The vital role of Community in Open Source Software Development: A Framework for Assessment and Ranking

Journal:	*Journal of Software: Evolution and Process*
Manuscript ID	JSME-23-0044
Wiley - Manuscript type:	Research Article - Empirical
Date Submitted by the Author:	11-Feb-2023
Complete List of Authors:	Singh, Jaswinder; Panjab University Gupta, Anu; Panjab University Kanwal, Preet; Sri Guru Gobind Singh College
Editor Selection:	Darren Dalcher
Keywords:	Open Source Software, Project Community, Software Quality, Multi Criteria Decision Making

The vital role of Community in Open Source Software Development: A Framework for Assessment and Ranking

Jaswinder Singh[1], Dr. Anu Gupta[2], Dr. Preet Kanwal[3]

[1,2]Panjab University, Chandigarh, India
[3]Sri Guru Gobind Singh College, Chandigarh, India

jaswinder.davc@gmail.com, anugupta@pu.ac.in, pk.sggs@yahoo.com

Abstract. Open Source Software (OSS) follows a unique paradigm of software development whereby self-motivated volunteers scattered all around the globe contribute to the development in the form of code, documentation, feature recommendations, bug reporting and bug resolution. These volunteers, commonly referred to as OSS project community, serve as the foundation of the OSS project, fostering its creation, sustenance, and long-term support. The quality and sustainability of the OSS project is contingent upon the development and structure of the self-governing community. In a business organization, the acquisition of OSS is frequently predicated upon the expectation of its future maintenance and support, along with other quality criteria, such as security, reliability, and scalability. The organization wants to ensure that the OSS project will continue to be developed, maintained, and supported in the future so that it can rely on it as a stable and secure technology solution. Free and easy availability of the complete development history for millions of OSS projects from cloud based software hosting platforms like GitHub has enabled researchers to analyze and provide quantitative and scientific observations about the quality of a software project. The OSS community is a crucial aspect in evaluating the quality of an OSS project, therefore, it is necessary to assess the OSS community as part of the project quality assessment process. A Framework for Assessment and Ranking of OSS Community is being presented in the current research work following a detailed examination of the largest source code hosting and project collaboration platform, GitHub. Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) from Multi Criteria Decision Making (MCDM) toolkit has been used for ranking the project community. The effectiveness of the framework was verified through its application to 7 OSS projects hosted on GitHub. The performance score for each project was calculated using 3 sub-characteristics and 24 metrics. The outcome revealed that the "python-twitter-tools/twitter" project achieved the highest project community assessment score of 0.5906, while the "darkk/redsocks" project had the lowest score of 0.1288.

Keywords: Open Source Software, Project Community, Software Quality, Multi Criteria Decision Making

1 Introduction

With exponential growth in the adoption of OSS by industry and individuals, the demand for high quality software is ever increasing. As a result, quality assessment aspect of software engineering research has gained considerable interest [1]. OSS adoption is fairly worthwhile to individual practitioners as well as organizations as it effectively abolished the exertion and time involved in developing and testing the software in-house. Nevertheless, it should be acknowledged that this valuable advantage comes at a cost. Organizations interested in OSS acquisition need to assess the quality of the OSS project and choose among various competing solutions [2]. Moreover, organizations may have to follow laws of their regions, coding standards, security requirements, and other restrictions which the OSS project must meet to be get selected by the organization. Most important of all, the OSS project must be developed under a license that allows the easy adaptation of the software system into organization's environment. The OSS project and its future releases must comply with all the restrictions to avoid adverse effect on the operation and service quality of the business organization adopting the OSS project. In a situation where the OSS project is not complying to these restrictions, the system must be patched or replaced with a new one, which can lead the organization to unexpected service downtime and can consequently damage its trust and service quality. Individual practitioners, adopting the OSS, are also prone to various after-effects of poorly developed and maintained OSS project. Therefore, early OSS quality assessment is the need of the time and is highly recommended [3]. Big organizations can have infrastructure and manpower to assess the OSS projects with their own rigorous methods, but small scale organizations as well as individual adopters need some uncomplicated, practical and automated strategy through which they can take fast and scientific decisions. It can

be concluded from the above discussion that every stakeholder, including project adopters and maintainers, always require some sound logical reasoning for measuring the level of the quality of an OSS project [4].

Two main reasons can be stated for software quality assessment on the basis of exhaustive exploration of the literature on software quality. Firstly, the organization or individual interested in the adoption of software system want a reasonable judgement regarding the quality, support and future sustainability of the software system. Secondly, the software maintainers must ensure that the system has evolved and matured to the level that it can be released to the public. The requisite assessments are generally performed with the help of software quality models that fulfil the task by combining the defined software metrics and quality factors in a hierarchical fashion to present a single mathematical view of the software project [5]. In case of OSS, extra care is undertaken to take into account the quality of the development process as well the volunteer community built around the project in addition to the source code [6].

To streamline the OSS assessment and adoption, various OSS quality assessment models and decision support theories have been put forward in the past [7]. Various quality characteristics and corresponding software metrics have been proposed and validated to help managers and practitioners to take informed decisions on OSS selection and adoption. However, these models and methods are seldom used. Literature suggests that key reasons for non-popularity of existing OSS quality assessment methods include high learning curve to understand the method, difficulty in implementation of the proposed methodology and lack of automation [8]. Human intervention causing subjectivity in assessment process is also reported as one of the reasons for non-adoptability of these methods.

As OSS development follows a unique paradigm, whereby a group of self-motivated, geographically dispersed volunteers work together virtually for a common cause, a social network is developed among these volunteers and this social network is highly supported by various modern software hosting facilities like GitHub [9]. OSS project community is the main driving force in the sustainment and growth of the project as the volunteers produce the software, use the software, report the bugs, resolve the bugs, requests new features, fulfil these requests, maintain the project, document the project and so on. The quality of this complex and self-governing structure viz. project community is the biggest determinant to the quality of the OSS project. In the current research work, a Framework for Assessment and Ranking of OSS Project Community has been proposed which is elaborated in Section 3. Three main aspects related to OSS project community viz. Community Structure, Community Interaction and Core Contributors' Competence has been taken into account for developing the framework. GitHub, the largest software hosting platform of current times, is explored exhaustively to determine all the metrics related to an OSS project and the appropriate metrics are mapped to the three main aspects mentioned above. OSS projects under consideration are ranked as per the assessment scores obtained for their communities. Assessment and ranking of OSS projects has been done using Multi Criteria Decision Making (MCDM) approach [10]. Particularly, a widely cited and accepted MCDM method named as The Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) [11] has been applied for conducting assessment of the community of 7 OSS projects.

The paper has been structured as follows: In Section 2, the state of the art in the field of OSS quality assessment has been elaborated. Section 3 describes the proposed Framework for Assessment and Ranking of OSS Community with details regarding framework structure, assessment criteria and MCDM-TOPSIS in sub-sections. Section 4 presents the application of this framework on 7 OSS projects. Next, results are discussed and conclusions are made.

2 Related Work

The day software engineering researchers began to delve into the subject of quality in software systems, the notion of software quality models arose and began its journey towards evolution. These models decompose quality into characteristics, sub-characteristics and metrics in a hierarchical fashion. The metrics are directly measurable quantities that measure various parameters of a software project. Notable examples of quality assessment models are Rubey and Hartwick Model [12], McCall Model [13], Boehm Model [14], Dromey Model [15], FURPS Model [16-17] and ISO/IEC 9126/25010 Quality Models [18-19]. These models, generally termed as traditional quality models or closed source software quality models, assess the final software product only. That's why, traditional quality models are not adequate for the quality evaluation of OSS project because of novel practices followed by OSS which include open source code, open development process, volunteer nature, shared development, peer review, no formal support etc. [20]. To fulfil the need for OSS quality assessment, models specific to OSS quality measurement started evolving. These models consider not only the final software product but also take into account the software development process as well as project community in their assessment process. First OSS quality model is considered to be published in 2003 with the advent of Capgemini OSMM [21]. Barnard from Navicasoft mentioned about another OSMM model in his book "Succeeding with Open Source" [22]. Few other significant OSS quality assessment models are OpenBRR [23], QSOS [24], OpenBQR [25], SQO-OSS [4],

QualiPSo-OMM [26] and EFFORT [27]. A comparative study of 8 OSS quality models on the basis of seven research questions has been done in [28]. Authors concluded that there is a need for novel OSS quality assessment framework as existing quality assessment models lack in various aspects in their quality assessment practice. Some of the aspects where existing OSS quality assessment models lack are non-inclusion of social networking features provided by software hosting platforms, not considering the effect of commercial organizations in OSS development, level of subjectivity in assessment process, non-availability of tool support for automated quality assessment etc.

