applied sciences

MDPI

Article

Analyzing Key Features of Open Source Software Survivability with Random Forest

Sohee Park † **and Gihwon Kwon** *,†

Department of SW Safety and Cyber Security, Kyonggi University, Suwon-si 16227, Gyeonggi-do, Republic of Korea; sosop@kyonggi.ac.kr

* Correspondence: khkwon@kyonggi.ac.kr; Tel.: +82-10-2772-3632

† These authors contributed equally to this work.

Abstract: Open source software (OSS) projects rely on voluntary contributions, but their long-term survivability depends on sustained community engagement and effective problem-solving. Survivability, critical for maintaining project quality and trustworthiness, is closely linked to issue activity, as unresolved issues reflect a decline in maintenance capacity and problem-solving ability. Thus, analyzing issue retention rates provides valuable insights into a project's health. This study evaluates OSS survivability by identifying the features that influence issue activity and analyzing their relationships with survivability. Kaplan–Meier survival analysis is employed to quantify issue activity and visualize trends in unresolved issue rates, providing a measure of project maintenance dynamics. A random forest model is used to examine the relationships between project features—such as popularity metrics, community engagement, code complexity, and project age—and issue retention rates. The results show that stars significantly reduce issue retention rates, with rates dropping from 0.62 to 0.52 as stars increase to 4000, while larger codebases, higher cyclomatic complexity, and older project age are associated with unresolved issue rates, rising by up to 15%. Forks also have a nonlinear impact, initially stabilizing retention rates but increasing unresolved issues as contributions became unmanageable. By identifying these critical factors and quantifying their impacts, this research offers actionable insights for OSS project managers to enhance project survivability and address key maintenance challenges, ensuring sustainable long-term success.

Keywords: Kaplan–Meier survival function; open source software; random forest; software maintenance; survivability

check for updates

Academic Editor: Alessandro Di Nuovo

Received: 27 November 2024
Revised: 28 December 2024
Accepted: 6 January 2025
Published: 18 January 2025

Citation: Park, S.; Kwon, G. Analyzing Key Features of Open Source Software Survivability with Random Forest. *Appl. Sci.* **2025**, *15*, 946. https://doi.org/10.3390/app15020946

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Open source software (OSS) refers to software whose source code is freely available for use, modification, and distribution [1]. This openness fosters transparency and collaboration, allowing developers worldwide to contribute their ideas and expertise. As a result, OSS has become a significant driver of innovation, offering advantages such as cost efficiency, flexibility, and scalability [2,3]. Platforms like GitHub play a pivotal role in managing OSS projects, providing tools for source code management, issue tracking, and contributor coordination. These platforms enable developers to work collaboratively across geographical boundaries, creating a dynamic and self-sustaining ecosystem that supports rapid software evolution.

Despite these advantages, OSS projects face significant challenges due to their dependence on voluntary contributors. While some contributors are deeply involved in the

long-term maintenance of projects, many participate only sporadically or for short periods [4,5]. This lack of consistent participation can lead to an accumulation of unresolved issues, such as bugs or feature requests, resulting in maintenance delays and reduced project reliability. These challenges are particularly problematic for OSS projects that lack dedicated maintenance personnel or established management frameworks. Over time, such delays hinder the project's ability to deliver updates or fix vulnerabilities, eroding user trust and diminishing software usability [6].

The impact of these challenges extends beyond individual projects, as OSS often serves as a foundation for commercial software systems and business applications. Unresolved vulnerabilities or maintenance failures in OSS can create cascading risks, affecting dependent systems and applications. For example, a poorly maintained OSS library used in a critical business system might lead to security breaches, system malfunctions, or even operational downtime [7]. Moreover, abandoned OSS projects, which no longer receive updates or security patches, leave developers with limited options. They must either seek alternative solutions or rebuild dependent software from scratch, which can be both costly and time-consuming [8,9]. These issues underscore the critical need to evaluate and enhance OSS survivability to mitigate such risks.

In this context, survivability refers to the ability of an OSS project to sustain itself and adapt over time by addressing maintenance challenges and continuing to improve [10]. High survivability indicates active community participation, efficient issue resolution, and the capability to deliver timely updates. Conversely, low survivability suggests inadequate maintenance, leading to unresolved bugs and eventual project abandonment. A key metric for assessing OSS survivability is the issue retention rate, which quantifies the probability of unresolved issues over time [11,12]. Projects with high issue retention rates often face delayed problem-solving and reduced community engagement, whereas those with low rates exhibit efficient maintenance practices and strong community activity. This study focused on analyzing issue retention rates as a means of evaluating OSS survivability, identifying the factors influencing these rates, and offering insights to improve the long-term sustainability of OSS projects.

This study evaluated the long-term survivability of open source software (OSS) projects by focusing on the issue retention rate as a critical metric. Survivability, in this context, refers to the ability of an OSS project to sustain itself through active maintenance, effective problem resolution, and consistent community participation. The issue retention rate, which quantifies the likelihood of unresolved issues over time, offers valuable insights into a project's maintenance capacity and overall health. Projects with high issue retention rates often face delayed problem-solving and reduced community engagement, which can significantly impact their ability to evolve and remain relevant. By investigating how issue retention rates reflect survivability, this study aimed to establish a robust framework for evaluating OSS sustainability and guiding effective project management strategies.

This research applied a systematic methodology comprising data collection, analysis, and feature evaluation to achieve this objective. Data were collected from GitHub APIs and SonarCloud, capturing a wide range of information, including issue occurrences, resolution times, code complexity, and community activity metrics such as project stars, forks, and watchers. These datasets provided a comprehensive foundation for understanding how various factors contribute to the accumulation of unresolved issues. Kaplan–Meier survival analysis was used to assess issue resolution times and generate survival curves, which visualize the probability of unresolved issues persisting over time. These survival curves enable the quantification of maintenance trends and offer a clear picture of how well an OSS project manages its challenges.

