly from the project's repository and existing tools provide very limited support to analyze the data associated with the OSS project community and its support in OSS evaluation.
- Existing Software Quality Models and Metrics such as the Architectural Quality index (Roveda et al., 2018), but also metrics such as the Cyclomatic complexity (McCabe, 1976) or the presence of Code Smells (Fowler, 1999) or anti-patterns (Brown et al., 1998).
  - Are usually targeting mainly on quality, while companies might be interested in other aspects while selecting OSS (e.g. Costs, licenses, features, …).
  - Lack of comparison of the magnitude of the observed effect: different models return different outputs. As an example, previous works indicated that high levels of cyclomatic complexity might result in less readable source code. While besides that, the presence of some smells can be more harmful than others, but the analysis did not take into account the magnitude of such observed phenomenon. As an example, it is not clear if a piece of code with a cyclomatic complexity equals to 10 is twice more complex to read than the same piece of code with that of 5. The same applies to the comparison of several other metrics, including the presence of different amount of code smells. Thus, the comparison of the results of the metrics, increases the complexity of the analysis and comparison between projects.
  - Lack of complex and historical analysis. A complete comparison of an OSS might require not only the analysis of the latest snapshots of a project, but a historical analysis, thus increasing the complexity and the effort required to perform an analysis.

In addition, lack of expertise in companies, in particular on software quality, hinders practitioners to select the most suitable quality models for comparing the projects.

To cope with the aforementioned issues, this paper aims at corroborating and extending previous empirical research on OSS

[2] SonarQube http://www.sonarqube.org.

Table 1
OSS source of information and portals reported in the literature.

Portal	Created on	Last activity	Information stored
OSS aggregators			
(Anon, 2020p) Software Heritage	2018	2020	Source Code
(Anon, 2020f) OpenHub	2004	2020	OSS tracker
(Anon, 2020g) FlossHub	2008	2018	Research portal
(Anon, 2020e) SourceForge Research Data	2005	2008	Statistics
(Anon, 2020c) GH Archive	2012	2020	Timeline Record
(Anon, 2020d) GH Torrent	2013	2019	GH event monitoring
(Anon, 2020a) FLOSSMole	2004	2017	Project data
(Anon, 2020q) PROMISE	2005	2006	Donated SE data
(Anon, 2020b) FLOSSMetrics	2006	2010	Metrics and Benchmarking
Audit and analysis tools			
(Anon, 2020k) WhiteSource	2011	2020	Security
(Anon, 2020r) FossID	2016	2020	OS Compliance and Security
(Anon, 2020j) Synopsys (formerly BlackDuck)	2012	2020	legal, security, and quality risks
(Anon, 2020l) SonarQube	2006	2020	Code quality and security
(Anon, 2020n) WhiteHat	2001	2020	Software composition analysis
(Anon, 2020l) SonarCloud	2008	2020	Software quality analysis
OSS mining data tools			
(Anon, 2020s) BOA	2015	2019	Source code mining
(Anon, 2020h) Candoia	2016	2017	Software repository mining
(Anon, 2020i) RepoGrams	2016	2020	OSS Comparison
Questions and answers portal			
(Anon, 2020t) Stack Exchange	2009	2020	Q&A
(Anon, 2020u) Reddit	2009	2020	Q&A

selection and adoption, so as to enable, not only our target company to assess and compare OSS but also other companies. More specifically, this study aims at extending our previous work (Lenarduzzi et al., 2020) from different point of views:

- We performed a survey to update the common factors that are currently considering when selecting OSS and we compared them with the factors considered in the past (elicited in the Euromicro/SEAA SLR Lenarduzzi et al., 2020)
- We analyzed the source of information and portals that can be used to assess the aforementioned factors
- We analyzed the public APIs that can be accessed to automatically assess the factors from the aforementioned portals
- We extracted the information for 100K projects, to validate their availability.

The source of information associated with the factors and metrics adopted to measure them will help developers to understand and adopt OSS selection models for their specific needs and constraints. Moreover, they will help to remind OSS producers to provide information commonly expected by the potential users of the software.

Together with the validated APIs to automatically extract the assessment information, the result of the work forms a critical step towards developing semi-automatic tools to facilitate the practice of OSS selection.

The remainder of this paper is structured as follows. Section 2 presents related works. Section 3 describes the research method we adopted to achieve our goals. Section 4 reports the results while Section 5 discuss them. Section 6 finally draws conclusions and future works.

2. Related work

To cope with the need of selecting valuable OSS projects, several evaluation models have been proposed (e.g. Duijnhouwer and Widdows (2003), Golden (2008), Taibi et al. (2007b) and Semeteys (2008)). At the same time, different research groups proposed project aggregators to ease the access of different information on OSS, measures and other information. Last, but not least, research in mining software repositories also evolved in parallel, and different researchers provided datasets of OSS projects, portals and tools to extract information from OSS projects.

In the remainder of this Section, we summarize related work on the factors adopted to evaluate OSS, OSS evaluation and selection models, OSS aggregator portals, tools for OSS repository mining and tools for OSS analysis and audit.

2.1. The factors considered during the adoption of OSS

In the systematic literature review (SLR) on OSS selection and adoption models (Lenarduzzi et al., 2020) that we are extending in this work, we analyzed 60 empirical studies, including 20 surveys, 5 lessons learned on OSS adoption motivation and 35 OSS evaluation models.

Regarding the common factors of OSS selection and adoption, eight main categories were reported by the selected studies, including, *Community and Adoption*, *Development process*, *Economic*, *Functionality*, *License*, *Operational software characteristics*, *Quality*, *Support and Service*. For each category, sub-factors or metrics are reported. Results show that not all factors were considered equally important according to evaluation models and to surveys and lessons learned. For example, factor *cost* is considered much more important by the surveys than by the models when, on the contrary, the importance of factor *maturity* is seen oppositely. Furthermore, certain factors are considered important by both groups, such as, *Support and Service*, *Code Quality*, *Reliability*, etc.

Table 1 lists sources of information mentioned in the related works to assess the common factors considered during the adoption of OSS. The table only reports the indirect sources of information. Direct sources of information such as the official portal, or the versioning system (e.g. GitHub, GitLab) are not mentioned in the table.

2.2. OSS evaluation and selection models

Within the 35 OSS models identified in our previous literature review (Lenarduzzi et al., 2020), 21 (60%) were built via case study, with 5 via interview, 5 via experience and the other 4 via

the combination of interview and case study. All proposed models provide either checklist (13 models) or measurement (8 models) or both (14 models) as their working approaches. On the other hand, regarding the studies with surveys and lesson learned, the majority (13) target at adoption motivation identification with the remainders on other scopes. Furthermore, 12 tools are introduced in 22 of the given studies; however, only two out of the 12 are properly maintained.

All the models propose to evaluate OSS with a similar approach:

- *Identification of OSS candidates*. In this step, companies need to identify a set of possible candidates based on their needs.
- *Factors evaluation* A list of factors are then assessed, by extracting the information or measures from the OSS portals, or by measuring/running the project candidates
- *Project scoring* The final score is then normalized based on the importance of each factor and the final evaluation is computed.

The Open Source Maturity Model (OSMM), was the first model proposed (Duijnhouwer and Widdows, 2003; Golden, 2008) in the literature. OSMM is an open standard that aims at facilitating the evaluation and adoption of OSS. The evaluation is based on the assumption that the overall quality of the software is proportional to its maturity. The evaluation is performed in three steps:

1. Evaluation of the maturity of each aspect. The considered aspects are: the software product, the documentation, the support and training provided, the integration, the availability of professional services.
2. Every aspect is weighted for importance. The default is: 4 for software, 2 for the documentation, 1 for the other factors.
3. The overall maturity index is computed as the weighted sum of the aspects' maturity.

The OSMM has the advantage of being simple. It allows fast (subjective) evaluations. However, the simplicity of the approach is also a limit: several potentially important characteristics of the products are not considered. For instance, one could be interested in the availability of professional services and training, in details of the license, etc. All these factors have to be 'squeezed' into the five aspects defined in the model.

The Open Business Readiness Rating (OpenBRR) (Wasserman et al., 2006) is an OSS evaluation method aiming at providing software professionals with an index applicable to all the current OSS development initiatives, reflecting the points of view of large organizations, SMEs, universities, private users, etc. The OpenBRR is a relevant step forward with respect to the OSMM, since it includes more indicators, the idea of the target usage, and the possibility to customize evaluations performed by other, just by providing personalized weights. With respect to the latter characteristics, the OpenBRR has however some limits: one is that for many products it is difficult to choose a "reference application" that reflects the needs of the users; another is that there are lots of possible target usages, each with its own requirements; finally, the evaluation performed by a user could be not applicable to other users. In any case, the final score is a synthetic indicator to represent the complex set of qualities of a software product. On the official OpenBRR site several evaluations were available, and originally provided as spreadsheet. However the OpenBRR website and tools are not available anymore.

The Qualification and Selection of Open Source Software (QSOS) (Semeteys, 2008) works similarly as OpenBRR, but requires first to create an Identity Card (IC) of each project, reporting general information (name of the product, release date, type of application, description, type of license, project URL, compatible OS, …), then to evaluate the available services, functional and technical specifications and grade them (in the 0..2 range). Then, evaluators can specify the importance of the criteria and their constraints. Finally a normalized score is computed to compare the selected project candidates. Although the method is effectively applicable to most OSS, the QSOS approach does not represent a relevant step forward with respect to other evaluation methods. Its main contribution is the set of characteristics explicitly stated which compose the IC, and the provision of a guideline for the consistent evaluation of these characteristics. The evaluation procedure is rigid. For instance, it requires to define the IC of each OSS under evaluation, even if they are not completely matching the requirements. Such a procedure is justified when the ICs of products are available from the OS community before a user begins the evaluation. However even in this case it may happen that the user needs to consider aspects not included in the IC: this greatly decreases the utility of ready-to-use ICs. The strict guidelines for the evaluation of the IC, necessary to make other users' scoring reusable, can be ill suited for a specific product or user. Finally, even though in the selection criteria it is possible to classify requirements as needed or optional, there is no proper weighting of features with respect to the intended usage of the software.