Hierarchical software quality models whether closed source or open source, provide a basic understanding regarding what software metrics to use and how these metrics are mapped to quality sub-characteristics and characteristics. However, most of the software quality models do not provide any information regarding how the quality metrics comprising different data units can be aggregated to quantify sub-characteristics and characteristics so that a single quality score can be obtained. Literature suggests various aggregation and ranking methods, among them MCDM is one of the most cited, validated and used method. MCDM suggests techniques like Weighted Product Method (WPM), Weighted Sum Method (WSM), Simple Additive Weighting (SAW), Hierarchical Adaptive Weighting (HAW), and Technique for order performance by similarity to ideal solution (TOPSIS) [29]. Although, MCDM is an operations research technique, it has been employed by software engineering researchers for aggregating software quality metrics to quantify various quality characteristics and sub-characteristics in a software quality framework. Nadire Cavus [30] has created a web-based tool to facilitate the selection of a Learning Management System (LMS) that not only accommodates the user's specifications but also surpasses all other alternatives that may be at one's disposal. The tool used MCDM algorithm that has been derived using artificial intelligence concepts. The methodology and software tool put forth by the authors is exclusive to LMS and cannot be adapted to any type of OSS. Ahmad et al. [31] have created a hierarchical model for OSS selection on the basis of various functional as well as non-functional criteria. Analytic Hierarchy Process (AHP) form the MCDM basket have been used for assessment and selection of most suitable OSS software project. The proposed model has been validated by applying at a technology management company in United Arab Emirates. Khondoker et al. [32] have proposed a decision making template for the selection of a Software Defined Networking (SDN) Controller that best fit the needs of an organization or an individual user. The AHP method from MCDM toolkit has been adapted by the authors for applying it on five SDN controllers among them Ryu SDN has emerged as the best controller. Azadeh et al. [33] have combined Fuzzy AHP and Fuzzy TOPSIS for the assessment, ranking and selection of appropriate simulation software among available alternatives. The applicability and superiority of the proposed approach has been confirmed through expert's judgement. In addition, sensitivity analysis has been performed for validation of the proposed assessment and selection framework.

A thorough examination of the existing literature on software quality and project community reveals that little to no research has been previously published that solely focuses on the evaluation of OSS communities. Rather community quality assessment has always been a part of overall OSS quality assessment. The OSS quality models found in literature have tried their best to give an objective look while attempting to quantify the quality of OSS projects, but with time, these models have become obsolete. Modern web-based software hosts are leveraging the new technological advancements to facilitate the OSS development process. As the existing OSS quality models don't take into account these novel software development features provided by cloud-based software hosts such as GitHub, they are not able to provide a comprehensive quality assessment desired by users. The Framework for Assessment and Ranking of OSS Community proposed in the present study is quite comprehensive in nature as it takes into account the various aspects related to project community of an OSS project. For that purpose, GitHub, the largest OSS project host has been explored comprehensively and relevant software project metrics identified from GitHub have been mapped to the community quality characteristics defined as part of the assessment criteria of the proposed framework. TOPSIS, the most cited method from MCDM toolkit, has been used for metrics aggregation, quality characteristics quantification and OSS projects ranking.

2.1 Salient features of the proposed framework

- The proposed Framework for Assessment and Ranking of OSS Community is completely objective and automatic. No human intervention is needed, as all the metrics, sub-characteristics and characteristics proposed as part of the assessment criteria of the framework can be computed automatically. Moreover, the ranking process involving MCDM-TOPSIS approach is completely automatic and objective. That's why the presented framework is ideal for modern OSS project hosts where the number of software projects is in millions.

- The proposed framework is quite exhaustive in nature as almost all the aspects related to OSS communities has been taken into account which reflects from the definition of assessment criteria.
- The proposed framework works independently of any programming language. OSS project written in any programming language and from any domain can be analyzed using the framework.

3 Proposed Framework for Assessment and Ranking of OSS Community

The idea of an OSS project is generally perceived by an individual or an organization and subsequently the project is built by contributors and managed by a core team of project maintainers. The bug reports are submitted by users and documentation is provided by non-technical as well as technical users. All these volunteer members constitute the community of the project. Quality of the software project directly influences the quality of the software system, which is the end-product of this whole exercise. In case of OSS, the major component of the software project is the volunteer community, which is the main driving force behind the development and maintenance of the software system [6]. This section elaborates the proposed Framework for Assessment and Ranking of OSS Community.

The proposed framework works in phases as depicted in the Fig. 1. Firstly, the OSS project alternatives are identified that satisfy the high level requirements of the organization or individuals. Secondly, the factors that really affect the volunteer community associated with the OSS project are explored and corresponding community quality characteristics and sub-characteristics are identified as well as proposed. OSS projects hosted on case study hosting platform (GitHub in the present case) are analyzed to identify and formulate the metrics. These metrics are then mapped to the defined characteristics and sub-characteristics as per their definition to construct a hierarchical assessment criterion. The data is procured for the designated metrics from the case study hosting platform, and the procured data is stored within a MySQL database. For a number of metrics, no additional computations are necessary on the obtained data, while other metrics are derived utilizing the data obtained in the preceding step. In the last stage, the computed metrics are aggregated for assessing the project community of chosen OSS projects. A widely cited and accepted MCDM technique known as TOPSIS is used for aggregating the metrics [10]. The OSS projects are then ranked in decreasing order of the TOPSIS performance score and the OSS project with best project community score can be recommended for adoption.