To delve deeper into the relationship between project characteristics and issue retention rates, random forest models were employed. This machine learning technique is particularly suited for capturing complex, nonlinear relationships and interactions among multiple variables. By leveraging random forest models, this research evaluated the impact of community engagement, technical complexity, and project characteristics on unresolved issues. Hyperparameter optimization was conducted using RandomizedSearchCV to improve prediction accuracy, and RobustScaler was applied to minimize the effects of outliers in the dataset. These methodological refinements ensured that the model captured meaningful insights while maintaining reliability. Feature importance and permutation importance analyses revealed the relative influence of each factor, while partial dependence plots (PDP) provided a nuanced understanding of how specific features, such as project popularity or contributor count and shape issue retention rates.

This study contributes significantly to the field by presenting a data-driven framework for assessing OSS survivability. By centering the analysis on the issue retention rate, it provides a quantitative approach to understanding how unresolved issues influence long-term sustainability. This research identifies key factors that affect survivability, such as technical complexity, community engagement, and project characteristics, offering practical recommendations for improving project management. Additionally, the integration of survival analysis and random forest models demonstrates the strength of combining statistical and machine learning approaches to address the challenges of OSS maintenance. These insights not only aid project managers in formulating effective strategies but also pave the way for future research on enhancing the sustainability of OSS ecosystems.

2. Related Works

Research on OSS survivability has led to the proposal of various methods of assessing and predicting long-term success, focusing on factors such as community engagement, contributor retention, and technical complexity using data from platforms like GitHub. These studies have provided valuable insights into the factors influencing OSS projects' health, but they have often failed to incorporate the temporal aspects of issue resolution and retention, which are critical for a comprehensive evaluation of OSS survivability. This study sought to address these gaps by focusing on issue retention rates and resolution times, offering a more dynamic and time-sensitive approach to evaluating OSS survivability.

Ait et al. analyzed the survivability of OSS projects by examining early activity levels, commit records, and organizational involvement [13]. Their findings emphasized that projects with high initial activity and organizational support are more likely to survive in the long term. However, their study focused primarily on early engagement and activity continuity, without exploring how unresolved issues over time affect survivability. By applying Kaplan–Meier analysis to evaluate issue retention rates and resolution times, this study builds on their work by providing a temporal perspective that captures the influence of time-based issue dynamics on project survivability.

Han et al. developed a model to predict OSS project popularity based on features such as forks, issues, and contributors, demonstrating that popularity can drive increased engagement and better maintenance [14]. While their study focused on identifying the predictors of project visibility and engagement, it did not examine how these factors influence long-term survivability. This study extends their findings by integrating issue retention rates and resolution metrics, offering a broader understanding of how community activity and project characteristics impact both short-term engagement and long-term survivability.

Zhou et al. proposed GitEvolve, a system for predicting user interactions and event patterns in OSS projects using multitask models and graph-based embeddings [15]. Their approach models long-term growth and user contributions by predicting event types and

timing, but it does not explore how issue-specific factors influence project survivability. In contrast, this study combined Kaplan–Meier analysis with machine learning to evaluate how temporal metrics like issue resolution times and retention rates affect project health, providing actionable insights for improving OSS maintenance practices.

Eluri et al. utilized MLP models to predict the survivability of OSS components by identifying software complexity, user base, and developer participation as key factors [16]. Their research highlighted the role of developer engagement in maintaining OSS, but it lacked a detailed examination of how individual factors influence survivability. This study addresses this limitation by employing feature importance analysis and PDP to quantify the relative impact of each feature, offering project managers a clearer understanding of how specific characteristics drive survivability.

Decan et al. focused on contributor activity patterns to predict the likelihood of contributor inactivity using survival analysis, developing the GAP tool to monitor and predict contributor engagement [17]. While their research emphasized the importance of retaining contributors for project success, it did not assess broader project features like issue retention or resolution rates. This study distinguishes itself by prioritizing these metrics as critical indicators of OSS survivability, providing a more comprehensive framework for evaluating and improving long-term project health.

To address the limitations of prior research, this paper introduces a novel framework that combines Kaplan–Meier survival analysis and machine learning to evaluate issue retention rates, resolution times, and their effects on OSS survivability. Prior studies, such as those by Ait et al., Han et al., Zhou et al., Eluri et al., and Decan et al., have provided valuable insights into OSS survivability by examining factors like community engagement, contributor retention, technical complexity, and project popularity. However, these studies have often overlooked the temporal dynamics of issue resolution and retention, which are crucial for a comprehensive evaluation of OSS survivability. For instance, Ait et al. focused on early engagement and organizational support but did not address unresolved issues over time, while Zhou et al.'s GitEvolve system models user interactions but does not examine issue-specific factors. This study addresses these gaps by incorporating issue retention rates and resolution times into the analysis, offering a time-sensitive perspective on OSS survivability. By doing so, it not only enhances the understanding of key factors influencing survivability but also provides practical recommendations for optimizing maintenance practices, fostering community engagement, and ensuring the long-term health of OSS projects.

3. Analyzing the Relationship Between OSS Survivability and Features

The purpose of this study was to quantitatively evaluate the issue retention rate's impact on the survivability of OSS projects and to identify the key factors influencing this rate. To achieve this, the research process was structured into four stages: data collection, issue retention rate measurement, analysis model construction, and feature impact analysis. The following sections provide a detailed explanation of each stage.