OpenBQR (Taibi et al., 2007a) works in a similar way as OpenBRR, but requires the evaluators to first specify the importance of the factors, and then to assess the projects, so as to avoid to invest time evaluating factors that are not relevant for the specific context. OpenBQR is an important step forward in terms of effort required to evaluate the projects.

OSS-PAL (Wasserman et al., 2017) works similarly as QSOS, but proposed to introduce a semi-automated evaluation, supported by an online portal. Unfortunately, the portal seems to be only a research prototype, and does not collect any data automatically.

All the aforementioned models have some drawbacks:

- Existing methods usually focus on specific aspects of OSS. For example, the OSMM focuses on software maturity, but misses some potentially interesting characteristics like license compliance or security for the quality assessment. On the other hand, methods like OpenBRR, QSOS, OpenBQR, etc. provide a set of indicators reflecting a wide range of potential users' viewpoints for the quality assessment. This requires individuals to identify the importance of assessment factors according to their needs and introduces extra effort and complexity to adopt a method for practice.
- The OSS evaluation requires effort to run the software and to extract information from the OSS portals. The assessment process of existing methods is not optimized. Methods such as QSOS proceed to evaluate indicators before they are weighted, so some factors may be measured or assessed even if they are later given a very low weight or even a null one. This results in unnecessary waste of time and effort.
- The dependence of the users of OSS is not adequately assessed, especially the availability of support over time and the cost of proprietary modules developed by third parties.

2.3. OSS aggregators

Many platforms have been developed to collect and share OSS-related data, enabling a quick extraction of the information on different OSS projects.

Ohloh was one of the first project aggregators (Bruntink, 2014; Allen et al., 2009) on the market (2004) aimed at indexing several projects from different platforms (GitHub, SurceForge, …). In 2009, Ohloh was acquired by Geeknet, owners of SourceForge (Anon, 2020e) that then sold it to Black Duck Software in

2010. Black Duck, was already developing a product for OSS audit, with a particular focus on the analysis of the license compatibility, and integrated Ohloh's functionality with their products. In 2014, Ohloh became "Black Duck Open Hub" (Anon, 2020f). Finally, Synopsys acquired Black Duck and renamed the Black Duck Open Hub into "Synopsys Open Hub". Synopsis Open Hub is currently the only continuously updated OSS aggregator that include information of different OSS projects from different sources (versioning control systems, issue tracking systems, vulnerability databases). On January 2021, OpenHub indexed nearly 500K projects, and more than 30 billions of lines of code. It provides flexibility for users to select the metrics to compare project statistics, languages, repositories, etc. However, it lacks the OSS evaluation facilities that allow to adjust the importance of selected metrics according to users' needs for automatically scoring the candidate software. In addition, it lacks information related to the community popularity, documentation, availability of questions and answers and other information.

Other OSS aggregators have been proposed so far. FlossHub (Anon, 2020g) and FLOSSMole (Anon, 2020a) had similar goal of OpenHub. However, they have not been updated in the last years. FlossMetrics (Anon, 2020b) had the goal of providing software metrics on a set of OSS. However, it has also been abandoned in 2010.

The Software Heritage (Di Cosmo and Zacchiroli, 2017), differently than the previously mentioned platforms, has the goal of collecting and preserving the history of software projects, and is not meant to enable the comparison or to provide support for selecting OSS. The project is sponsored by different companies and foundations, including the UNESCO foundation. The Software Heritage could be used as a source of information to analyze the activity of a project. However, its access is not immediate, and users need to use APIs to get detailed data on the projects.

Other platforms, designed for supporting mining software activities, might also be used for obtaining relevant information from OSS. In particular, the Sourceforge research data archives (Madey, 2008) shared the SourceForge.net data with academic researchers studying the OSS software phenomenon; GH Archive (Anon, 2020c) records the public GitHub timeline and makes it accessible for further analysis; and the GHTorrent project (Gousios, 2013b) creates a mirror of data offered through the Github REST API to monitor the event timeline, the event contents, and the dependencies.

2.4. OSS repository mining tools

Besides the platforms that aggregate heterogeneous metric providers to track repositories associated with a wide range of OSS projects, there are also research prototypes or projects to mine information from given repositories. In particular, BOA (Dyer et al., 2013, 2015) provide support to mine source code and development history from project repositories using the domain-specific language. Candoia (Anon, 2020h) also provided a platform for mining and sharing information from OSS projects. RepoGrams (Rozenberg et al., 2016; Anon, 2020i) allows to visually compare projects based on the history of the activity of their git repositories.

Other groups developed tools not aimed at supporting the selection of OSS, but that can be used as valuable sources of information. As an example, PyDriller (Spadini et al., 2018) can be used to obtain detailed information from commits.

Surprisingly, none of the previously mentioned papers cited other tools such as Cauldron or SourceCred. Cauldron[3] is a free open source software that is used to collect information from multiple sources as different information are retrieved. SourceCred[4] is an OSS technology which analyzes a project and determines the contributions of individuals in it. It is built on the idea that communities matters but also that the work of singles need to be visible and rewardables.

The Community Health Analytics Open Source Software (CHAOSS)[5] project. CHAOSS, a Linux Foundation project, also developed tools to measure OSS projects, and in particular to measure community health, to analyze software community development and to develop programs for the deployment of metrics not attainable through online trace data.

Different European projects also developed tools for mining data from OSS repositories. The EU H2020 CROSSMINER project[6] (Rocco et al., 2021) includes techniques and tools for extracting knowledge from existing open source components generating relevant recommendations for the development of user's projects. The recommendation system focuses on 4 main activities:

- **Data Preprocessing:** containing tools to extract metadata from repositories
- **Capturing Context:** uses metadata to generate knowledge for mining functionalities
- **Producing Recommendation:** IDE to generate recommendations.
- **Presenting Recommendation:** IDE to show recommendations.

QUALOSS (Quality in Open Source Software) (Soto and Ciolkowski, 2009) and QualiSPo (Quality Platform for Open Source Software) (Del Bianco et al., 2010) projects aimed at identifying quality models to evaluate the quality and the trustworthiness of OSS. Both projects proposed different tools for extracting data from repositories, to calculate software metrics and to identify possible issues in the code or in the community activity. However, none of the tools developed is currently active, and several of them are not available anymore (e.g., QualiSPo Del Bianco et al., 2010)

2.5. OSS audit and analysis tools

Companies like Synopsys (formerly BlackDuck) (Anon, 2020j) and WhiteSource (Anon, 2020k) provide solutions to software composition analysis and offer services of the assessment of OSS quality and code security. Synopsis focuses on their professional services of the license compatibility while WhiteSource emphasizes the open source management to offer services such as viewing the state of OSS components, their license compliance, and the dependencies; prioritizing components' vulnerabilities based on how the proprietary code is utilizing them; analyzing the impact of the vulnerabilities, etc.

Different tools to assess specific qualities are also available on the market. As an example, companies can use tools such as SonarQube (Anon, 2020l) or Sonatype (Anon, 2020m) to evaluate different code-related qualities such as the standard compliance or the technical debt. Or Security-specific tools such a White-Hat Security (Anon, 2020n), Kiuwan (Anon, 2020o) or others to evaluate the security vulnerabilities.

[3] **Cauldron:** https://cauldron.io.

[4] **SourceCred:** https://sourcecred.io/docs/.

[5] **CHAOSS project:** https://chaoss.community/about/.

[6] **CROSSMINER project:** https://www.crossminer.org.

2.6. Gaps of the current OSS assessment models

The different OSS assessment models, tools and platforms provide a possibility to assess OSS projects mainly from the perspectives of license obligation, application security, code quality, etc. They are about the state of software and its quality and comprise an essential part of the assessment model for OSS selection and adoption (Sbai et al., 2018; Lenarduzzi et al., 2020). Besides, activities, supports, or other projects surrounding a project form an important perspective demonstrating if a project exists in a lively ecosystem (Jansen, 2014). In particular, metrics such as response times in Q&A forums and bug trackers, the active contributors and their satisfaction, the user's usage and their satisfaction, the number of downloads, the number of forks, bug-fix time, etc. are informative references indicating the productivity and a propensity for growth of the OSS project community. Some of the measures can be cross-referenced from different data sources, while some need further analysis based on the collected data. To the best of our knowledge, no portal has effectively taken these community-related factors into account when providing service to evaluate and compare OSS projects.

Furthermore, companies have their distinct strategies, needs, and constraints to adopt OSS projects in software development (Lenarduzzi et al., 2020). After practitioners identify a list of candidates that cover the expected features, meet requirements, and fit with the existing technical solution, they specify the importance of the selection criteria , complying with the company's needs and restrictions. As highlighted in our previous systematic literature review (Lenarduzzi et al., 2020), it is impractical for companies to study every software assessment model to select the one that fits their needs best. Therefore, there is a need to call for the OSS evaluation and selection tools that not only guides developers to adapt OSS assessment criteria by identifying and weighing the ones fitting in a specific scenario, but also automates the process of assessing and comparing among a set of selected software based on information which can be extracted from the public APIs of available portals.

3. Research method

3.1. Goal and research questions

Our goal is to investigate and determine the factors that practitioners are currently considering when selecting OSS, to identify the sources (portals) that can be used to evaluate such factors mentioned by the practitioners, and to validate the public APIs that can be accessed to automatically evaluate those factors from the sources and portals.

To achieve the aforementioned goals, we defined four main research questions(RQs).

RQ1. What factors are practitioners considering when selecting OSS projects to be integrated in the software they develop?
In this RQ we aim at collecting the information adopted by practitioners when selecting projects to be integrated in the software they develop. We are not considering OSS products supporting software development process and the management such as IDEs, Office Suites, but software libraries, frameworks or any other tool that will be integrated and packaged as part of the product developed by the company.

RQ2 Which metrics are used by practitioners to evaluate the factors adopted during the selection of OSS?
In this RQ we aim at identifying the metrics adopted by practitioners to evaluate the factors they are interested to assess. As an example, practitioners might assess the size of the community checking the number of committer in the repository, or might check the size of the project by checking the number of commits in the repository or even downloading the software and measuring its size in lines of code.