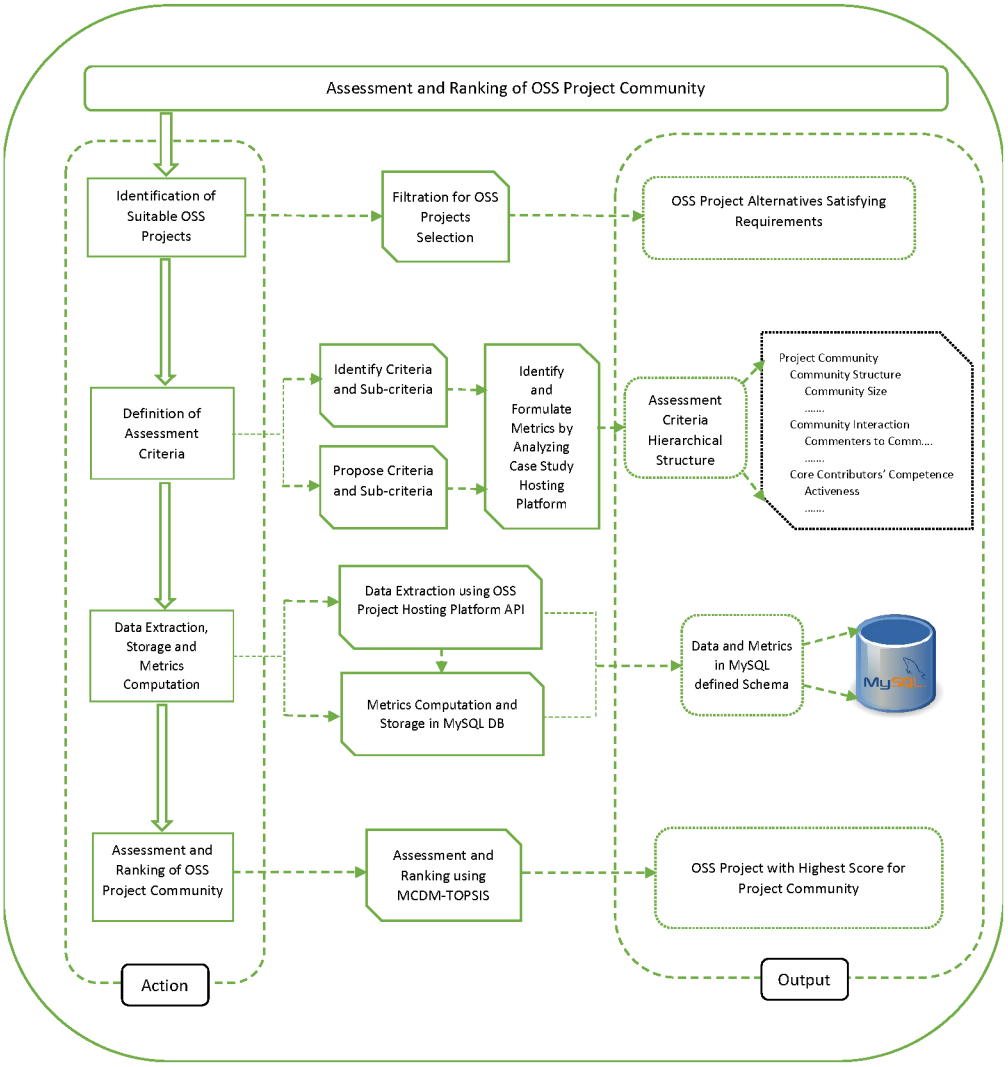

Fig. 1. Proposed Framework for Assessment and Ranking of OSS Project Community

3.1 Project Community Assessment Criteria

Project community is one of the most important aspects of the OSS quality assessment. While defining the assessment criteria for the Framework for Assessment and Ranking of OSS Project Community, following factors have been considered:

- The volunteer community built around an OSS project is the main driving force behind the development and maintenance of the software project. Project community with a reasonable size and competence is an asset to the OSS project.
- The community must be structured in such a way that every member of the community must understand his role in an unambiguous way.
- As geographically distant volunteers are participating in the development of the software system, interaction among them is another crucial factor in the success of the project.
- In every OSS project, a few of the contributors are designated as core contributors, who perform most of the designing and coding part. Their activeness and expertise play very important role in the overall development of the system. The competency of these core contributors encourages other contributors to join the project at hand, strengthening the overall project community.

Taking into account the above factors, assessment criteria for OSS project community has been defined. A simplified version of the Goal Question Metric (GQM) approach [34] has been applied to define the assessment criteria. The first goal is to "*Assess and rank the Project Community of an OSS project*". For this goal, a question has been formulated as "*How the Project Community of an OSS project can be assessed?*". For answering this question, the concept of Project Community is divided into three sub-parts namely Community Structure, Community Interaction and Core Contributors' Competence for which we formulated the questions "*How the Community Structure can be assessed?*", "*How the Community Interaction can be assessed?*" and "*How the Core Contributors' Competence can be assessed?*". Following the similar procedure, questions are formulated iteratively until a level is reached where the particular quality attributes can be quantified directly using collected data. The iterative process gives rise to a hierarchical structure where, at the summit of the hierarchy, resides the Project Community, and at the base of the hierarchy, are the metrics that can either be obtained directly or be computed through the utilization of elementary computations. A tree view of the assessment criteria is shown in Fig. 2. The meaning and significance of the metrics established have been explained in Table 1. Note that depending upon the significance of the metrics, their higher value or lower value is desired for the OSS project. For example, for the metric *Community Size*, higher value is desired as larger the size of the community, better the OSS project will be maintained. The significance of the metrics has been designated as H and L to emphasize that metrics having significance H (High) are desired to have high value and metrics having significance L (Low) are desired to have low value. Incidentally, all the metrics finalized for Project Community Assessment Criteria are of high significance.

3.1.1 Community Structure

This sub-characteristic emphasizes the structure among the members of the volunteer community. The metrics to measure this quality attribute are:

- Community Size
- Project Maintainers to Community Ratio
- Contributors to Community Ratio
- Core Contributors Count
- Contributors' Inclusion Rate

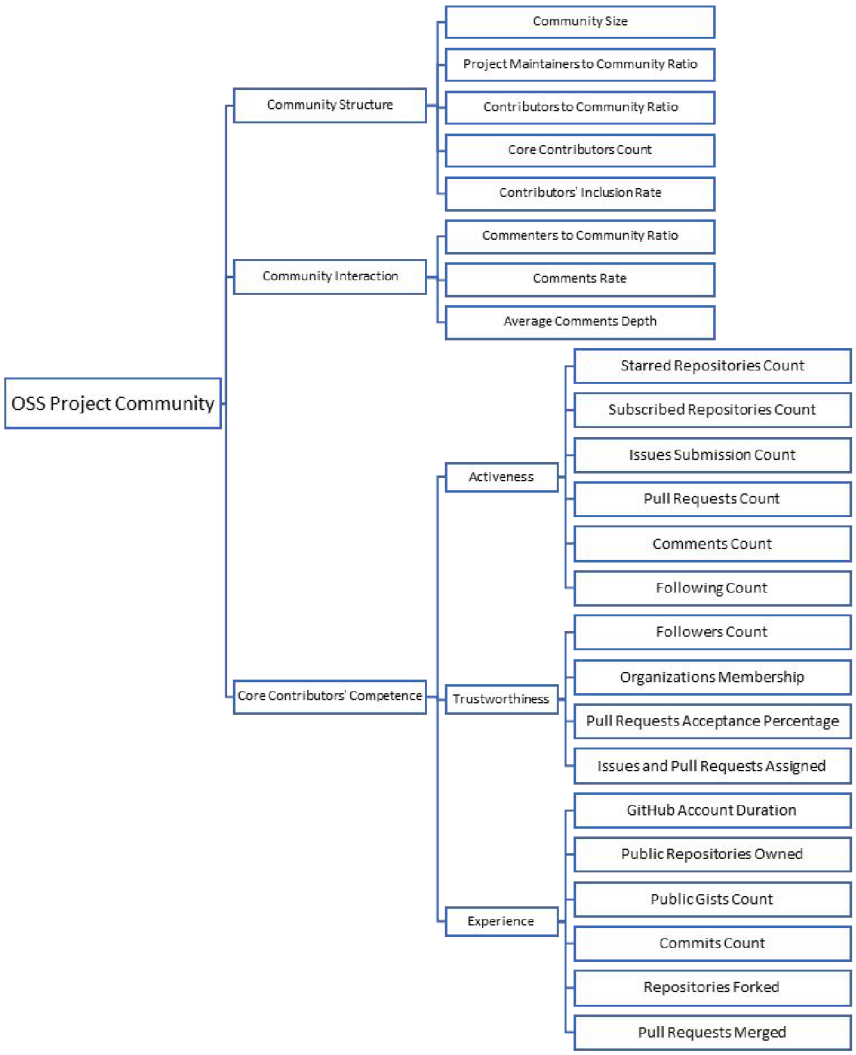

Fig. 2. Hierarchical View of Project Community Assessment Criteria

The *Community Size* metric includes all the registered GitHub users who are somehow related to the progress of the project such as committers (termed as contributors in GitHub terminology), issue and pull request submitters, issue and pull request assignees, project stargazers, project subscribers, commenters who have made any comment to issue, pull request or commit, project forkers, and collaborators. Project maintainers are those community members who maintains the OSS project and have many special privileges on the OSS project. Project maintainers are designated as "Collaborators" and "Members" in author_association attribute in the GitHub API data. *Project Maintainers to Community Ratio* metric spotlights how many members among the project community are maintaining the project. The contributors are the developers who have made at least a single commit to the software project repository ever. *Contributors to Community Ratio* emphasizes that how much percentage of the community are developers. *Core Contributors Count* metric provides the number of contributors who are doing most of the development work. The procedure for identifying core contributors have been explained further. It is quite common for developers to leave a project after some time and move on to some other project. For proper evolution and maintenance of the software project, a continuous flow of developers is required into the project. To measure the number of new contributors joining the project, the metric *Contributors' Inclusion Rate* has been devised and is equal to the average number of new contributors joining the project per month.