3.1. Experimental Setup

3.1.1. Issue Retention Rate and Survivability

The issue retention rate is a key metric for evaluating the survivability of OSS projects, serving as an important indicator of project management and community activity. OSS projects depend on voluntary contributions, making the speed of issue resolution and retention rate effective measures of their survivability. The issue retention rate refers to the probability that an issue remains unresolved over a specific period, reflecting the maintenance capacity and activity levels of the community.

This study assumed that OSS survivability can be effectively assessed through issue activity, specifically the retention rate of unresolved issues. While this assumption was not directly tested, it underpins the analytical framework, with conclusions contingent on its validity.

The significance of the issue retention rate is supported by prior studies. Alshomali et al. [11] emphasized that unresolved issues lead to the accumulation of technical debt, undermining a project's long-term potential. Their research highlights that prolonged unresolved issues can erode trust between users and developers, increasing the likelihood of project abandonment. Similarly, Jarczyk et al. [12] analyzed collaborative processes on GitHub and found that faster issue resolution is strongly linked to effective communication and division of responsibilities among contributors. These factors collectively enhance project stability and survivability.

Avelino et al. [18] further noted that higher issue retention rates are associated with decreased maintenance efficiency and reduced community activity, ultimately jeopardizing project sustainability. Their findings support the idea that unresolved issues are critical indicators of maintenance challenges. Dabbish et al. [19] explored the transparency and collaborative structure of GitHub, demonstrating how issue activity can foster community engagement and facilitate problem-solving. These studies collectively highlight the utility of issue retention rates as a proxy for evaluating OSS project health and survivability.

Building on these findings, this study focused on issue retention rates as a core metric for assessing OSS survivability. GitHub issues were employed as the primary data source, providing insights into a project's maintenance status and future development potential. Kaplan–Meier survival analysis was applied to capture temporal changes in retention rates, offering a dynamic and comprehensive evaluation of how unresolved issues affect project survivability.

3.1.2. Features for OSS Survivability Analysis

To predict and analyze OSS survivability, a variety of features were selected. OSS survivability is influenced by diverse human and technical factors, such as user retention, irreplaceability, contributor and user counts, and code quality. This reflects the fact that OSS is not merely a technical product but an ecosystem that relies on continuous interest and participation from users and contributors. Therefore, analyzing OSS survivability requires consideration of various aspects, including code quality, community activity, and user interest [14,16].

In this study, both external factors like project popularity and internal factors such as contributors and project complexity were analyzed to understand their relationship with OSS survivability. As such, features such as stars, watchers, and contributor count, which indicate project popularity and activity, were selected. In addition, inspired by studies like Eluri et al., LoC (Line of Code) and cyclomatic complexity awerere included to examine how complexity impacts survivability. The selected features are summarized in Table 1.

Table 1. Features selected for analysis.

Feature	Description
Star	Level of user interest in the project
Watch	Number of users interested in project updates
Contributor	Number of developers contributing to the project
Fork	Number of users replicating the project for further development
LoC	Size of the project
Cyclomatic Complexity	Complexity of the project code
Age	Time since the project was created

Stars are one of the most common indicators of a repository's popularity. They show how much attention a project receives and how well known it is among users. Borges et al. analyzed how the number of stars impacts project success and popularity, finding that projects with more stars tend to have higher community participation and maintenance activity [20,21]. Similarly, Han et al. reported that the number of unresolved issues affects project popularity [14]. This suggests that increased interest in a project leads to higher participation from new contributors and users. Therefore, the number of stars is selected as a feature, as it reflects sustained community interest and contributions, which enhance the project's long-term survivability.

Watch represents the number of users who subscribe to notifications about project updates, indicating sustained user interest in changes to the project. Han et al. demonstrated how the number of watchers contributes to project maintenance activity. Projects with more watchers are more likely to have active maintenance, as watchers tend to be highly responsive to significant changes such as bug fixes or feature updates. Thus, the number of watchers is chosen as a key feature for analyzing OSS survivability.

Contributors refer to the number of developers actively contributing code or documentation to a project. Zhou et al. found that the contributor count plays a crucial role in project growth and maintenance [15]. Projects with a larger number of contributors tend to improve and stabilize more quickly. Studies by Eluri et al. and Han et al. have also confirmed that a higher contributor count is associated with increased survivability, as it implies faster issue resolution and abundant resources for developing new features [14,16]. Consequently, the contributor count is selected as a feature directly linked to OSS maintenance capacity.

Fork refers to instances where users replicate a project to their own GitHub repositories to further develop it or propose modifications separate from the original. The number of forks reflects how much the project influences developers, both contributors and noncontributors. Ait et al. concluded that projects with higher fork counts exhibit greater potential for maintenance and contributions. Since more forks indicate a higher likelihood of future contributions and modifications, the number of forks was chosen as a feature.

LoC represents the size and complexity of a project. Higher LoC values can make the codebase more challenging to understand, requiring greater effort for maintenance. Eluri et al. analyzed the impact of the LoC on OSS project maintenance, finding that while a higher LoC indicates larger-scale projects, it also requires more resources for maintenance [16]. Thus, the LoC was selected as a feature to represent project complexity and the associated maintenance costs.

Cyclomatic complexity quantifies code complexity based on the number of decision points and branches in the code. Ebert et al. explained that higher cyclomatic complexity increases the difficulty of maintenance and can negatively affect project stability [22]. Code with higher complexity is more prone to errors and requires additional time for resolution, which may impact survivability. For this reason, cyclomatic complexity was included as a feature.