RQ3 Which source of information and portals are used to assess OSS?
In this RQ we aim at understanding which portals or other sources are used by practitioners to evaluate the factors identified in RQ1, based on the metrics reported in RQ2.

RQ4 Which factor can be extracted automatically from OSS portals?
In this RQ, we aim to systematically analyze the common portals hosting OSS, to identify the information that can be extracted via APIs.

In order to answer our RQs, we conducted our work in three main steps:

Step 1: Interviews among experienced software developers and project managers to elicit the factors affecting the OSS selection (RQ1), the metrics (RQ2) and the sources of information they adopt (RQ3).

Step 2: Analysis of the APIs of the source of information (portals) identified in RQ3.

Step 3: Analysis of the availability of the metrics collected in the previous step (RQ2) in the public API of the sources of information adopted by practitioners (RQ3) among 100k projects (RQ4).

Fig. 1 depicts the process adopted in this work. The detailed process is reported in the remainder of this section.

3.2. Step 1: Interviews on the factors considered when selecting OSS

In order to elicit the factors adopted by practitioners when selecting an OSS in the software product development process, we designed and conducted a semi-structured interview based on a questionnaire.

3.2.1. The interview population

We identified the population for our interviews considering participants who can best provide the information needed in order to answer our RQs. We selected participants that fulfilled the following criteria:

- Currently developing software projects. With this criteria, we aim at selecting participants that are still working on software projects. This criteria will exclude persons that had a long experience but are not working anymore in software development projects (e.g. upper managers)
- At least 5 years of experience in developing software projects. We aim at including only practitioners with a minimum level of experience, excluding freshman and newly graduated ones.
- At least 3 years of experience in the domain they are working. We want to consider only practitioners that have a minimum level of experience in the domain they are working, to avoid incongruences due to the lack of knowledge of the domain.
- At least 3 years of experience in deciding which OSS component integrate in the product they develop.

3.2.2. The questionnaire

The interviews were based on the same questionnaire adopted in our previous works to elicit the factors considered important for evaluating OSS (Del Bianco et al., 2009; Taibi, 2015). We organized the questions in the questionnaire adopted for the interviews two sections, according to the types of information we sought to collect:

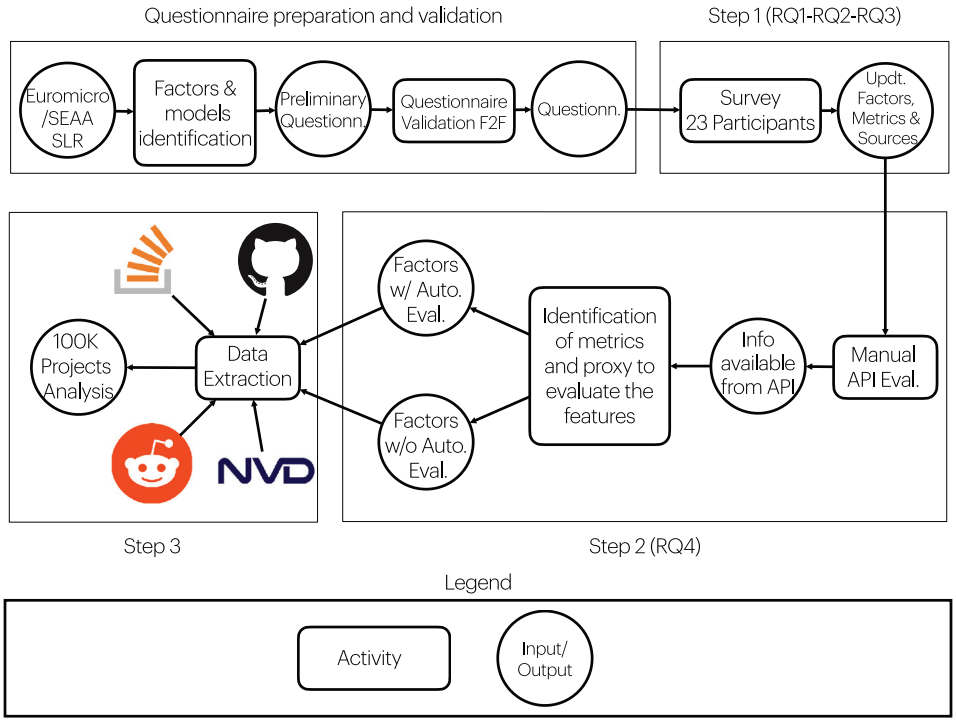


Fig. 1. The study process.

- ***Demographic information***: In order to define the respondents' profile, we collected demographic background information in relation to OSS, including the number of years of experience in selecting OSS components to be integrated in the software they develop. This information considered predominant roles and relative experience. We also collected company information such as application domain, organization's size via number of employees, and number of employees in the respondents' own team.
- ***Factors considered during the adoption of OSS***: Here we asked to list and rank the factors considered during the adoption of OSS software to be integrated in the products they develop, based on their importance, on a 0-to-5 scale, where 0 meant "totally irrelevant" and 5 meant "fundamental".
  - We first asked to list the factors the respondents consider when adopting an OSS in the software product they develop, and to rank the them on the 0-to-5 scale. This open question is to encourage respondents to identify the important factors which might not be clarified in our Euromicro/SEAA SLR (Lenarduzzi et al., 2020).
  - Then, we asked to rank other possible factors not mentioned in the previous step, on the 0-to −5 scale. Please note that the interviewer listed the remaining factors identified in our Euromicro/SEAA SLR and not mentioned by the participant (Lenarduzzi et al., 2020). The factors identified in the Euromicro/SEAA SLR are reported below.
    - Community & Support
    - Documentation
    - Economic
    - License
    - Operational SW Characteristics
    - Maturity
    - Quality
    - Risk
    - Trustworthiness
  - For each factor ranked higher or equal than 3, we asked to:
    - Report the related sub-factors and their associated metrics with the importance ranking on the 0-to −5 scale
    - Report the source they commonly use to evaluate them (e.g. GitHub, Jira, manual inspection, …)
    - Report the metrics they adopt to measure the factor
  - We finally asked if they think the factors they reported enable a reasoned selection of OSS or if they would still need some piece of information to have a complete picture of the assessment.

The complete questionnaire adopted in the interviews is reported in the replication package (Anon, 2021).

3.2.3. Interviews execution

The interviews were conducted online, using different video-conferencing tools (Zoom, Skype and Microsoft Teams), based on the tool preferred by the interviewed participant. Interviews were carried out from September 2020 to December 2020.

Because of time constraint, and of the impossibility to conduct face-to-face interviews during public events, interviewees were selected using a convenience sampling approach (also known as Haphazard Sampling or Accidental Sampling) (Battaglia, 2008). However, we tried to maximize the diversity of the interviewees, inviting an equal number of developers from large and medium companies, and from companies in different domains. The selected participants are experienced developers or project managers, and have been involved in the OSS selection process or the software integration and configuration management process. We did not consider any profiles coming from academia, such as researchers or students, nor any inexperienced or junior profiles.

3.2.4. Interviews data analysis

Nominal data on the factor importance is determined by the proportion of responses in the according category. In order to avoid bias, the interviewees are asked to recall the important factors without being provided with options. Thus, the proportion of interviewees who mentioning a factor shall reflect its importance fairly. Ordinal data, such as 5-point Likert scales, was not converted into numerical equivalents to prevent the risk of misleading to subsequent analysis. Apparently when the deviation of the responses is large, such a phenomenon will be overlooked. In this way, we can better identify the potential distribution of the interviewees' responses.

Open questions (application domain, other factors reported, platforms adopted to extract the information and metrics adopted to evaluate the factors) were analyzed via open and selective coding (Wuetherick, 2010). The answers were interpreted by extracting concrete sets of similar answers and grouping them based on their perceived similarity. Two authors manually provided a hierarchical set of codes from all the transcribed answers, applying the open coding methodology (Wuetherick, 2010). The authors discussed and resolved coding discrepancies and then applied the axial coding methodology (Wuetherick, 2010).

3.3. Step 2: Analysis of the APIs of the OSS portals

We manually analyzed the APIs of the portals identified in RQ3, looking for APIs that allowed to assess the information needed to measure the factors reported by the interviewees (RQ1 and RQ2). The first two authors independently analyzed all the portals seeking for these pieces of information, and then compared the results obtained. In case of discrepancies, all the incongruities were discussed by all the authors, reaching a 100% consensus.

Some factors were not directly analyzable. For example, the responsiveness of an OSS community cannot be directly measured; hence, a proxy metric, i.e., the average time spans between the created time of issues and the first actions, is adopted. Therefore, the first two authors proposed a list of proxy metrics, considering both the metrics adopted by the interviewees and metrics available in the literature. Then, all metrics were discussed by all the authors until we reach a consensus.

However, as expected, not all the metrics can be automatically extracted, and some of them require a manual assessment. An example of a factor that cannot be automatically extracted is the availability of complete and updated architectural documentation.

3.4. Step 3: Analysis of projects that provide information to assess the factors

3.4.1. Validation of the factors analyzability on the OSS portals

This step was based on three sub-steps:

- *Project selection*. We selected the top 100K GitHub projects, based on the number of stars. The list of selected projects were determined on 2020-11-10. The number of projects was limited to the time available. In particular, the different APIs limit the number of queries that can be executed in one hour, and therefore we limited the study to 100k most starred projects to ensure that the data can be extracted in 2 months. We are aware that some projects might not be code-based projects, and some repositories might only have the purpose to collect resources. However, since it is not possible to automatically exclude non-code projects, we consider them all.
- *Information extraction*. We extracted the selected information from the APIs. We decided to extract only the information needed to evaluate the factors. Other information are available, but requires to run a higher number of queries, and therefore would have reduced the number of projects that we can extract. The extraction process started on 2020-11-16 with data collected gradually till 2020-12-29. As an example, it would be possible to extract all the details on project issues (issue title, author, date, comments, …), but this would have required to run a number of additional queries, without providing any information considered valuable by our interviewees.
- *Analysis of the information available*. In this step we analyzed which information is actually available for each project. As an example, not all the projects might use different issue trackers instead of using the one provided by GitHub, or some projects might not be listed by the NIST NVD database, or more, some project might not have questions and answers available in StackOverflow or Reddit.