Table 1. Description of the metrics established for Project Community Assessment Criteria

S.No.	Metric	Description	Significance
1.	Community Size	Total size of the volunteer community	H
2.	Project Maintainers to Community Ratio	Ratio of OSS project maintainers to the total size of the community	H
3.	Contributors to Community Ratio	Ratio of community members that have ever made a commit to the total size of the community	H
4.	Core Contributors Count	Count of the core contributors	H
5.	Contributors Inclusion Rate	Average number of new contributors being added to the OSS project per month	H
6.	Commenters to Community Ratio	Ratio of community members that have ever created a commit, issue or review comment to the total size of the community	H
7.	Comments Rate	Number of commit, issue and review comments being made per month	H
8.	Average Comments Depth	Average depth of the discussion threads	H
9.	Starred Repositories Count	Total number of GitHub repositories to which the core contributors have given a star	H
10.	Subscribed Repositories Count	Total number of GitHub repositories subscribed by the core contributors	H
11.	Issues Submission Count	Total number of issues created by the core contributors	H
12.	Pull Requests Count	Total number of pull requests created by the core contributors	H
13.	Comments Count	Total number of commit, issue, and review comments created by core contributors	H
14.	Following Count	Total number of GitHub users that the core contributors are following	H
15.	Followers Count	Total number of followers of the core contributors	H
16.	Organizations Membership	Total number of organizations that the Core Contributors are member of	H
17.	Pull Requests Acceptance Percentage	The percentage of total number of Pull Requests accepted to the total number of Pull Requests Created by Core Contributors	H
18.	Issues and Pull Requests Assigned	Total Number of Issues and Pull Requests assigned to the Core Contributors	H
19.	GitHub Account Duration	Average of the total duration (in days) since the GitHub Account created by all the Core Contributors	H
20.	Public Repositories Owned	Total Number of Public Repositories created by the Core Contributors	H
21.	Public Gists[1] Count	Total Number of Public Gists created by the Core Contributors	H
22.	Commits Count	Total Number of Commits made by the Core Contributors	H
23.	Repositories Forked	Total Number of Repositories Forked by the Core Contributors	H
24.	Pull Requests Merged	Total Number of Pull Requests Merged by the Core Contributors	H

3.1.2 Community Interaction

This sub-characteristic measures the level of interaction among the community members. As the OSS project community members are geographically dispersed, the level of interaction among the members is a key to the success in such a unique global development scenario. On GitHub platform, the community members interact

[1] Gists are short snippets of code or notes which are useful to share instantly with any one. Essentially these are GitHub repositories but usually containing one file only like an article on a blog.

through the comments made on commits, issues and pull requests (called review comments). The metrics chosen for measuring the *Community Interaction* part are:

- Commenters to Community Ratio
- Comments Rate
- Average Comments Depth.

3.1.3 Core Contributors' Competence

Quality of the project community ensures that we can expect a reasonably good quality of software product, maintenance of the software system for a long duration and evolution of the software in accordance with the industry requirements. The notion of core developers or core contributors is quite common in OSS projects. The third component of project community is Core Contributors' Competence, which has been devised to find out how much active, trustworthy and experienced the core contributors are.

A novel approach has been followed while defining the core contributors. In previous literature on OSS quality assessment, core contributors were defined as the contributors whose total number of commits to the OSS project covers 80% or more of the total number of commits in the project [35-36]. It has been found that for some contributors, the number of commits is large but the number of lines added/deleted in those commits is quite low, whereas some contributors are committing code with quite a large number of source code lines in a single commit. Because, only the number of commits for core contributors' identification were considered, it may happen that the contributors with large sized commits not get identified as core contributors. To overcome this discrepancy, a novel approach has been followed where total number of commits are multiplied with source code line changes, to compensate for the smaller or large sized commits. The core contributors are then sorted out having a contribution of 80% or more of the total contribution. In simpler terms, modified weighted sum method has been employed where the metric number of commits has been assigned a weight number of code line changes in each commit.

Consider $c_1, c_2, c_3, \ldots . c_n$ to be the number of commits made by contributors 1, 2, 3,...,n and $s_1, s_2, s_3, \ldots . s_n$ are the source code line changes made by contributors 1, 2, 3,...,n in commits $c_1, c_2, c_3, \ldots . c_n$.

Total contributors' contribution = $c_1 * s_1 + c_2 * s_2 + c_3 * s_3 + \ldots\ldots\ldots\ldots + c_n * s_n = \sum_1^n c_i s_i$

Let c be the total number of commits in repository and s be total source code line changes in these c commits.

Then Total repository contribution=$c * s$

Sort the contributors' contribution in decreasing order, and pick the first k contributors as core contributors for which the cumulative contribution accounted for the 80% of the total contribution in repository.

Core Contributors' Competence have been decomposed into three sub-characteristics namely *Activeness, Trustworthiness* and *Experience.* These three sub-characteristics have been measured using various metrics for which data have been fetched from GitHub related to identified core contributors. The metrics corresponding to sub-characteristics *Activeness, Trustworthiness* and *Experience* of *Core Contributors' Competence* have been depicted in Fig. 2 and their meaning and relative significance has been elaborated in Table 1.

3.2 Multi Criteria Decision Making

MCDM, a well-known branch of operations research, is the most used decision making method in decision theory [10]. In MCDM, multiple criteria are evaluated to choose the best among various available alternatives [37]. The hierarchical software quality models consider a number of aspects in the form of various characteristics, sub-characteristics and metrics in their quality assessment process to choose among various available OSS systems; hence these models can be considered as MCDM models. Various techniques under the umbrella of MCDM has been proposed by researchers in the due course of the time. Some of the most utilized MCDM techniques are WPM, WSM, SAW, HAW, and TOPSIS. Various research publications have summarized the advantages and limitations of MCDM techniques [29][10][38][39][40][41] and have concluded that TOPSIS is one of the best techniques for real world decision problems. Number of pairwise comparisons are very less and the results are computed comparatively quickly in TOPSIS, making it suitable for situations where the number of alternatives or

criteria are large. Moreover, this technique is most suited among available MCDM methods for cases where the data is quantitative.

3.2.1 Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)

TOPSIS is a widely used, simple yet powerful MCDM technique. It compares the set of alternatives based on some predefined criteria. The method works by finding the Euclidean distance of the alternatives from the ideal solution [11]. The best alternative is chosen in such a way that the computed distance for the selected alternative must be shortest from the positive ideal solution and largest from the negative ideal solution. TOPSIS works in steps that are depicted in Fig. 3.

Step 1. Evaluation Matrix Construction

In the first step of applying TOPSIS, evaluation matrix is created by fetching data for the projects under consideration. Let us suppose, m alternatives are to be analyzed based on n number of criteria, then a $m * n$ matrix X= $[x_{ij}]_{m*n}$ is constructed with the alternatives and criteria, often called the evaluation or decision matrix. Each entry x_{ij} in the matrix represents the score of i^{th} alternative with respect to j^{th} criteria.