Project age measures how long a project has been maintained and developed since its creation. Decan et al. investigated the impact of project age on long-term survivability, noting that while older projects signify prolonged maintenance, they may also face declining contributor interest or accumulated technical debt [17]. Older projects can have higher survivability but require consistent maintenance efforts. Therefore, project age was selected as a feature.

3.1.3. Data Collection

This study collected the necessary data from various OSS projects to analyze seven key features and the target variable, OSS survivability. The data collection process involved

using Python's PyGitHub library to access project data via the GitHub API. Additionally, SonarCloud, a tool for analyzing code quality, was used to gather supplementary code-related data.

Features such as stars, contributor count, forks, and watchers were collected from the GitHub API using the PyGitHub library. These indicators were used to evaluate project popularity, community activity, and development engagement. These data points represented the repository information available on GitHub and served as metrics for assessing project growth and maintenance capacity.

Since the GitHub API alone did not provide sufficient information for analyzing LoC and cyclomatic complexity, these metrics were collected using SonarCloud. SonarCloud analyzed the size and complexity of the code, offering critical insights into the project's maintainability.

To measure OSS survivability, issue data was collected from the issue tab on GitHub using PyGitHub. The data included the issue creation date, resolution date (or the current date if unresolved), and resolution status. The issue creation date represents when the issue was initially reported, while the resolution date indicates when it was resolved. For unresolved issues, the collection date was used as the resolution date. The resolution status captures whether the issue is closed on GitHub. Projects with fewer than 20 issues were excluded from the analysis to ensure data reliability.

Through this data collection process, 218 OSS projects were ultimately selected for this study. These projects varied in size and community activity, providing sufficient data for analyzing OSS survivability. The distribution of the collected data is visualized in Figure 1. Figure 1 shows the distribution of the LoC and other features. These visualizations reveal significant differences in the features' magnitudes and the presence of outliers, highlighting the need for models unaffected by feature scaling during the analysis phase.

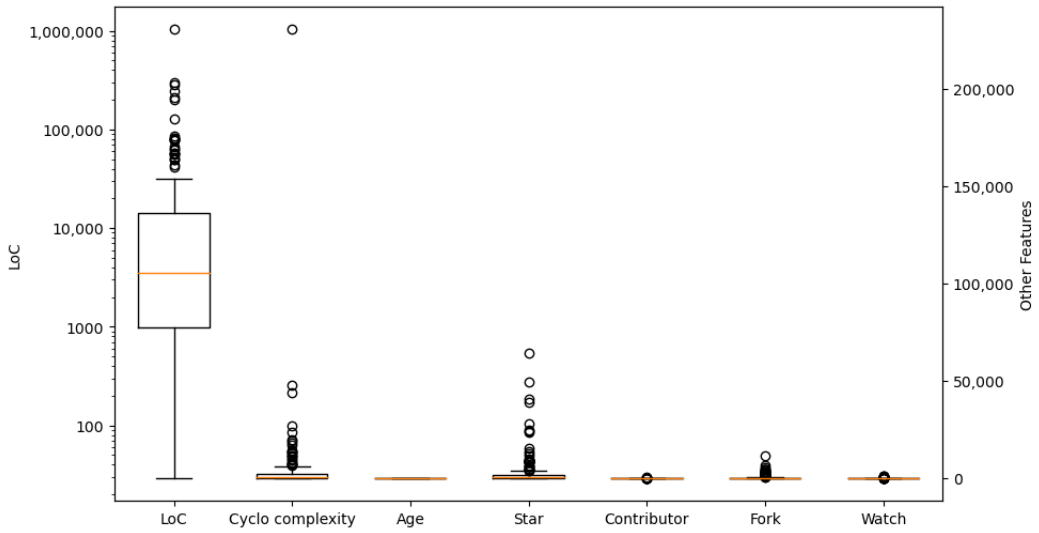


Figure 1. Distribution of collected features.

3.2. Measuring Issue Retention Rate of OSS

3.2.1. Kaplan–Meier Survival Function

The Kaplan–Meier survival function, also known as the product limit or product-limit method, was used to estimate survival probabilities from survival data that included censored values [23]. This method involves the following steps:

(a) Sort the data in order of time.
(b) At each time t_i, identify the number of events (deaths) d_i and the number of censored observations c_i.

(c) Let n_i be the number of projects at risk just before time t_{i-1}, calculated using the relation Equation (1):

$$n_i = n_{i-1} - d_{i-1} - c_{i-1} \tag{1}$$

(d) Estimate the conditional probability of survival at each time t_i:

$$P(T > t_i | T > t_{i-1}) = \frac{n_i - d_i}{n_i} = 1 - \frac{d_i}{n_i} \tag{2}$$

(e) Use the above conditional probabilities to calculate

$$\hat{S}(t_i) = \prod_{j=1}^{i} \left(1 - \frac{d_j}{n_j}\right) \tag{3}$$

Here, $\hat{S}(t_i)$ represents the probability of surviving until time t_i, which is obtained by multiplying the survival probabilities of all previous times.

3.2.2. Survival Function of OSS

To quantitatively evaluate the issue retention rate of OSS projects, this study employed the Kaplan–Meier estimation method. Traditionally used in fields such as medicine to measure survival rates by setting death as an event or in business to assess retention rates with customer churn as the event, this method was applied here to measure issue retention rates over time. By generating survival functions for entities, the Kaplan–Meier method provideed insights into the occurrence rates of events over time, enabling the measurement of issue retention rates and durations for each OSS project. For this study, issue resolution was treated as the event, and survival functions were generated for each project.