3.5. Replication

In order to ease the replication of this work we provide the complete replication package including the questionnaire adopted for the interviews, the results obtained in the interviews, the data crawling script and the results of the data analysis (Anon, 2021).

4. Results

Here we first provide information about the sample of respondents, which can be used to better interpret the results and then, we show the collected results with a concise analysis of the responses obtained, with insights gained by statistical analysis.

We collected 23 interviews from experienced practitioners. Fig. 2 contains the distribution of company sizes where our interviewees belong, Fig. 3 shows the percentage for organizational roles identified in the questionnaire while Fig. 4 shows the distribution of the experience of our interviewees in selecting OSS components to be integrated in the projects they develop.

4.1. RQ1: Factors considered by practitioners when selecting OSS

Our interviewees consider 8 main factors and 46 sub-factors when they select OSS, reporting an average of 2.35 factors per interviewee, a minimum of 1 and a maximum of 21 factors.

The factor that is mentioned more frequently from the interviewees is *License* which has received a median importance of 4 out of 5. Surprisingly, this is not the value which has received the highest median value of importance as *Community Support and Adoption, Performances* and *Perceived Risk* received a median value of importance of 4.5 out of 5. It is interesting to note that no participant mentioned *economic* and its related sub-factors such as license costs, or cost for training. So this factor is not reported in our results.

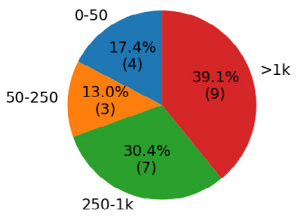
In Table 2, we report the list of factors and sub-factors together with the number of participants who mentioned them (column RQ1- #) and the median of the importance reported by the interviewees (column RQ1 - Median).

4.2. RQ2: Metrics used by practitioners to evaluate factors during OSS selection

When we asked practitioners to report the metrics they use to evaluate the factors they mentioned in RQ1, and to rank their usefulness, practitioners mentioned 110 different metrics.

Company Size (# Employees)	# Interviewees
0-50	4
50-250	3
250-1000	7
>1000	9

(a)

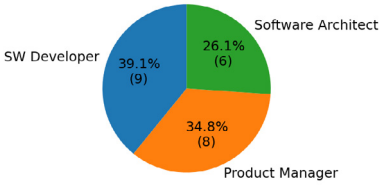


(b) Distribution of company size

Fig. 2. Distribution of company sizes of our interviewees.

Roles	# Interviewees
SW developer	9
Product Manager	8
Software Architect	6

(a)



(b) The roles of our interviewees

Fig. 3. Distribution of roles of our interviewees.

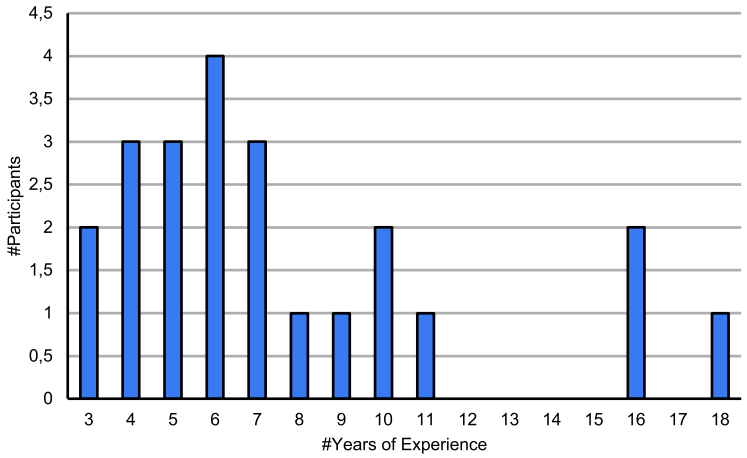


Fig. 4. Number of years of experience of the participants in selecting OSS components to be integrated in the products they develop.

The complete list of metrics reported for each factor is reported in Appendix.

In Table 2 (Column RQ2 - #Metrics) we report the count of metrics considered as useful by practitioners (likert scale $\geq$ 3, where 0 means "This metric is useless to evaluate this factor" and 5 means that the metric is extremely useful).

Surprisingly, the factor where practitioners provided the highest number of metrics to assess it, is *Maturity*, which has been mentioned only 6 times compared to the *License*, mentioned 21 times, where practitioners provided 7 metrics instead. This indicates a wide variety of interpretations on *Maturity*, and practitioners use different metrics to evaluate this factor. The careful reader can also observe that for some factors considered as relevant in RQ1 such as *Perceived risks*, no metrics have been mentioned. This result proves that in some cases, some of the most important factors in an OSS cannot be objectively measured and the interviewees do not know how to retrieve such information appropriately.

4.3. RQ3: Sources of information and portals used to assess OSS

Our interviewees mentioned 9 different source of information and portals they commonly consider when they select OSS.

In Table 3 we list the sources of information adopted by the practitioners to evaluate OSS, together with the number of participants who mentioned it (columns # and % of mentions). To increase the readability, we grouped the source of information in five main categories: *version control systems*, *issue tracking systems*, *Question and Answer portals* (Q&A), *forum and blogs* and *security* related platforms.

The 5 most reported sources of information are GitHub (and GitHub Issue tracker), StackOverflow, Reddit, and NIST Security Vulnerability (NVD) with respectively: 23, 19, 12, 12 and 14 mentions. All of these are mentioned from more than 20% of the interviewees and are therefore those which prove to be the most useful when retrieving information related to OSS. Other platforms such as Bitbucket or Jira were rarely mentioned. The results presented in Table 3 could also be useful to other OSS stakeholders such as software administrators and software operators.

4.4. RQ4: Factors that can be extracted automatically from the portals

We focused our attention to the four most used sources of information reported in RQ3: GitHub (Anon, 2020v), StackOverflow (Anon, 2020w), Reddit (Anon, 2020u) and the NIST National

Table 2
High-level Factors considered during the adoption of OSS.

RQ1			RQ2
Factor	#	Median	#Metrics
Community support and adoption	10	4.5	
Popularity	9	3	4
Community reputation	11	3	3
Community size	13	3	5
Communication	6	3.5	5
Involvement	9	3	1
Sustainability	11	3	1
Product Team	5	3	2
Responsiveness	1	5	1
Documentation	14	4	
Usage documentation	4	4	5
Software requirements	11	3	1
Hardware requirements	8	3.5	1
Software Quality Documentation	5	3	3
License	21	4	7
Operational SW characteristics	6	4	
Trialability	5	3	2
Independence from other SW	11	3	4
Development language	5	4	3
Portability	1	4	1
Standard compliance	5	4	0
Testability	6	3.5	0
Maturity	6	3.5	11
Quality	6	3.5	
Reliability	3	4	6
Performances	4	4.5	1
Security	15	4	6
Modularity	3	3	1
Portability	3	4	2
Flexibility/Exploitability	3	3	3
Code Quality	13	4	6
Coding conventions	9	3	0
Maintainability	3	4	0
Testability	2	4	0
Existence of benchmark/test	4	3.5	4
Changeability	2	3.5	0
Update/Upgrade/Add-ons/Plugin	3	4	1
Architectural quality	5	3	0
Risk (Perceived risks)	7	4.5	
Perceived lack of confidentiality	5	1	0
Perceived lack of integrity	5	3	0
Perceived high availability	5	4	3
Perceived high structural assurance	5	2	0
Strategic risks	5	3	0
Operational risks	5	1	1
Financial risks	5	2	0
Hazard risks	5	4	5
Trustworthiness	6	4	
Component	4	3.5	3
Architecture	4	3	2
System	4	3.5	3
OSS provider reputation	4	3.5	0
Collaboration with other product	4	2.5	3
Assessment results from 3rd parties	2	3.75	0

Table 3
The source of information reported by the interviewees (RQ3).

ID	Source of information	#	% of mentions
	Version control systems:		
R	GitHub Anon (2020v)	23	100
R	GitLab Battaglia (2008)	1	4.3
R	SourceForge Wuetherick (2010)	1	4.3
R	Bitbucket Anon (2020v)	1	4.3
	Issue tracking systems:		
I	GitHub Issues Anon (2020v)	19	82.6
I	Jira Anon (2020w)	1	4.3
	Question and answer portals:		
Q	StackOverflow Anon (2020w)	12	51.2
Q	Reddit Anon (2020u)	12	51.2
	Forum and blogs:		
F	Medium Cai and Zhu (2016)	5	5
F	Hackernews Hu et al. (2012)	5	5
	Security:		
S	CVE Anon (2020y)	1	4.3
S	CVSS Anon (2020y)	2	8.7
S	CWE Anon (2020z)	1	4.3
S	NVD Anon (2020x)	14	21.7

Vulnerability Database (Anon, 2020x). For such purpose, we first identified the APIs that can be adopted to extract the information needed to measure the factors, and then we extracted the data from the 100k with more stars in GitHub.

The extraction of the information for 100K projects took a total of 5 days for GitHub, 53 days for StackOverflow, 4 days for Reddit and 2 Days for the NIST National Vulnerability Database. The long processing time is due to the limit of queries that can be performed on the APIs for different IP addresses.