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

Step 2. Evaluation Matrix Normalization

The matrix $X = [x_{ij}]_{m*n}$ is then normalized using vector normalization to form the matrix $V = [v_{ij}]_{m*n}$ using the equation: $v_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}},\ i = 1,2,\dots\dots\dots\dots,m,\ j = 1,2,\dots\dots\dots\dots,n$ (1)

The resultant normalized matrix is $V = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \dots & \dots & \dots & \dots \\ v_{m1} & v_{m2} & \dots & v_{mn} \end{bmatrix}$

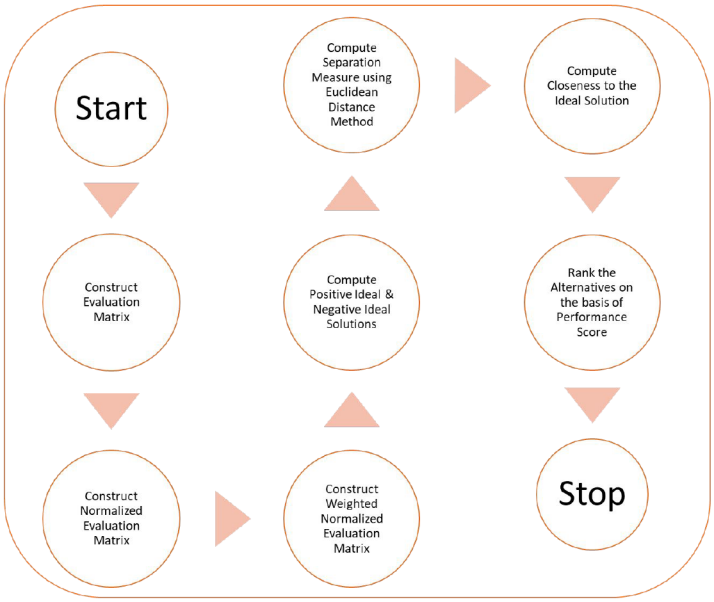

Fig. 3. MCDM-TOPSIS method

Step-3. Weighted Normalized Matrix

Next, weighted normalized evaluation matrix is constructed by multiplying each entry of the normalized matrix with weights associated with the criteria. Users and organizations can apply the weights as per their requirements and preferences. The resultant weighed normalized matrix $W = [w_{ij}]_{m*n}$ has been shown below.

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}$$

Step-4. Positive and Negative ideal solution construction

Positive and Negative ideal solutions are computed next, so that the Euclidean distance of each of the alternative can be computed from these solutions. As per the definition of the TOPSIS method, the alternative with minimum distance from positive ideal solution and maximum distance from negative ideal solution will be the finalized best alternative. The positive ideal solution is the solution that maximizes the beneficial criteria (criteria for which high value is desired like *Commits Frequency*) and minimizes the non-beneficial criteria (criteria for which low value is desired like *Issue Response Rate*) whereas the negative ideal solution maximizes the non-beneficial criteria and minimizes the beneficial criteria. The two solutions are computed as

$$A^+ = \{(max(w_{ij}|i = 1, 2, 3, \dots\dots, m)|j \in J^+), (min(w_{ij}|i = 1, 2, 3, \dots\dots, m)|j \in J^-)\}$$
$$= \{w_j^+, j = 1, 2, 3, n\} \quad (2)$$

$$A^- = \{(min(w_{ij}|i = 1, 2, 3, \dots\dots, m)|j \in J^+), (max(w_{ij}|i = 1, 2, 3, \dots\dots, m)|j \in J^-)\}$$
$$= \{w_j^-, j = 1, 2, 3, n\} \quad (3)$$

Step 5. Separation measure from positive ideal solution and negative ideal solution

Separation measure is computed for each given alternative from the positive ideal solution A^+ and negative ideal solution A^- using the Euclidean distance formula shown below. The two distances are represented as S_{i^+} and S_{i^-}

$$S_{i^+} = \sqrt{\sum_{j=1}^{n} (w_{ij} - w_j^+)^2}, \; i = 1, 2, 3, \dots\dots, m \quad (4)$$

$$S_{i^-} = \sqrt{\sum_{j=1}^{n} (w_{ij} - w_j^-)^2}, \; i = 1, 2, 3, \dots\dots, m \quad (5)$$

Step 6. Closeness to the ideal solution

After computing these separation measures, final performance value C_i, which signifies the closeness to the ideal solution, is computed from S_{i^+} and S_{i^-} using equation written below.

$$C_i = \frac{S_{i-}}{(S_{i+} + S_{i-})}, \quad 0 < C_i < 1, \quad i = 1, 2, 3, \ldots\ldots, m \qquad (6)$$

Step 7. Final Ranking

On the basis of the performance scores computed in previous step, the selected OSS project alternatives are ranked in decreasing order of performance score obtained for their project community.

4 Framework Implementation and Validation

The framework has been validated by applying it on 7 randomly selected OSS projects hosted on GitHub. Research data for these 7 OSS projects is fetched through GitHub REST-API, the most suitable method provided by GitHub for extracting data related to any hosted OSS project. Final Project Community score is then computed for the selected projects using MCDM-TOPSIS and the OSS projects under consideration are ranked in decreasing order of the obtained score. A brief introduction and meta-data about the selected 7 projects have been given in Table 2.

Table 2. OSS Projects chosen for framework application

S. No.	Full Name of the OSS Project	Acronym Used	Brief Description	Number of Commits	Number of Stars
1	prototypejs/prototype[2]	Project-1	Prototype JavaScript framework	1259	3511
2	darkk/redsocks[3]	Project-2	Transparent TCP-to-proxy redirector	211	2863
3	web2py/web2py[4]	Project-3	Free and open source full-stack enterprise framework	8250	2031
4	python-twitter-tools/twitter[5]	Project-4	Python Twitter API	751	2931
5	vifm/vifm[6]	Project-5	A file manager with curses interface	12413	2281
6	django-haystack/django-haystack[7]	Project-6	Tool for modular search for Django	1815	3389
7	connectbot/connectbot[8]	Project-7	ConnectBot is the first SSH client for Android.	2113	2001

4.1 Project Community Assessment using MCDM-TOPSIS

Step 1. The evaluation matrix is constructed by extracting data for 7 OSS projects hosted on GitHub using its official REST-API as shown in Table 3.

[2] https://github.com/prototypejs/prototype
[3] https://github.com/darkk/redsocks
[4] https://github.com/web2py/web2py
[5] https://github.com/python-twitter-tools/twitter
[6] https://github.com/vifm/vifm
[7] https://github.com/django-haystack/django-haystack
[8] https://github.com/connectbot/connectbot

Table 3. TOPSIS Evaluation Matrix

OSS Project	Community Interaction			Community Structure					Core Contributors' Competence																
									Activeness						Trustworthiness				Experience						
	Commenters to Community Ratio	Comments Rate	Average Comments Depth	Community Size	Project Maintainers to Community Ratio	Contributors to Community Ratio	Core Contributors Count	Contributors Inclusion Rate	Starred Repositories Count	Subscribed Repositories Count	Issues Submission Count	Pull Requests Count	Comments Count	Following Count	Followers Count	Organizations Membership	Pull Request Acceptance Percentage	Issues and Pull Requests Assigned	GitHub Account Duration	Public Repositories Owned	Public Gists Owned	Commits Count	Repositories Forked	Pull Requests Merged
Project-1	0.0392	8.3296	3.8527	4204	0.0093	0.0121	2	0.2591	262	107	645	816	2561	116	594	7	168.9516	172	10875	278	142	13352	219	725
Project-2	0.0420	3.2471	4.0357	3378	0.0000	0.0041	2	0.0638	339	123	180	207	657	41	479	2	132.2981	21	10070	75	35	24810	49	163
Project-3	0.1450	28.5906	2.7668	3318	0.0115	0.0817	1	2.0000	35	115	72	47	959	2	515	0	78.7234	67	4201	92	11	115133	25	37
Project-4	0.0575	7.2471	3.1509	3703	0.0051	0.0267	3	0.5449	817	848	1430	5903	10975	207	731	24	222.1588	104	13333	1133	38	65955	944	4997
Project-5	0.1266	20.2722	3.9334	2607	0.0015	0.0222	1	0.3373	73	69	23	65	1254	8	70	1	72.3077	280	4484	54	9	22373	31	47
Project-6	0.2221	31.2367	3.3624	5452	0.0000	0.0405	2	1.2959	2666	267	667	816	3976	279	1254	4	132.9311	397	10588	316	366	21761	207	598
Project-7	0.1628	14.0798	3.1267	2850	0.0046	0.0260	1	0.2852	36	68	87	511	955	1	213	2	94.3249	254	4970	67	9	4970915	38	482

Step 2. The matrix $X = [x_{ij}]_{m*n}$ is then normalized using vector normalization to form the matrix $V = [v_{ij}]_{m*n}$ using equation 1. The normalized matrix has been shown in Table 4.