The Kaplan–Meier method was implemented using Python's scikit-learn library, and the survival curves for each OSS project were estimated based on the following data:

- **Issue duration:** This is the time between the creation and resolution of an issue. It represents how long the issue remains unresolved, calculated as `issue resolution time − issue creation time`.
- **Event observed (resolution status):** This indicates whether the issue has been resolved. Resolved issues were marked as `1`, representing an event occurrence, while unresolved issues were marked as `0`, treated as censored data.

When applying the Kaplan–Meier method, it is crucial to consider the data collection period, as the survival function results may vary depending on the duration of issue data used. Shorter data periods reflect recent patterns, while longer periods provide a clearer understanding of long-term issue resolution trends in projects. To determine the effect of different data durations on the accuracy of the analysis, this study tested various data collection periods, such as one month and one year.

The survival function generated with a 90-day data duration is shown in Figure 2. Using this function, the issue retention rate at $t = 15$ was extracted. The choice of $t = 15$ was motivated by the focus on identifying the likelihood of issues being resolved quickly. Specifically, $t = 15$ corresponds to the typical duration of a single sprint in the Scrum methodology, providing a practical benchmark for early issue resolution. The distribution of the measured issue retention rates is shown in Figure 3. This figure reveals that the distribution of issue retention rates was imbalanced, highlighting the need for analytical models suitable for imbalanced data.

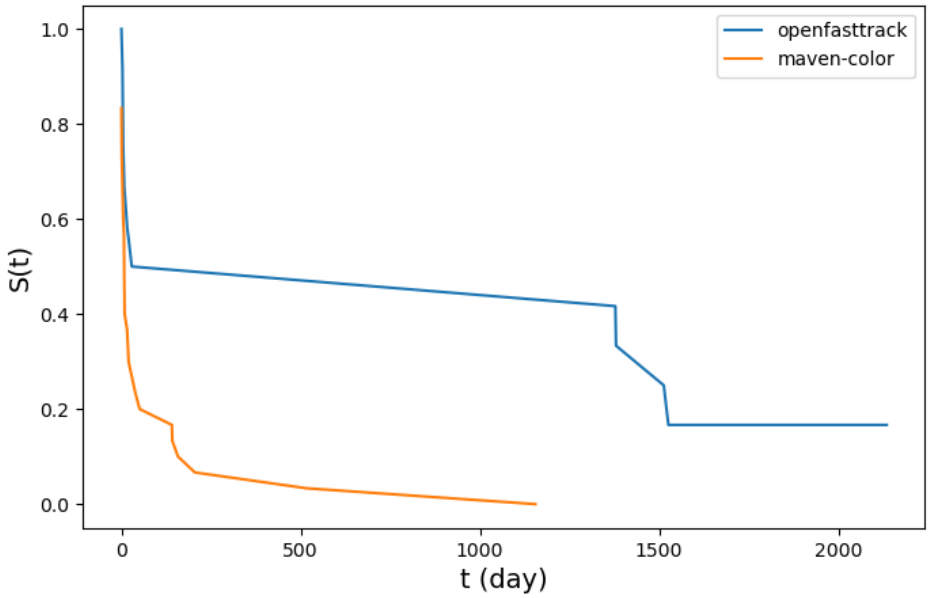


Figure 2. Issue survival function generation for OSSs.

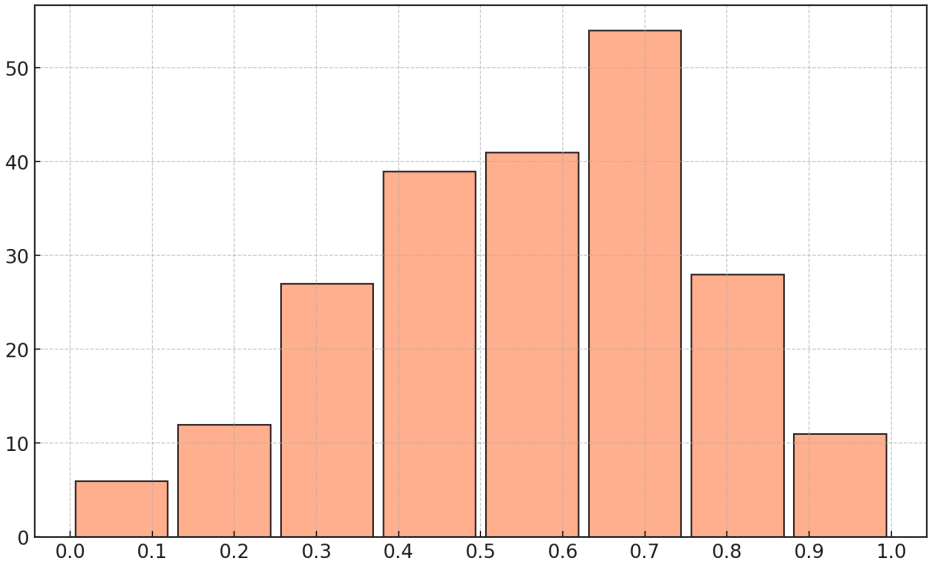


Figure 3. Measured results of the issue retention rate.

3.3. Building the Analysis Model

This study aimed to build an analysis model to predict OSS survivability by adopting the random forest method. Random forest demonstrates robust performance even when dealing with imbalanced data distributions. As shown in Figure 1, the collected data were significantly imbalanced, particularly in the LoC feature, which exhibited a much larger scale compared to other features such as star or contributor. This imbalance could lead to over-representation or under-representation of certain features during model training, potentially biasing predictions. Random forest mitigated this by employing bootstrap sampling, where individual decision trees are trained on randomly selected subsets of the data, ensuring a balanced reflection of diverse feature distributions.