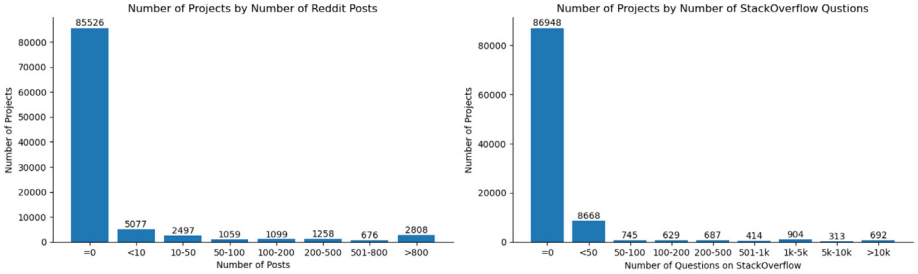
Considering the projects extracted (Fig. 6(a)), more than half of these projects (53.3%) have been active for 2–6 years. Around 12.1% of these projects are active for one year or less when only 3.9% of them are active for more than 10 years. Majority of these projects (87.0%) have less than 500 issues during their life cycle when around 3.4% of them have more than 1k issues (Fig. 6(b)). Furthermore, there are 1791 projects being very popular having more than 10k stars when 25.9% projects having stars ranging from 1k to 10k (Fig. 6(c)) with 46.7% having less than 500 stars. On the other hand, regarding project size, more than half of them (52.8%) have lines of code (LOC) ranging from 20k to 500k. 6.1% projects contain more than 5 m LOC when only 0.9% of them have less than 1k LOC (Fig. 6(d)). Regarding developing languages, Javascript is the most popular being the primary language of OSS projects (17k projects) with Python and Java at 2nd and 3rd (Fig. 6(e)). They are the primary languages for 40.9% of the projects. However, regarding LOC by languages, C language (57.6b) and Javascript (57.2b) rank at the top. Both have almost doubled the amount of C++ language (31.7b) which ranks the 3rd in terms of total LOC (Fig. 6(f)). Regarding release numbers, 58.1% projects do not have any specific release recorded. 37.6% have less than 50 releases when only the rest 4.3% have more than 50 releases. All the previously mentioned data is always available from GitHub, and queries to the GitHub APIs will always return a valid information.

Regarding the adopted open source licenses, 23.7% projects did not specify the licenses they adopt. As for the other projects, MIT, Apache 2.0 and GNU GPL v3.0 are the most popular licenses with 53.2% projects adopting one of them (Fig. 6(h)). Therein, 13.6% projects adopted non-mainstream license (identified as 'Other').

We also validate the APIs of Reddit and StackOverflow by finding the amount of discussion threads on each of the 100k OSS projects. As shown in Fig. 5, 14.5% projects are generally discussed in Reddit with only 5.8% projects having more than 100 posts (Fig. 5(a)). On the other hand, 13.0% projects are discussed (raised technical questions) in StackOverflow (Fig. 5(b)). Therein, 3.6% projects have more than 100 questions raised on. In general, such results show that it is hard to find sufficient generic or technical discussion regarding specific OS projects from Reddit and StackOverflow.

Based on the interview results, especially the obtained factors that are considered important by the practitioners (shown in Table 2), we further validate whether such factors can be analyzed via the automatically obtained data from the previously mentioned portals with the results shown in Table 4.

According to the investigation on automatic OSS data extraction on the Top 100k OSS projects on Github via the APIs of



(a) OSS Project Stats by Reddit Post Number (b) OSS Project Stats by StackOverflow Questions

Fig. 5. Stats for Top 100k Github Projects in Reddit and StackOverflow.

the five public portals, we find that the majority of the metrics towards the *Community Support and Adoption* factor can be automatically done via data extractions from such APIs. To be noted, regarding *Communication*, the question and answer sources (i.e., StackOverflow) contain information on limited number of OSS projects, which limits the availability towards the evaluation on such category. In addition, we also find, despite the availability of automatic data extraction and measuring via APIs, many of the metrics require further calculation and learning, as well as multiple queries to obtain. For example, in order to measure the *Number of Independent Developers*, we must get the list of contributors of a particular project via multiple queries first (max items per page for Github API is 100), and check the "Independentness" of each contributor via further investigating his/her organization status. Thus, such process shall be, to some extent, time-consuming, when the limit rate of the API usage shall be also taken into account.

Another category can be automatically measured is *License*, as shown in the data, 76.6% of the projects contain specific License information. On the other hand, for the other categories, automatic data extraction and full-grained evaluation is hindered by the limited availability of the according data, as only very limited percentage of the metrics can be automatically done via data extraction (shown in Table 2). And amongst these categories, the evaluation of *Maturity* category depends on the availability of the release data from repository dataset, when for the obtained 100k projects only 42.2% provide such information. Measuring the availability of *Documentation* shall also depend on the information extracted from project description and homepage, while only around 40% of projects provide those.

As for the security vulnerability evaluation, the NVD Dataset provide information for 12838 projects (12.8%). However, it is not clear if the projects not reported in NVD do not contain security vulnerabilities at all, or simply are not indexed by the NVD dataset. However, it is important to note that the NVD performs analysis on CVEs published to the CVE Dictionary. Every CVE has a CVE ID which is used by cybersecurity product/service vendors and researchers for identifying vulnerabilities and for cross-linking with other repositories that also use the CVE ID. However, it is possible that some security vulnerabilities are not publicly reported with assigned CVE IDs.

As shown in Table 4, amongst the 170 metrics identified, 40 of them are potentially available to be extracted automatically. In addition, *License type* and *Development Language*, though seen as sub-factors, can be automatically measured as well. Therein, the number of automatic measurable metrics for all 100k projects (#full-auto) and that for part of them (#part-auto) are also shown. Only 22 metrics out of 170 can be obtained automatically for all 100k projects when the others are only available for part of the projects. In addition, 22 metrics require multi-queries to complete when 21 require further calculation and/or learning to determine.

5. Discussion

In this Section, we discuss the results of our RQs and we present the threats to validity of our work.

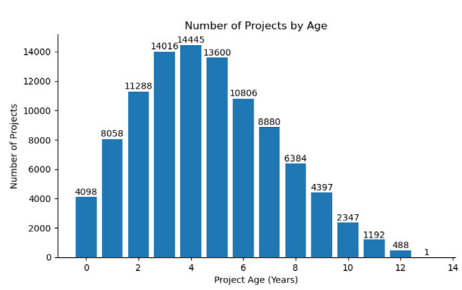
The factors and the metrics adopted to evaluate and select OSS (RQ1–RQ2) evolved over time. While in 2015 (Taibi, 2015; Lenarduzzi et al., 2020) factors such as *Customization easiness* and *Ethic* were the most important, nowadays we cannot state the same. Already in 2020 (Lenarduzzi et al., 2020) such factors have been incorporated inside other more valuable factors such as *Quality*, while today are not mentioned anymore. On the other side *License* and *Documentation*, which are the most mentioned factors nowadays were side factors in 2020. As a matter of fact the latter was a sub-section of *Development Process*, while both were not even considered in 2015. Moreover, ethical principles, that were very relevant in 2015, are not even mentioned in 2020.

Nowadays the trend is to search for OSSs which are ready to be integrated as is. In order to incorporate OSSs without falling into lawsuit particular attention needs to be put into the *License type*, while to guarantee the correct functioning the focus needs to be put in the documentation. A clear example is the necessity of a clear definition of the *system requirements* when incorporating libraries.

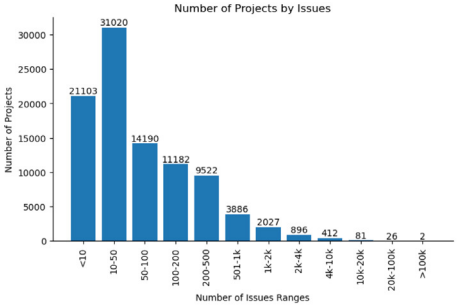
Another factor which gained a lot of importance over time is *Security*. While in 2015 was a factor with medium relevance, nowadays it is a keypoint for measuring quality of an OSS. The growing number of portals dedicated to ensure absence of vulnerabilities proves that people are concerned of the use of OSSs when embedded in their system. In particular they strongly rely on such portals to check the history of the OSSs to incorporate and in some cases also to ensure that proper reports are delivered when a new vulnerability is discovered.

Also, the importance of a factor is not necessarily proportional to the number of metrics. When specifying the importance of the assessment factors, we may see that some measurable factors are perhaps just eliminated, while some important factors cannot be automatically evaluated using the extracted information and require a manual assessment. Moreover, it is important to note that the lack of concrete metrics for some factors, such as community reputation or community sustainability might be because these factors are too abstract. Some researchers already addressed some of these aspects. As an example, Cai and Zhu (2016) and Hu et al. (2012) already proposed some metrics to evaluate the community reputation while Gamalielsson and Lundell (2014) also identified approaches for contributing to the community sustainability. However, these models are not yet diffused in industry, and this might be the reason why our interviewees were not aware of them.

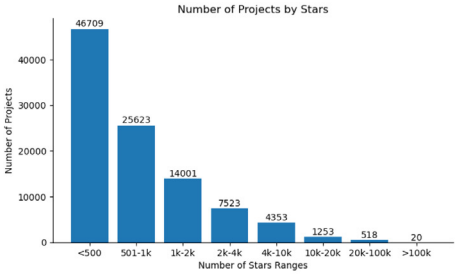
The source of information adopted to evaluate OSS (RQ3) did not change completely from the previous years. Users are



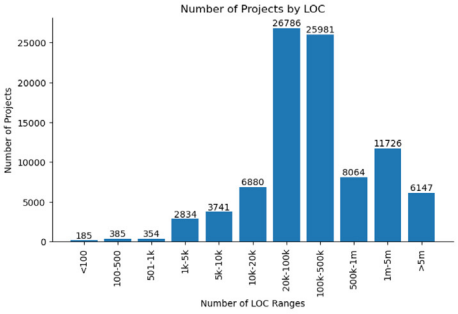
(a) OSS Project Stats by Age (Years)