Table 4. Normalized Evaluation Matrix

OSS Project	Community Interaction			Community Structure					Core Contributors' Competence																
									Activeness						Trustworthiness				Experience						
	Commenters to Community Ratio	Comments Rate	Average Comments Depth	Community Size	Project Maintainers to Community Ratio	Contributors to Community Ratio	Core Contributors Count	Contributors Inclusion Rate	Starred Repositories Count	Subscribed Repositories Count	Issues Submission Count	Pull Requests Count	Comments Count	Following Count	Followers Count	Organizations Membership	Pull Request Acceptance Percentage	Issues and Pull Requests Assigned	GitHub Account Duration	Public Repositories Owned	Public Gists Owned	Commits Count	Repositories Forked	Pull Requests Merged
Project-1	0.1135	0.1654	0.4172	0.4237	0.5680	0.1192	0.4082	0.1037	0.0928	0.1168	0.3754	0.1351	0.2115	0.3147	0.3427	0.2746	0.4626	0.2925	0.4547	0.2283	0.3583	0.0027	0.2204	0.1419
Project-2	0.1216	0.0645	0.4370	0.3405	0.0000	0.0407	0.4082	0.0255	0.1201	0.1342	0.1048	0.0343	0.0543	0.1112	0.2763	0.0784	0.3622	0.0357	0.4210	0.0616	0.0883	0.0050	0.0493	0.0319
Project-3	0.4194	0.5679	0.2996	0.3344	0.7013	0.8024	0.2041	0.8005	0.0124	0.1255	0.0419	0.0078	0.0792	0.0054	0.2971	0.0000	0.2155	0.1139	0.1757	0.0756	0.0278	0.0232	0.0252	0.0072
Project-4	0.1664	0.1439	0.3412	0.3732	0.3142	0.2626	0.6124	0.2181	0.2895	0.9255	0.8324	0.9772	0.9063	0.5615	0.4217	0.9414	0.6083	0.1768	0.5575	0.9306	0.0959	0.0133	0.9500	0.9778
Project-5	0.3662	0.4027	0.4260	0.2627	0.0940	0.2186	0.2041	0.1350	0.0259	0.0753	0.0134	0.0108	0.1036	0.0217	0.0404	0.0392	0.1980	0.4761	0.1875	0.0444	0.0227	0.0045	0.0312	0.0092
Project-6	0.6426	0.6204	0.3641	0.5495	0.0000	0.3982	0.4082	0.5187	0.9446	0.2914	0.3882	0.1351	0.3283	0.7568	0.7234	0.1569	0.3640	0.6751	0.4427	0.2595	0.9235	0.0044	0.2083	0.1170
Project-7	0.4710	0.2797	0.3386	0.2872	0.2793	0.2551	0.2041	0.1141	0.0128	0.0742	0.0506	0.0846	0.0789	0.0027	0.1229	0.0784	0.2583	0.4319	0.2078	0.0550	0.0227	0.9996	0.0382	0.0943

Step-3. Here in the current research work, it is assumed that all the software quality metrics are of equal importance and hence equal weight. Therefore, the matrix shown in Table 4 remains the same. Although, the users and organizations can set the weights as per their preferences and requirements.

Step-4. Positive ideal and Negative ideal solutions are computed next, so that Euclidean distance of each of the alternative can be computed from these solutions. According to the TOPSIS method, the alternative with minimum distance from positive ideal solution and maximum distance from negative ideal solution will be the best alternative. The two distances are represented as V_j^+ and V_j^- in Table 5.

Step 5. Separation measure (Positive ideal distance and Negative ideal distance) is computed for each given alternative from the positive ideal solution and negative ideal solution using the Euclidean distance formula. The two distances are represented as S_i^+ and S_i^- in Table 5.

Table 5. Positive and Negative ideal solution construction, separation measure and TOPSIS score.

Metrics	Project-1	Project-2	Project-3	Project-4	Project-5	Project-6	Project-7	Significance	V_j^+	V_j^-
Commenters to Community Ratio	0.1135	0.1216	0.4194	0.1664	0.3662	0.6426	0.4710	H	0.6426	0.1135
Comments Rate	0.1654	0.0645	0.5679	0.1439	0.4027	0.6204	0.2797	H	0.6204	0.0645
Average Comments Depth	0.4172	0.4370	0.2996	0.3412	0.4260	0.3641	0.3386	H	0.4370	0.2996
Community Size	0.4237	0.3405	0.3344	0.3732	0.2627	0.5495	0.2872	H	0.5495	0.2627
Project Maintainers to Community Ratio	0.5680	0.0000	0.7013	0.3142	0.0940	0.0000	0.2793	H	0.7013	0.0000
Contributors to Community Ratio	0.1192	0.0407	0.8024	0.2626	0.2186	0.3982	0.2551	H	0.8024	0.0407
Core Contributors Count	0.4082	0.4082	0.2041	0.6124	0.2041	0.4082	0.2041	H	0.6124	0.2041
Contributors Inclusion Rate	0.1037	0.0255	0.8005	0.2181	0.1350	0.5187	0.1141	H	0.8005	0.0255
Starred Repositories Count	0.0928	0.1201	0.0124	0.2895	0.0259	0.9446	0.0128	H	0.9446	0.0124
Subscribed Repositories Count	0.1168	0.1342	0.1255	0.9255	0.0753	0.2914	0.0742	H	0.9255	0.0742
Issues Submission Count	0.3754	0.1048	0.0419	0.8324	0.0134	0.3882	0.0506	H	0.8324	0.0134
Pull Requests Count	0.1351	0.0343	0.0078	0.9772	0.0108	0.1351	0.0846	H	0.9772	0.0078
Comments Count	0.2115	0.0543	0.0792	0.9063	0.1036	0.3283	0.0789	H	0.9063	0.0543
Following Count	0.3147	0.1112	0.0054	0.5615	0.0217	0.7568	0.0027	H	0.7568	0.0027
Followers Count	0.3427	0.2763	0.2971	0.4217	0.0404	0.7234	0.1229	H	0.7234	0.0404
Organizations Membership	0.2746	0.0784	0.0000	0.9414	0.0392	0.1569	0.0784	H	0.9414	0.0000
Pull Requests Acceptance Percentage	0.4626	0.3622	0.2155	0.6083	0.1980	0.3640	0.2583	H	0.6083	0.1980
Issues and Pull Requests Assigned	0.2925	0.0357	0.1139	0.1768	0.4761	0.6751	0.4319	H	0.6751	0.0357
GitHub Account Duration	0.4547	0.4210	0.1757	0.5575	0.1875	0.4427	0.2078	H	0.5575	0.1757
Public Repositories Owned	0.2283	0.0616	0.0756	0.9306	0.0444	0.2595	0.0550	H	0.9306	0.0444
Public Gists Count	0.3583	0.0883	0.0278	0.0959	0.0227	0.9235	0.0227	H	0.9235	0.0227
Commits Count	0.0027	0.0050	0.0232	0.0133	0.0045	0.0044	0.9996	H	0.9996	0.0027
Repositories Forked	0.2204	0.0493	0.0252	0.9500	0.0312	0.2083	0.0382	H	0.9500	0.0252
Pull Requests Merged	0.1419	0.0319	0.0072	0.9778	0.0092	0.1170	0.0943	H	0.9778	0.0072
S_i^+ (Positive Ideal Distance)	2.8802	1.1346	3.4330	0.5075	3.2517	1.4499	1.9336			
S_i^- (Negative Ideal Distance)	1.1346	3.4330	0.5075	3.2517	1.4499	1.9336	2.7891			
TOPSIS Score	0.2826	0.1288	0.3084	0.5906	0.1622	0.4730	0.2767			

Final Ranking	4	7	3	1	6	2	5			

Step 6. After computing these separation measures, final performance value C_i is computed from S_i^+ and S_i^- as shown in Table 6.