Additionally, random forest is highly resilient to nonlinear relationships. Factors influencing OSS survivability often involve complex interactions and nonlinear dependencies that traditional linear- or polynomial-based models struggle to capture. Random forest partitions the data into multiple segments through its tree-based structure and learns

independently within each segment, effectively capturing these nonlinear relationships. For example, while polynomial regression models high-order polynomials globally, random forest creates localized partitions that better reflect the underlying data dynamics.

We further illustrate random forest's mechanisms:

(1) **Bootstrap Sampling**: Each decision tree is trained on a subset of the original data,

$$D_t = \{(X_r, y_r)\}_{r=1}^{N_t}, \quad N_t \leq N$$

where N is the total number of samples in the dataset D, and N_t is the number of samples in the bootstrapped dataset D_t. Here, X_r represents the features, and y_r is the corresponding target value for the r-th sample.

(2) **Feature Sampling**: At each node, a random subset of features is selected:

$$F_t \subseteq F, \quad |F_t| = \texttt{max_features}$$

where F represents the set of all feature types in the dataset, and F_t is the subset of features randomly selected for the current node. This ensures that no single feature dominates the model, promoting diversity in the decision trees.

(3) **Prediction Aggregation**: Final predictions are obtained by averaging the outputs of all decision trees:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} h_t(x),$$

where T is the total number of trees, and $h_t(x)$ represents the prediction of tree t.

Given these advantages, random forest is expected to be more suitable for datasets characterized by imbalanced distributions and nonlinear relationships, making it a powerful tool for predicting OSS survivability. To validate this, the performance of random forest was compared with that of the polynomial regression model.

3.3.1. Model Performance Optimization

To optimize model performance, this study employed RandomizedSearchCV for hyperparameter tuning. RandomizedSearchCV selects and evaluates random combinations of hyperparameters over a predefined number of iterations, balancing search efficiency and computational cost. This method was chosen over GridSearchCV, which exhaustively evaluates all combinations but incurs significant computational overhead, and Bayesian Optimization, which is efficient but complex to set up.

The hyperparameter tuning process is outlined in Algorithm 1. The algorithm iteratively selects hyperparameter combinations, evaluates them using cross-validation, and identifies the combination yielding the lowest mean squared error (MSE) as optimal. For detailed hyperparameter ranges and optimal values, refer to Tables 2 and 3.

Table 2. Hyperparameter combination settings.

Hyperparameter	Value Range	Description
`n_estimators`	{100, 200, 300, 400, 500}	Number of decision trees
`max_depth`	{None, 10, 20, 30, 40}	Maximum depth of the tree
`min_samples_split`	{2, 5, 10}	Minimum samples required to split a node
`min_samples_leaf`	{1, 2, 4}	Minimum samples required at each leaf node
`max_features`	{'auto', 'sqrt', 'log2'}	Number of features to consider for node split
`bootstrap`	{True, False}	Whether to use bootstrap sampling

Table 3. Optimal hyperparameter settings.

Hyperparameter	Optimal Value
n_estimators	400
max_depth	10
min_samples_split	5
min_samples_leaf	1
max_features	log2
bootstrap	True

Algorithm 1 Efficient hyperparameter optimization using RandomizedSearchCV.

Define the range of hyperparameters Θ_{range}
Initialize $k = 50$: Number of iterations, and $F = 5$: Number of folds for cross-validation
for $p = 1$ to k **do**
  Randomly sample a hyperparameter combination $\theta_p \sim \text{Uniform}(\Theta_{\text{range}})$
  for $q = 1$ to F **do**
    Train the model on $F - 1$ folds and validate on the remaining fold
    Compute mean squared error (MSE_q)
  end for
  Calculate cross-validated score: $\text{CV_Score}_p = \frac{1}{F} \sum_{q=1}^{F} \text{MSE}_q$
end for
Identify θ^* with the lowest CV_Score_p
Return θ^*

3.3.2. Model Performance Evaluation

To evaluate model performance, this study used R^2, adjusted R^2, MSE, and residual plots. R^2 and adjusted R^2 assess how well a model fits the training data, with adjusted R^2 accounting for the number of features to provide a more robust evaluation. MSE measures prediction accuracy on unseen test data, reflecting how closely a model's predictions align with actual values. Residual plots are utilized to visually inspect the distribution of prediction errors, helping to identify potential issues such as bias or overfitting. By combining these metrics, this study performed a comprehensive evaluation of the OSS survivability prediction model.

4. Results and Analysis

This section evaluates the performance of the constructed analysis model and examines the impact of the features on the issue retention rate. To achieve this, feature importance analysis and PDP analysis were employed. Feature importance analysis identified how significantly each feature contributed to the prediction model, allowing for the evaluation of the relative importance of individual features in predicting OSS survivability. PDP analysis, on the other hand, provided a visual representation of the relationship between specific features and the predicted outcomes, enabling a deeper understanding of how variations in feature values influenced the issue retention rate. These methods collectively offered insights into which features played a critical role in OSS survivability and how they interacted with the model predictions.

4.1. Model Performance Evaluation and Comparison

This section evaluates whether the random forest model was appropriately constructed for predicting OSS project survivability and examines its suitability through a performance comparison with the polynomial regression model. The main performance metrics of both models are summarized in Table 4.

The polynomial regression model was constructed using Lasso regression, which automated both feature selection and the degree setting for each feature. The model's

performance was evaluated through 5-fold cross-validation, ensuring the selection of the most optimal model for the given dataset.

Table 4. Performance evaluation results of random forest and polynomial regression models.