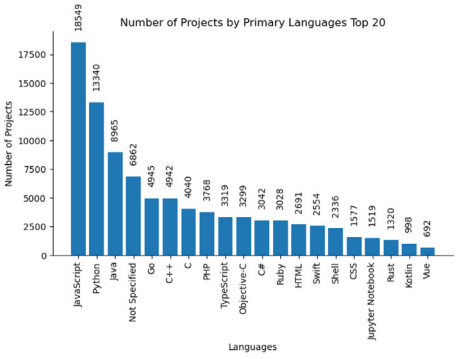
(b) OSS Project Stats by Issues



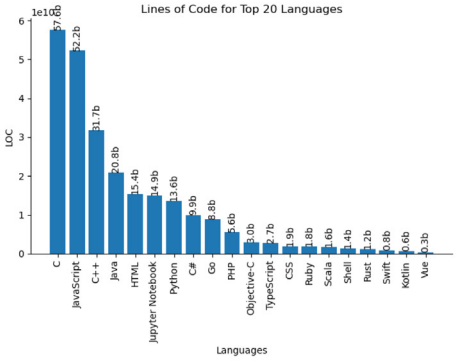
(c) OSS Project Stats by Stars



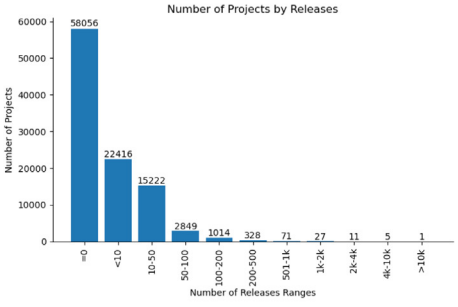
(d) OSS Project Stats by LOC



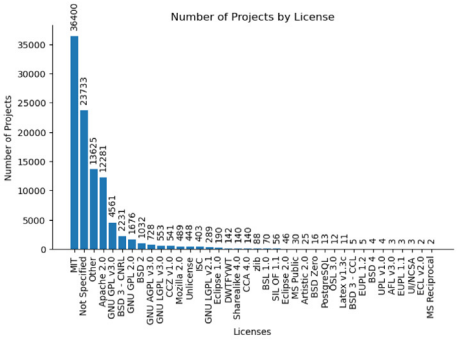
(e) OSS Project Stats by Top 20 Primary Languages



(f) Stats for Top 20 Language by LOC



(g) OSS Project Stats by Releases



(h) OSS Project Stats by Licenses

Fig. 6. General Stats for Top 100k Github Projects.

still adopting project repositories and issue trackers as main source of information. Moreover, an effect of the newly introduced factor security, is that now the selection also require security related information, that are commonly fetched from security databases such as the NIST NVD (Anon, 2020x), CVE (Anon, 2020y), and CWE (Anon, 2020z). In addition, many vendors like Synopsys (Anon, 2020j) and WhiteSource (Anon, 2020k) offer software composition analysis solutions that facilitate licence risk management, vulnerability identification and management, risk reporting, etc.

Unexpectedly, even nowadays, not all the portals can provide complete information for evaluating the information needed by the practitioners (RQ4). The analysis of the 100K most starred projects in GitHub showed that only the information coming from

Table 4
Factor information availability stats for top 100K projects.

Factor (#part-auto/#full-auto/#metrics)	Portal	#	%
Community Support and Adoption (5/16/41)			
Popularity	R		
Number of Watch	R	100000	100.00%
Number of Stars	R	100000	100.00%
Number of Forks	R	100000	100.00%
Number of Downloads	R	42260	42.26%
Community reputation	I+*		
Fast response to issues (△)	I+*	100000	100.00%
Community size	R+*		
Number of Contributors	R*	100000	100.00%
Number of Subscribers	R	100000	100.00%
Community age	R+	100000	100.00%
Number of Involved developers per company	R+*	100000	100.00%
Number of Independent developers	R+*	100000	100.00%
Community support	R+*, Q+*		
Activeness (△)	R+*	100000	100.00%
Responsiveness (△)	R+*, Q+*	14474	14.47%
Communication	I+*, Q+*		
Availability of questions/answers	I, Q	100000	100.00%
Availability of forum	Q	14474	14.47%
Responsiveness of postings	I+*, Q+*	14474	14.47%
Quality of postings (△)	Q+*	14474	14.47%
Sustainability	I+*		
Existence of maintainer	I+*	100000	100.00%
Product Team	R+*		
Developer quality (△)	R+*	100000	100.00%
Developer Productivity (△)	R+*	100000	100.00%
Responsiveness	I+*		
Avg. bug fixing time	I+*	100000	100.00%
Avg time to implement new issues	I+*	100000	100.00%
Documentation (3/0/12)			
Usage documentation	R*, F+		
Availability of updated documentation (△)	R*	42260	42.26%
Availability of documentation/books/online docs (△)	R	39499	39.50%
Availability of Tutorial or Examples (△)	R	39499	39.50%
License (1/0/9)	R		
License type	R	76576	76.58%
Operational SW Characteristics (1/0/9)	R		
Development language	R	93148	93.15%
Maturity (4/3/11)	R+*		
Number of forks	R	100000	100.00%
Release frequency	R+*	42260	42.26%
Number of releases	R	42260	42.26%
Age (in Years)	R+	100000	100.00%
Number of commits	R	100000	100.00%
Development versions	R*	42260	42.26%
New feature integration	R+*	42260	42.26%
Quality (5/3/47)			
Reliability	I+*		
Average bug age	I+*	100000	100.00%
Avg. bug fixing time	I+*	100000	100.00%
Security	R+*, Q+*, S+*		
Number security vulnerabilities	S	12838	12.84%
Number of Vulnerabilities reported on the NVD portal	S	12838	12.84%
Vulnerability Resolving time (△)	R+*, S+	12838	12.84%
Community concern towards security (△)	R+*, Q+*, S+*	12838	12.84%
Vulnerability impact (△)	S	12838	12.84%
Code Quality	R		
Code size	R	100000	100.00%
Trustworthiness (1/0/22)	R+*		
Consistent release updates pace (△)	R+*	42260	42.26%

Repository(R), Issue Tracker(I), Questions and Answer portals(Q), Forum & Blogs(F), Security(S)
Calculation Required(+), Multi-queries Required(*), Proxy Metrics(△).

the project repositories (e.g GitHub) are always available, except for the license information (76.6%). Considering other factors such as the communication, the situation does not improve, and only in 14.5% of projects we were able to automatically extract the relevant information from their APIs. For example, it is noticeable that the APIs of StackOverflow enable the extraction of other information, e.g., the textual content of questions and answers, the users' reputation, and so on. However, it requires further learning and calculation to elicit additional information from such textual content. The possibilities towards such directions shall be studied further in our future studies. Furthermore, some of the Github views, though providing valuable information but being inaccessible directly from APIs, (e.g., the GitHub insight view) are not covered herein. Due to the diversity of application domains, organizational needs, and constraints, practitioners in different organizations may explain the factors from their own perspective and may adopt different metrics in the OSS evaluation. A good example are the metrics associated with the factor of *Maturity*. Metrics such as the number of releases and the system growth in the roadmap were commonly concerned in the evaluation

of software maturity; besides, some practitioners also took the number of commits and the number of forks as important metric in the evaluation. The total number of commits itself might not be enough when evaluating software maturity. The prevalence of commits over time and the types of commits could be additional and useful information in the evaluation. Therefore, how the metrics help with the evaluation of the factors could have been clarified.

The results of this work show a discrepancy between the information required by the practitioners to evaluate OSS and the information actually available on the portals, confirming that the collection of the factors required to evaluate an OSS project is very time-consuming, mainly because most of the information required is not commonly available on the OSS project (Kamei et al., 2018; Lenarduzzi et al., 2020; Del Bianco et al., 2010).

The automation of the data extraction, using portal APIs might help practitioners reducing the collection time and the subjectivity. The result of this work could be highly beneficial for OSS producers, since they could check if they are providing all the information commonly required by who is evaluating their products, and maximize the likelihood of being selected. The result can also be useful to potential OSS adopters, who will speed-up the collection of the information needed for the evaluation of the product.

Even in case OSS producers do not enhance their portals by providing the information required by the practitioners to assess OSS, the results of this work could be useful for practitioners that need to evaluate an OSS product. The list of factors can be effectively used as checklist to verify if all the potentially important characteristics of OSS have been duly evaluated. For instance, a practitioner could have forgotten to evaluate the trend of the community activity and he/she could adopt an OSS product that has a "dissolving" community: this could create problems in the future because of the lack of maintenance and updates. The usage of checklist would allow practitioners to double check if they considered all factors, thus reducing the potential unexpected issues that could come up after the adoption.

5.1. Future research directions

As a result of our findings, we propose the following directions for future research in this area.

Focus on the definition of a common tool to automatically extract information needed for the evaluation of OSS, investigating proxy metrics in case direct metrics are not available.

Definition of refined and customizable models (which may be obtained by merging multiple available approaches) and favor its adoption through rigorous and extensive validation in industrial settings. This could increase the validity of the model and thus its dissemination in industry, where OSS is still not widely adopted. Several models already exist but, according to the results of our previous literature review (Lenarduzzi et al., 2020), they have not been strongly validated and, as a consequence, adoption has been limited.

Try to target the models at quality factors that are of real interest for stakeholders. Most of the available models focus on the overall quality of the product, but few of them are able to adequately assess each single factor that composes the overall quality of the OSS product. This can complicate the assessment of OSS products by stakeholders, who are interested in specific quality factors: e.g., developers are likely more interested in reliability or testability aspects, while business people may be more interested in cost or maintenance factors, etc..

In the studies, we identified 170 metrics to measure the factors for OSS evaluation and selection, based on which we shall conduct an in-depth analysis to gain a better understanding of the rationale for the metrics. The rationale explains why a metric helps gain insights into the factors, and the assumption or other information useful in evaluating an OSS. It helps to identify the needed data to extract from the available portals and to automate the OSS analysis and assessment process. With the explanation of why the metrics are needed, practitioners can also better understand the factors and their assessment, which further eases the process to adopt the OSS selection models and tools in the software development practice.

Furthermore, besides the common evaluation metrics identified in this study that suits targeting any OSS, it is noticeable that the domain fitness of such targeting OSS is also of great importance. Though this study focuses on the general quality attributes of OSS, the assessment of domain fitness should be taken into account with the domain-fitting OSS candidates limited so that the evaluation effort can be largely reduced.

Develop tools that support the research directions listed above (i.e., tools able to support and simplify the applicability of the proposed models during the evaluation of OSS products).

Disseminate the information that should be provided on OSS portals, so as to enable OSS producers to consider them as part of their marketing and communication strategies (Del Bianco et al., 2012).