Step 7. The OSS projects chosen for the current study are ranked as per their performance values (Table 6).

Table 6. Final Ranking of the OSS projects.

OSS Project	Acronym	TOPSIS Performance Score	Rank
python-twitter-tools/twitter	Project-4	0.5906	1
django-haystack/django-haystack	Project-6	0.4730	2
web2py/web2py	Project-3	0.3084	3
prototypejs/prototype	Project-1	0.2826	4
connectbot/connectbot	Project-7	0.2767	5
vifm/vifm	Project-5	0.1622	6
darkk/redsocks	Project-2	0.1288	7

5 Results and Discussion

7 OSS projects are chosen from GitHub for the application of the proposed Framework for Assessment and Ranking of OSS Community. The final performance score (Table 6) has been computed for these 7 OSS projects on the basis of 3 sub-characteristics and 24 metrics. Project-4 (python-twitter-tools/twitter) has secured highest performance score of 0.5906 and Project-2 (darkk/redsocks) has scored least with 0.1288 marks. To have a detailed analysis of the results obtained, TOPSIS method has been applied separately on the three sub-characteristics *Community Structure, Community Interaction* and *Core Contributors' Competence* of Project Community. The TOPSIS performance scores obtained for the three sub-characteristics has been depicted in Table 7, Table 8 and Table 9. In addition, a line chart mixed with bar chart has been constructed (see Fig. 4) depicting the trends of the performance scores in a better way. The bars in the chart are representing the overall Project Community score for the 7 OSS projects and three lines corresponds to the performance scores for three sub-characteristics of Project Community.

Table 7. Community Structure Performance Score

OSS Projects	Performance Score	Rank
Project-1	0.3849	4
Project-2	0.1414	7
Project-3	0.7373	1
Project-4	0.4011	3
Project-5	0.1621	6
Project-6	0.4440	2
Project-7	0.2506	5

Table 8. Community Interaction Performance Score

OSS Projects	Performance Score	Rank
Project-1	0.1817	5
Project-2	0.1530	6
Project-3	0.6878	2
Project-4	0.1327	7
Project-5	0.5559	3
Project-6	0.9135	1
Project-7	0.5154	4

Table 9. Core Contributors' Competence Performance Score

OSS Projects	Performance Score	Rank
Project-1	0.2627	4
Project-2	0.1244	5
Project-3	0.0800	7
Project-4	0.6341	1
Project-5	0.1217	6
Project-6	0.4581	2
Project-7	0.2678	3

From the chart, it is quite evident that for Project-1 (prototypejs/prototype) and Project-2 (darkk/redsocks), the scores for the three sub-characteristics are low which resulted in low score for Project Community. For Project-3, although the score for *Community Structure* and *Community Interaction* are high, the average performance score can be attributed to the low score of *Core Contributors' Competence* part. The highest value of performance score for Project-4 can be credited to high value of *Core Contributors' Competence* score and good value of *Community Structure* score, although the score for *Community Interaction* is not very impressive. Project-5 has low overall score as *Community Structure* and *Core Contributors' Competence* are having low values. Although, *Community Interaction* score for Project-6 is very high, it is accompanied with average values of other two sub-characteristics which led Project-6 to achieve an overall second position. For Project-7, low overall value can be attributed to low value of *Community Structure* and *Core Contributors' Competence.*

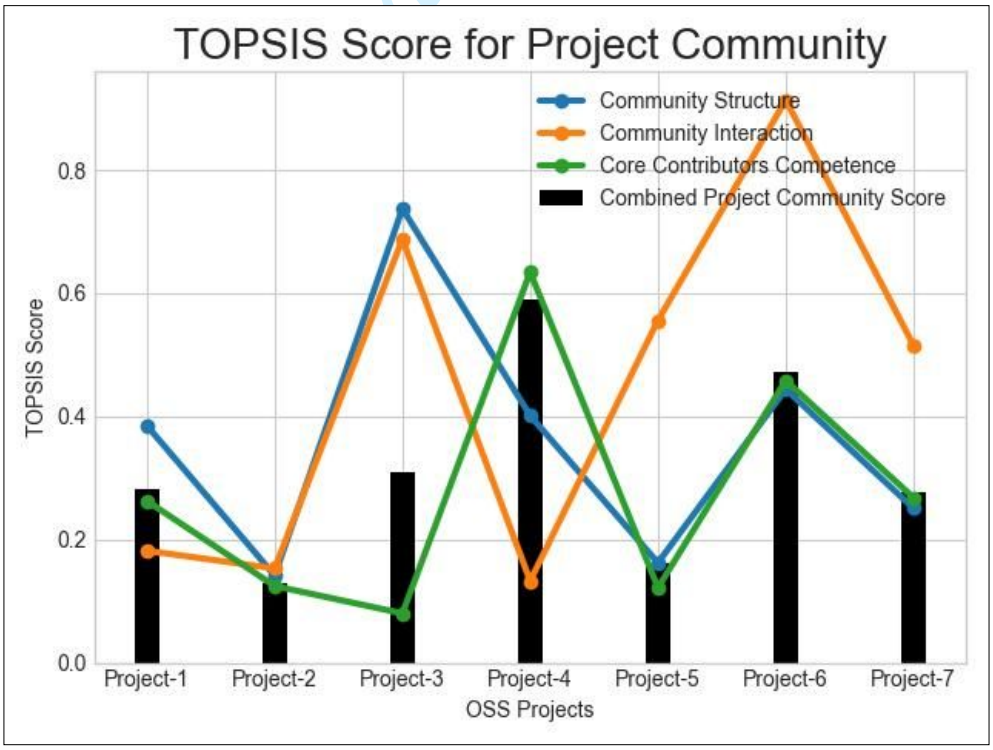

Fig. 4. Project Community TOPSIS score compared with the scores of its three sub-characteristics

6 Conclusion

A Framework for Assessment and Ranking of OSS Community has been proposed. The framework works in stages and has been designed by carefully examining all aspects of OSS project community. In the process of

defining assessment criteria for assessing the Project Community, the concept of OSS Project Community has been divided into three parts namely *Community Structure*, *Community Interaction* and *Core Contributors' Competence*. The OSS project metrics related with volunteer community have been formulated as well as identified by thoroughly analyzing GitHub and have been mapped with the characteristics and sub-characteristics defined. The proposed framework is validated by applying it on 7 randomly selected GitHub OSS projects and the OSS projects are ranked on the basis of performance score obtained by the application of MCDM-TOPSIS method. OSS Project "python-twitter-tools/twitter" has secured highest performance score of 0.5906 and OSS Project darkk/redsocks has scored least with 0.1288 marks. The TOPSIS method has also been applied on the three sub-characteristics of OSS Project Community so that an in-depth analysis can be performed on the obtained performance score for Project Community. It has been observed that the effect of Core Contributors' Competence is greatest and effect of Community Interaction aspect is least on the overall performance score. Such trend can be explained on the basis that the sub-characteristic Core Contributors' Competence constitutes 16 metrics which take part in assessment process whereas the sub-characteristic Community Interaction contributes only 3 metrics towards assessment. The proposed framework will aid business organizations as well as individual open source practitioners in identifying an OSS project with a strong volunteer community that has the potential to deliver a high quality OSS product with sustained product.