Metric	Random Forest	Polynomial Regression
R^2	0.716	0.19
Adjusted R^2	0.704	1.38
MSE	0.043	0.07
Key Characteristics	Effectively captures nonlinear interactions and complex patterns in the data	Limited by the inability to fully capture nonlinear characteristics due to the constraints of polynomial degrees

The random forest model demonstrated superior performance, as indicated by the various performance metrics and residual distribution. The R^2 value for the random forest model was 0.716, meaning that the model explained approximately 71.6% of the data variance. The adjusted R^2 value of 0.704 also indicated stability, suggesting that the model effectively captured the complexity and nonlinearity of the data without overfitting. The MSE value was 0.043, signifying a small error between the predicted and actual values and demonstrating high prediction accuracy.

Additionally, as shown in Figure 4, the residual distribution of the random forest model was generally uniform, indicating that the model effectively learned various data patterns. However, as the target value approached zero, the data became sparse, resulting in increased errors, which represents a limitation of the random forest model. Despite this, the uniform residual distribution across most ranges suggests that the model provides consistent predictive performance.

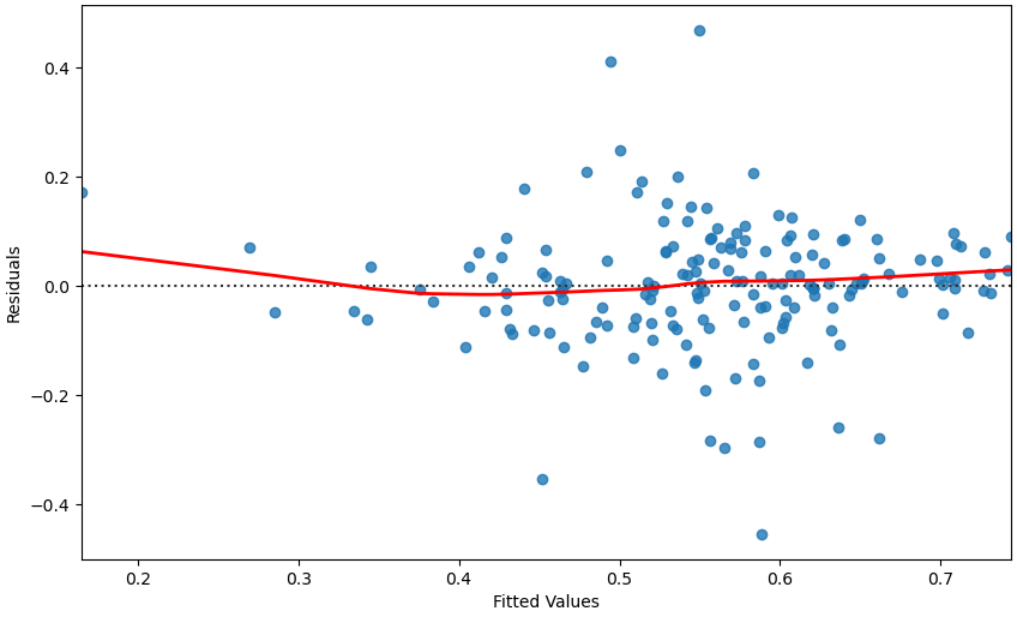


Figure 4. Residual graph of random forest model (Red Line: Smoothed Trend, Blue Dots: Residuals).

The performance of the random forest model was compared with that of the polynomial regression model to identify differences in predictive accuracy. Despite optimization using the Lasso technique, the polynomial regression model exhibited limitations, with an R^2 value of 0.19, indicating its inability to adequately capture the nonlinearity of the data. The adjusted R^2 value was -1.38, an invalid result, reflecting the model's poor predictive

capability. Additionally, the MSE value of the polynomial regression model was 0.07, higher than that of the random forest model, indicating lower predictive accuracy.

As illustrated in Figure 5, the residual distribution of the polynomial regression model showed a tendency for residuals to concentrate randomly in specific regions, reflecting the model's struggle to capture the nonlinearity of the data. In contrast, the random forest model exhibited uniformly distributed residuals, demonstrating its ability to learn a diverse range of data patterns effectively. These performance metrics and residual distribution comparisons confirmed that the random forest model was more suitable than the polynomial regression model for predicting OSS survivability.

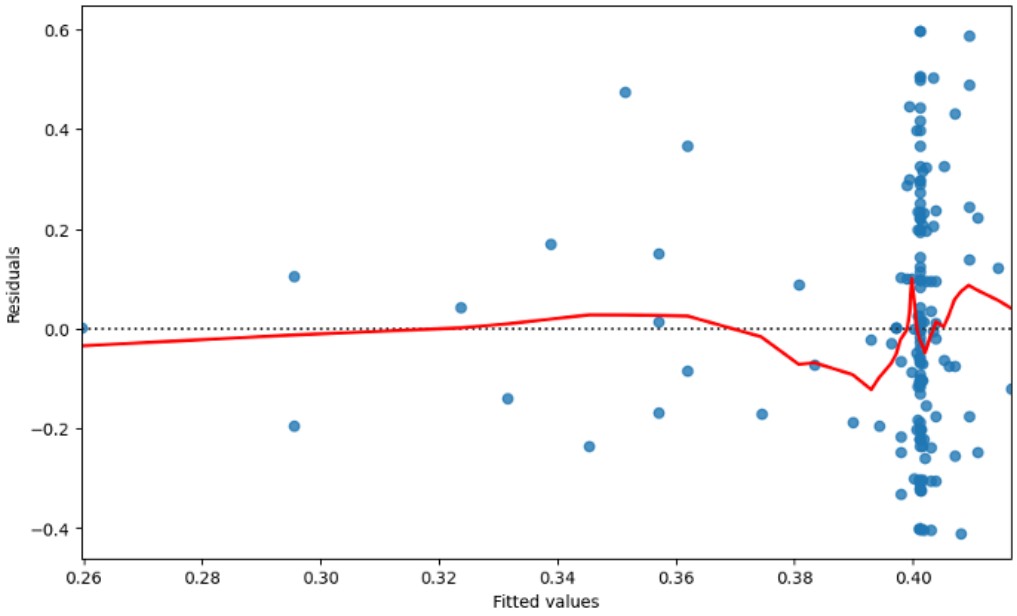


Figure 5. Residual graph of polynomial regression model (Red Line: Smoothed Trend, Blue Dots: Residuals).