5.2. Threats to validity

We applied the structure suggested by Yin (2009) to report threats to the validity of this study and measures for mitigating them. We report internal validity, external validity, construct validity, and reliability.

Internal Validity. One limitation that is always a part of survey research is that surveys can only reveal the perceptions of the respondents which might not fully represent reality. However, our analysis was performed by means of semi-structured interviews, which gave the interviewers the possibility to request additional information regarding unclear or imprecise statements by the respondents. The responses were analyzed and quality-checked by a team of four researchers.

External Validity. Overall, a total of 23 practitioners were interviewed. We considered only experienced respondents and did not accept any interviewees with an academic background. However, we are aware that the convenience sampling approach we adopted could be biased, even if we tried to maximize the diversity. For example, practitioners from different domains, such as those developing real-time or safety-critical systems, might have provided a different set of answers. As for the projects we selected to validate the presence of information in OSS portals, we are aware that the 100K most starred GitHub projects might not represent the whole OSS ecosystem, but we believe they might be a good representative of them. We also think that less popular projects, might only perform worst than the selected ones.

We therefore think that threats to external validity are reasonable. However, additional responses and additional projects should be analyzed in the future.

Construct Validity. The interview guidelines were developed on the basis of the previously performed surveys (Del Bianco et al., 2009; Taibi, 2015). Therefore, the questions are aligned with standard terminology and cover the most relevant characteristics and metrics. In addition, the survey was conducted in interviews, which allowed both the interviewees and the interviewer to ask questions if something was unclear.

Reliability. The survey design, its execution, and the analysis followed a strict protocol, which allows replication of the survey. However, the open questions were analyzed qualitatively, which is always subjective to some extent, but the resulting codes were documented.

Table A.1
Results from the interviews.

Factor	Measure	#	Median
Community support and adoption		10	4.5
Popularity		9	3
	# Watch	4	3
	# Stars	13	3
	# Fork	4	3
	# Downloads	13	3
Community reputation		11	3
	Member of a foundation	1	4
	Complete administration mechanism	1	5
	Fast response to issues	1	5
Community size		13	3
	# Contributors	11	4
	# Subscribers	3	3
	Community age	12	3
	# Involved developers per company	3	3
	# Independent developers	3	3
	Activeness	3	3
	Responsiveness	2	3.5
Communication		6	3.5
	Availability of questions/answers	11	3
	Availability of forum	4	2.5
	# Mailing lists	3	3
	Traffic on the mailing list	3	3
	Responsiveness of postings	4	4
	Friendliness	6	2.5
	Quality of postings	3	3
Involvement		9	3
	Clear project management	1	5
Sustainability		11	3
	Existence of maintainer	11	3
Product Team		5	3
	Developer quality	3	4
	Developer Productivity	2	3
Responsiveness		1	5
Scheduled updates		1	2.5
Fast respond to user's needs		1	2.5
Documentation		14	4
Avail. of documentation/books/online docs		5	3
	Avg time to implement new issues	1	4
	Avail. of updated documentation	9	4
Avail. of development process documentation		4	3
	Avail. of getting started tutorial	1	5
Avail. of Tutorial or Examples		5	5
Usage documentation		4	4
Avail. of best practices		4	4
		4	3
Software requirements		11	3
	Complete doc. on SW requirements	1	5
Hardware requirements		8	3.5
	Complete doc. on HW requirements	1	5
Roadmap		7	3
Test case documentation		4	3
License		21	4
	License type	20	5
	Law conformance	9	5
	License Compatibility	10	3
	OSS obligation fulfillment	1	5
	Existence of malicious OS obligation	1	5
	Contagiousness	1	5
	Multiple license option	1	2.5
	Dual License with limited features	7	4

(*continued on next page*)

This work was based on information extracted from OSS portals and the available APIs, and therefore, reliability of the assessment depends partly on the availability and reliability of the portals. Some projects might be managed and discussed on different platforms like the different issue tracking systems, the extracted information from the available APIs might be incomplete, which may affect the assessment results. On the other hand, we identified from the interviews the most used portals in each category of the sources of information. This helps mitigate the threat to some extent. Moreover, some projects might serve the purpose of providing resources, and not source code. However, Github API does not provide filtering functions towards excluding such projects. We believe that, this threat could be mitigated by the large amount of projects we selected.

Table A.1 (*continued*).

Factor	Measure	#	Median
Operational SW characteristics		6	4
Trialability		5	3
	Available for independent verification and compile	1	5
	Provide demo for quick evaluation	1	4
Independence from other SW		11	3
	Run independently	1	5
	Supports independent libraries	1	5
	Fewer dependences	7	4
Development language		5	4
	Mainstream dev Lang	4	4
	Language know in the company	2	4.5
	Programming language uniformity	5	4
Multiplatform support		5	3
Standard compliance		5	4
	Testability	6	3.5
Maturity		6	3.5
	# forks	3	3
	Stability	7	5
	Release version stability	1	5
	# releases	10	4
	Release frequency	7	4
	# releases	3	4
	Age (#Years)	4	4
	# commits	3	3
	Development versions	6	4
	System growth	9	4
	New feature integration	1	5
Risk (Perceived risks)		7	4.5
Perceived lack of integrity		5	3
Perceived high availability		5	4
	Test according to context	1	4
	Analysis and pre-examination	1	5
	Comply with business requirements	1	5
Strategic risks		5	3
	Influence of operation specified	1	4
Hazard risks		5	4
	consequences specified	1	5
	Code security	1	4
	Virus scanning	1	4
	Risk of no maintenance	1	2.5
Quality		6	3.5
Reliability		3	4
	Component reliability	2	5
	Architecture reliability	7	4
	System reliability	2	4.5
	# Bugs (open, closed, …)/bug density	8	4
	Average bug age	2	4.5
	Mean time between software failure (MTBF)	8	4
Performances		4	4.5
	Main functionality external performance standards	1	5
	Based on business	1	2.5
	Construct verification environment	1	2.5
	Comparison with similar software	1	2.5
Security		15	4
	# security vulnerabilities	12	3
	#Vulnerabilities reported on the NVD portal	14	4
	Security report	7	5
	Vulnerability Resolving time	2	5
	Community concern towards security	1	5
	Vulnerability impact	1	5
Modularity		3	3
	Select OSS based on module	1	5
Portability		3	4
	Adaptability	2	4.5
	Installability	2	4.5
Flexibility/Exploitability		3	3
	Support usage patterns	1	2.5
	Reasonable function wrapper	1	2.5

(*continued on next page*)

Table A.1 (*continued*).

Factor	Measure	#	Median
Code Quality		13	4
	Code complexity (class, methods, ..)	10	3
	Change proneness	3	3
	Fault proneness	3	4
	Test coverage	4	4.5
	Code size	7	3
	Technical difficulty	3	4
Coding conventions		9	3
	Usage of linters for checking coding conventions compliance	7	3
Maintainability		3	4
Testability		2	4
Changeability		2	3.5
Update/Upgrade/Add-ons/Plugin		3	4
	Update capability between versions	1	2.5
	Easy to update to new version	5	3
	API compatibility between versions	1	2.5
Architectural quality		5	3
Trustworthiness		6	4
Component		4	3.5
	Functionality	1	5
Architecture		4	3
	Difference with reality	1	4
System		4	3.5
	Percentage of system failure	1	5
OSS provider reputation		4	3.5
Existence of benchmark/test		4	3.5
	Fast responsiveness to malicious affairs	1	5
	Transparency	1	4
	Test cases availability	1	4
Collaboration with other product		4	2.5
	Even distribution among code submitters	1	2.5
	Consistent release updates pace	1	2.5
	In-time vulnerability publishing	1	2.5
	Measure-related information (i.e. measure possibility)	1	2.5
Assessment results from 3rd parties		2	3.75

6. Conclusion

In this paper, we investigated the factors considered by companies when selecting OSS to be integrated in the software they develop, and we analyzed their availability in the OSS portals, in particular using OSS portal APIs.

We identified a set 8 factors and 74 sub-factors, together with 170 metrics that companies commonly use to evaluate and select OSS. Unexpectedly, only a small part of the factors can be evaluated automatically, and out of 170 metrics, only 40 are available from project portals APIs.

The automated extraction of the information from the 100K most starred GitHub projects showed that only 22 metrics out of 40 returned information for all the 100K projects. 2 metrics returned information for around 80% of the projects while another 7 for around 40%. The other 4 metrics returned information for below 15%.

It is important to note that the extraction consider some of the most famous OSS projects. Therefore, we can speculate that the vast majority of less common and less used projects might provide even less information.

The result of this work enable us to create a list of updated factors and metrics, together with the list of automatically collectable ones, that practitioners can use to select OSS.

Results can be used also by researchers to further validate the factors and metrics, or providing frameworks or tools to ease the selection of OSS. Moreover, OSS producers, and repositories might also benefit of this results to understand which information they should provide from their APIs, so as to ease the evaluation of OSS projects, and increase the adoption likelihood.

Future work include the validation of the factors and metrics in industrial settings, reducing the subjectivity of the decisions. Moreover, we are planning to develop a tool and portal to automatically collect the information and enable the comparison of OSS projects, so as to ease the OSS selection phase.

CRediT authorship contribution statement

Xiaozhou Li: Software, Formal analysis, Investigation, Writing – original draft, Visualization. **Sergio Moreschini:** Software, Formal analysis, Investigation, Writing – original draft, Visualization. **Zheying Zhang:** Conceptualization, Investigation, Writing – review & editing. **Davide Taibi:** Conceptualization, Methodology, Funding acquisition, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Results from the interviews

See Table A.1.

References

Allen, J., Collison, S., Luckey, R., 2009. Ohloh web site API.
Anon, 2020a. Flossmole. https://flossmole.org. (Accessed 30 December 2020).