References

[1] G. Arcos-Medina and D. Mauricio, "Aspects of software quality applied to the process of agile software development: a systematic literature review," *Int. J. Syst. Assur. Eng. Manag.*, vol. 10, no. 5, pp. 867–897, 2019, doi: 10.1007/s13198-019-00840-7.

[2] M. Höst and A. Oručević-Alagić, "A systematic review of research on open source software in commercial software product development," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 616–624, 2011, doi: https://doi.org/10.1016/j.infsof.2010.12.009.

[3] A. Adewumi, S. Misra, and N. Omoregbe, "A review of models for evaluating quality in open source software," *IERI Procedia*, vol. 4, pp. 88–92, 2013.

[4] D. Spinellis *et al.*, "Evaluating the Quality of Open Source Software," *Electron. Notes Theor. Comput. Sci.*, vol. 233, pp. 5–28, 2009, doi: https://doi.org/10.1016/j.entcs.2009.02.058.

[5] B. Almeida *et al.*, "OSSMETER: Automated Measurement and Analysis of Open Source Software.," in *STAF Projects Showcase*, 2015, pp. 36–43.

[6] Y. Kuwata, K. Takeda, and H. Miura, "A study on maturity model of open source software community to estimate the quality of products," *Procedia Comput. Sci.*, vol. 35, pp. 1711–1717, 2014.

[7] K. Haaland, A.-K. Groven, N. Regnesentral, R. Glott, A. Tannenberg, and A. S. FreeCode, "Free/libre open source quality models-a comparison between two approaches," in *4th FLOS International Workshop on Free/Libre/Open Source Software*, 2010, pp. 1–17.

[8] A. Adewumi, S. Misra, N. Omoregbe, and L. F. Sanz, "FOSSES: Framework for open-source software evaluation and selection," *Softw. Pract. Exp.*, vol. 49, no. 5, pp. 780–812, 2019, doi: https://doi.org/10.1002/spe.2682.

[9] "GitHub: Where the world builds software." https://github.com/ (accessed Dec. 30, 2022).

[10] E. Triantaphyllou, B. Shu, S. N. Sanchez, and T. Ray, "Multi-criteria decision making: an operations research approach," *Encycl. Electr. Electron. Eng.*, vol. 15, no. 1998, pp. 175–186, 1998.

[11] C.-L. Hwang and K. Yoon, "Methods for Multiple Attribute Decision Making," in *Multiple Attribute Decision Making: Methods and Applications A State-of-the-Art Survey*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 58–191. doi: 10.1007/978-3-642-48318-9_3.

[12] R. J. Rubey and R. D. Hartwick, "Quantitative Measurement of Program Quality," in *Proceedings of the 1968 23rd ACM National Conference*, 1968, pp. 671–677. doi: 10.1145/800186.810631.

[13] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in *Proceedings of the software quality assurance workshop on Functional and performance issues*, 1978, pp. 133–139.

[14] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 592–605.

[15] R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, 1995.

[16] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. USA: Prentice-Hall, Inc., 1992.

[17] J. P. Miguel, D. Mauricio, and G. Rodriguez, "A Review of Software Quality Models for the Evaluation of Software Products," *CoRR*, vol. abs/1412.2, 2014, [Online]. Available: http://arxiv.org/abs/1412.2977

[18] ISO/IEC, "ISO/IEC 9126 Software engineering - Product quality," ISO/IEC, 1991.

[19] I. O. F. O. R. STANDARDIZATION, "ISO/IEC 25010: Systems and software engineering-systems and Software Quality Requirements and Evaluation (SQuaRE)." System and software quality models Geneva, 2011.
[20] A. I. Wasserman, X. Guo, B. McMillian, K. Qian, M.-Y. Wei, and Q. Xu, "OSSpal: finding and evaluating open source software," in *IFIP International Conference on Open Source Systems*, 2017, pp. 193–203.
[21] F. W. D. and C. Widdows, "Open Source Maturity Model," *Capgemini Expert Lett.*, 2003.
[22] B. Golden, *Succeeding with open source*. Addison-Wesley Professional, 2005.
[23] C. Wasserman, A.I., Pal, M., Chan, "Business readiness rating for open source," 2006.
[24] A. Origin, "Method for qualification and selection of open source software (QSOS)," *Web Publ. http//www. qsos. org (Last Visit. Jan., 2011)*, 2004.
[25] D. Taibi, L. Lavazza, and S. Morasca, "OpenBQR: a framework for the assessment of OSS," in *Open Source Development, Adoption and Innovation*, 2007, pp. 173–186.
[26] E. Petrinja and G. Succi, "Assessing the open source development processes using OMM," *Adv. Softw. Eng.*, vol. 2012, 2012.
[27] L. Aversano and M. Tortorella, "Evaluating the quality of free/Open source systems: A case study," in *International Conference on Enterprise Information Systems*, 2010, pp. 119–134.
[28] J. Singh, A. Gupta, and P. Kanwal, "Quality Assessment Models for Open Source Projects Hosted on Modern Web-Based Forges: A Review," in *Information, Communication and Computing Technology*, 2019, pp. 36–47.
[29] A. A. Zaidan, B. B. Zaidan, M. Hussain, A. Haiqi, M. L. Mat Kiah, and M. Abdulnabi, "Multi-criteria analysis for OS-EMR software selection problem: A comparative study," *Decis. Support Syst.*, vol. 78, pp. 15–27, 2015, doi: https://doi.org/10.1016/j.dss.2015.07.002.
[30] N. Cavus, "The evaluation of Learning Management Systems using an artificial intelligence fuzzy logic algorithm," *Adv. Eng. Softw.*, vol. 41, no. 2, pp. 248–254, 2010, doi: https://doi.org/10.1016/j.advengsoft.2009.07.009.
[31] N. Ahmad and P. A. Laplante, "A Systematic Approach to Evaluating Open Source Software," in *Strategic Adoption of Technological Innovations*, C. Howard, Ed. Hershey, PA, USA: IGI Global, 2013, pp. 50–69. doi: 10.4018/978-1-4666-2782-6.ch004.
[32] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, pp. 1–7. doi: 10.1109/WCCAIS.2014.6916572.
[33] A. Azadeh, S. Nazari-Shirkouhi, H. Samadi, and A. Nazari-Shirkouhi, "An integrated fuzzy group decision making approach for evaluation and selection of best simulation software packages," *Int. J. Ind. Syst. Eng.*, vol. 18, no. 2, pp. 256–282, 2014.
[34] V. R. B. G. Caldiera and H. D. Rombach, "The goal question metric approach," *Encycl. Softw. Eng.*, pp. 528–532, 1994.
[35] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams, "Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source Android apps," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, 2013, pp. 283–297.
[36] A. Mockus, R. T. Fielding, and J. Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," in *Proceedings of the 22nd International Conference on Software Engineering*, 2000, pp. 263–272. doi: 10.1145/337180.337209.
[37] P.-L. Yu, *Multiple-criteria decision making: concepts, techniques, and extensions*, vol. 30. Springer Science \& Business Media, 2013.
[38] E. Triantaphyllou, "Multi-Criteria Decision Making Methods BT - Multi-criteria Decision Making Methods: A Comparative Study," E. Triantaphyllou, Ed. Boston, MA: Springer US, 2000, pp. 5–21. doi: 10.1007/978-1-4757-3157-6_2.
[39] M. Aruldoss, T. M. Lakshmi, and V. P. Venkatesan, "A survey on multi criteria decision making methods and its applications," *Am. J. Inf. Syst.*, vol. 1, no. 1, pp. 31–43, 2013.
[40] K. P. Yoon and C.-L. Hwang, *Multiple attribute decision making: an introduction*. Sage publications, 1995.
[41] M. Mansooreh and J. Pet-Edwards, "Technical briefing: making multiple-objective decisions," *Inst. Electr. ve Electron. Eng. Inc., IEEE Comput. Soc. Press. USA*, 1997.