Anon, 2020b. Free/libre open source software metrics. https://www.flossmetrics.org. (Accessed 30 December 2020).
Anon, 2020c. GH archive. https://www.gharchive.org. (Accessed 30 December 2020).
Anon, 2020d. GH torrent portal. https://ghtorrent.org. (Accessed 30 December 2020).
Anon, 2020e. Sourge forge research data. https://www3.nd.edu/~oss/Data/data.html. (Accessed 30 December 2020).
Anon, 2020f. OpenHub. https://www.openhub.net. (Accessed 30 December 2020).
Anon, 2020g. FlossHub. https://flosshub.org. (Accessed 30 December 2020).
Anon, 2020h. Candoia. http://candoia.github.io. (Accessed 30 December 2020).
Anon, 2020i. Repograms. https://github.com/RepoGrams/RepoGrams. (Accessed 30 December 2020).
Anon, 2020j. Synopsys - EDA tools, semiconductor IP and application security solutions. https://www.synopsys.com/. (Accessed 30 December 2020).
Anon, 2020k. Whitesource. https://www.whitesourcesoftware.com. (Accessed 30 December 2020).
Anon, 2020l. Sonarcloud. https://sonarcloud.io. (Accessed 30 December 2020).
Anon, 2020m. Sonartype. https://www.sonatype.com/. (Accessed 30 December 2020).
Anon, 2020n. Whitehat security|application security platform. https://www.whitehatsec.com/platform/software-composition-analysis/. (Accessed 30 December 2020).
Anon, 2020o. Kiuwan - end-to-end application security. https://www.kiuwan.com/. (Accessed 30 December 2020).
Anon, 2020p. Software heritage. https://www.softwareheritage.org. (Accessed 30 December 2020).
Anon, 2020q. Promise. http://promise.site.uottawa.ca/SERepository/. (Accessed 30 December 2020).
Anon, 2020r. Fossid. https://fossid.com. (Accessed 30 December 2020).
Anon, 2020s. Boa. http://boa.cs.iastate.edu/boa//. (Accessed 30 December 2020).
Anon, 2020t. Stack exchange. https://stackexchange.com. (Accessed 30 December 2020).
Anon, 2020u. Reddit. http://reddit.com. (Accessed 30 December 2020).
Anon, 2020v. Github. https://www.github.com. (Accessed 30 December 2020).
Anon, 2020w. Stack overflow. https://stackoverflow.com. (Accessed 30 December 2020).
Anon, 2020x. National vulnerability database (NVD). https://nvd.nist.gov/general. (Accessed 30 December 2020).
Anon, 2020y. Common vulnerabilities and exposures (CVE). https://ossindex.sonatype.org/doc/cve. (Accessed 30 December 2020).
Anon, 2020z. Common weakness enumeration (CWE). https://cwe.mitre.org. (Accessed 30 December 2020).
Anon, 2021. Replication package. https://github.com/clowee/Exploring-Factors-and-Measures-to-Select-Open-Source-Software. (Accessed 07 January 2021).
Avgeriou, P., Taibi, D., Ampatzoglou, A., Arcelli Fontana, F., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, N., Pigazzini, I., Saarimäki, N., Sas, D.D., de Toledo, S.S., Tsintzira, A.A., 2020. An overview and comparison of technical debt measurement tools. IEEE Softw..
Battaglia, M., 2008. Convenience Sampling. Encyclopedia of Survey Research Methods. Londres: SAGE Publications.
Brown, W.J., Malveau, R.C., McCormick, H.W., Mowbray, T.J., 1998. AntiPatterns: Refactoring software, architectures, and projects in crisis. John Wiley and Sons, New York.
Bruntink, M., 2014. An initial quality analysis of the ohloh software evolution data. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 65.
Cai, Y., Zhu, D., 2016. Reputation in an open source software community: Antecedents and impacts. Decis. Support Syst. 91, 103–112.
Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.
Del Bianco, V., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., Tosi, D., 2012. A study on OSS marketing and communication strategies. In: Open Source Systems: Long-Term Sustainability. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 338–343.
Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., 2009. Quality of open source software: The qualipso trustworthiness model. In: Open Source Ecosystems: Diverse Communities Interacting. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 199–212.
Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D., 2010. The qualiSPo approach to OSS product quality evaluation. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/libre/open Source Software Research and Development. In: FLOSS, vol. 2010, New York, NY, USA, pp. 23–28.
Di Cosmo, R., Zacchiroli, S., 2017. Software heritage: Why and how to preserve software source code. In: IPRES 2017 - 14th International Conference on Digital Preservation. Kyoto, Japan, pp. 1–10.
Dixon, M., 2016. Methods and systems for generating software quality index. Google Patents, https://patents.google.com/patent/WO2009089294A3.
Duijnhouwer, F., Widdows, C., 2003. Open source maturity model. Capgemini Expert Letter.
Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N., 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: 2013 35th International Conference on Software Engineering. ICSE, pp. 422–431.
Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N., 2015. Boa: Ultra-large-scale software repository and source-code mining. ACM Trans. Softw. Eng. Methodol. 25 (1).
Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA.
Gamalielsson, J., Lundell, B., 2014. Sustainability of open source software communities beyond a fork: how and why has the LibreOffice project evolved? J. Syst. Softw. 89, 128–145.
Golden, B., 2008. Open source maturity model. Open Source Bus. Resour. 4–9, Copyright - Copyright Talent First Network May 2008; Document feature - Tables; Last updated - 2020-11-18; SubjectsTermNotLitGenreText - United States–US.
Gousios, G., 2013a. The GHTorrent dataset and tool suite. In: Proceedings Of the 10th Working Conference on Mining Software Repositories. In: MSR, vol. 13, IEEE Press, Piscataway, NJ, USA, pp. 233–236.
Gousios, G., 2013b. The GHTorrent dataset and tool suite. In: 2013 10th Working Conference on Mining Software Repositories. MSR, pp. 233–236.
Hu, D., Zhao, J.L., Cheng, J., 2012. Reputation management in an open source developer social network: An empirical study on determinants of positive evaluations. Decis. Support Syst. 53 (3), 526–533.
Jansen, S., 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. Inf. Softw. Technol. 56 (11), 1508–1519, Special issue on Software Ecosystems.
Kamei, Y., Matsumoto, T., Yamashita, K., Ubayashi, N., Iwasaki, T., Shuichi, T., 2018. Studying the cost and effectiveness of OSS quality assessment models: An experience report of Fujitsu QNET. IEICE Trans. Inf. Syst. 2744–2753.
Kilamo, T., Lenarduzzi, V., Ahoniemi, T., Jaaksi, A., Rahikkala, J., Mikkonen, T., 2020. How the cathedral embraced the bazaar, and the bazaar became a cathedral. In: Open Source Systems. Springer International Publishing, Cham, pp. 141–147.
Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L., Morasca, S., 2020. Open source software evaluation, selection, and adoption: a systematic literature review. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 437–444.
Madey, G., 2008. The sourceforge research data archive (SRDA). http://zerlot.cse.nd.edu/.
McCabe, T.J., 1976. A complexity measure. IEEE Trans. Softw. Eng. 2 (4), 308–320.
Petrinja, E., Nambakam, R., Sillitti, A., 2009. Introducing the OpenSource maturity model. In: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/libre/open Source Software Research and Development. In: FLOSS, vol. 09, IEEE Computer Society, USA, pp. 37–41.
Robles, G., Koch, S., GonZález-barahona, J.M., Carlos, J., 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems. RAMSS, IET, pp. 51–56.
Robles, G., Steinmacher, I., Adams, P., Treude, C., 2019. Twenty years of open source software: From skepticism to mainstream. IEEE Softw. 36 (6), 12–15.
Rocco, J.D., Ruscio, D.D., Sipio, C.D., Nguyen, P.T., Rubei, R., 2021. Development of recommendation systems for software engineering: the CROSSMINER experience. CoRR abs/2103.06987.
Roveda, R., Fontana, F.A., Pigazzini, I., Zanoni, M., 2018. Towards an architectural debt index. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 408–416.
Rozenberg, D., Beschastnikh, I., Kosmale, F., Poser, V., Becker, H., Palyart, M., Murphy, G.C., 2016. Comparing repositories visually with repograms. In: Proceedings Of the 13th International Conference on Mining Software Repositories. MSR, pp. 109–120.
Sbai, N., Lenarduzzi, V., Taibi, D., Sassi, S.B., Ghezala, H.H.B., 2018. Exploring information from OSS repositories and platforms to support OSS selection decisions. Inf. Softw. Technol. 104, 104–108.
Semeteys, R., 2008. Method for qualification and selection of open source software. Open Source Bus. Resour..
Soto, M., Ciolkowski, M., 2009. The qualOSS open source assessment model measuring the performance of open source communities. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 498–501.
Spadini, D., Aniche, M., Bacchelli, A., 2018. Pydriller: Python framework for mining software repositories. In: Proceedings Of the 2018 26th ACM Joint

Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 908–911.
Taibi, D., 2015. An empirical investigation on the motivations for the adoption of open source software. ISBN: 9781612084381.
Taibi, D., Lavazza, L., Morasca, S., 2007a. OpenBQR: a framework for the assessment of OSS. In: Open Source Development, Adoption and Innovation. Springer US, Boston, MA, pp. 173–186.
Taibi, D., Lavazza, L., Morasca, S., 2007b. OpenBQR: a framework for the assessment of OSS. In: Feller, J., Fitzgerald, B., Scacchi, W., Sillitti, A. (Eds.), Open Source Development, Adoption And Innovation. Springer US, Boston, MA, pp. 173–186.
Wasserman, A.I., Guo, X., McMillian, B., Qian, K., Wei, M.-Y., Xu, Q., 2017. Osspal: Finding and evaluating open source software. In: Balaguer, F., Di Cosmo, R., Garrido, A., Kon, F., Robles, G., Zacchiroli, S. (Eds.), Open Source Systems: Towards Robust Practices. Springer International Publishing, Cham, pp. 193–203.
Wasserman, A.I., Pal, M., Chan, C., 2006. The Business Readiness Rating: A Framework for Evaluating Open Source. Technical Report.
Wuetherick, B., 2010. Basics of qualitative research: Techniques and procedures for developing grounded theory. Can. J. Univ. Continuing Educ. 36.
Yin, R., 2009. Case Study Research: Design And Methods, (Applied Social Research Methods, Vol. 5), fourth ed.h SAGE Publications, Inc.