4.2. Analysis of Feature Impacts

This section analyzes the impact of the selected features on the performance of the OSS survivability prediction model. By employing general feature importance, permutation feature importance, and PDP analyses, this study aimed to understand how each feature contributed to the model and identify the key factors that play a significant role in predicting survivability. This analysis helped to uncover the relative influence of the features on survivability and derive actionable insights for improving it.

Feature importance analysis identifies features that significantly affect prediction accuracy. Permutation feature importance, based on the random forest model, refines this by measuring performance drops when feature values are randomly shuffled. This method highlights features the model depends on and can uncover noise-inducing features, allowing for targeted improvements to the model.

PDP analysis allows a visual examination of how changes in a specific feature impact a model's predictions while holding other features constant. This study used PDP to explore relationships between key features and OSS survivability. For instance, it may show that increasing contributors boosted survivability predictions, while excessive code complexity reduced them. These findings help determine optimal feature levels to improve survivability.

4.2.1. Feature Importance Analysis

The feature importance analysis, conducted using the random forest model, identified the features with the most significant impact on the issue retention rate of OSS projects.

As shown in Figure 6, the permutation feature importance analysis revealed that stars had the highest importance. Notably, the exclusion of contributors as a feature had a minimal impact on prediction performance, suggesting that development contributions did not strongly influence issue retention rates. Project age and cyclomatic complexity also showed low importance, indicating their relatively minor influence on survivability.

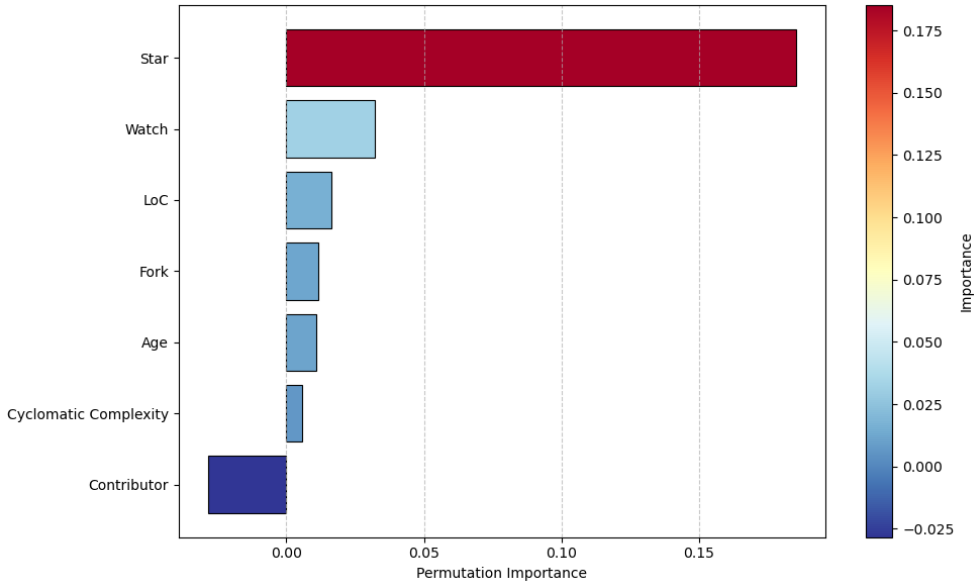


Figure 6. Feature Importance Analysis.

The analysis confirmed that OSS project popularity and community interest, represented by features such as stars and watchers, play critical roles in survivability. In contrast, features like contributors, project age, and cyclomatic complexity appear to have relatively lower impacts. Particularly, project age may negatively affect prediction performance. Despite the lower importance of some features, PDP analysis was necessary to further interpret their effects on survivability. Certain features may hold significant relevance in specific scenarios and can help uncover more nuanced relationships in predicting issue retention rates. Therefore, the additional analysis using PDP is outlined in Section 4.2.2.

4.2.2. PDP Analysis

This section examines the specific effects of each feature on the issue retention rate of OSS projects using PDP analysis. PDP analysis was used to visually demonstrate how the issue retention rate change as the values of individual features increase. A higher y-axis value in the graph indicates an increase in the issue retention rate, suggesting a potential decline in survivability.

Stars, a key indicator of project popularity, exhibit a significant decrease in the issue retention rate as the x-axis value increases from 0 to 4000,as shown in Figure 7a. Specifically, the y-axis value rapidly drops from approximately 0.62 to 0.52, indicating that less popular projects are more likely to have unresolved issues. After exceeding 4000 stars, the issue retention rate stabilizes, showing minimal changes. This suggests that projects reaching a certain level of popularity tend to achieve stable issue resolution rates.

For watch, as shown in Figure 7b, the y-axis value increases from approximately 0.54 to 0.58 as the x-axis value grows from 0 to 50. This implies that heightened attention to the project may correlate with an increased likelihood of unresolved issues. Beyond 50 watchers, the issue retention rate tends to stabilize.

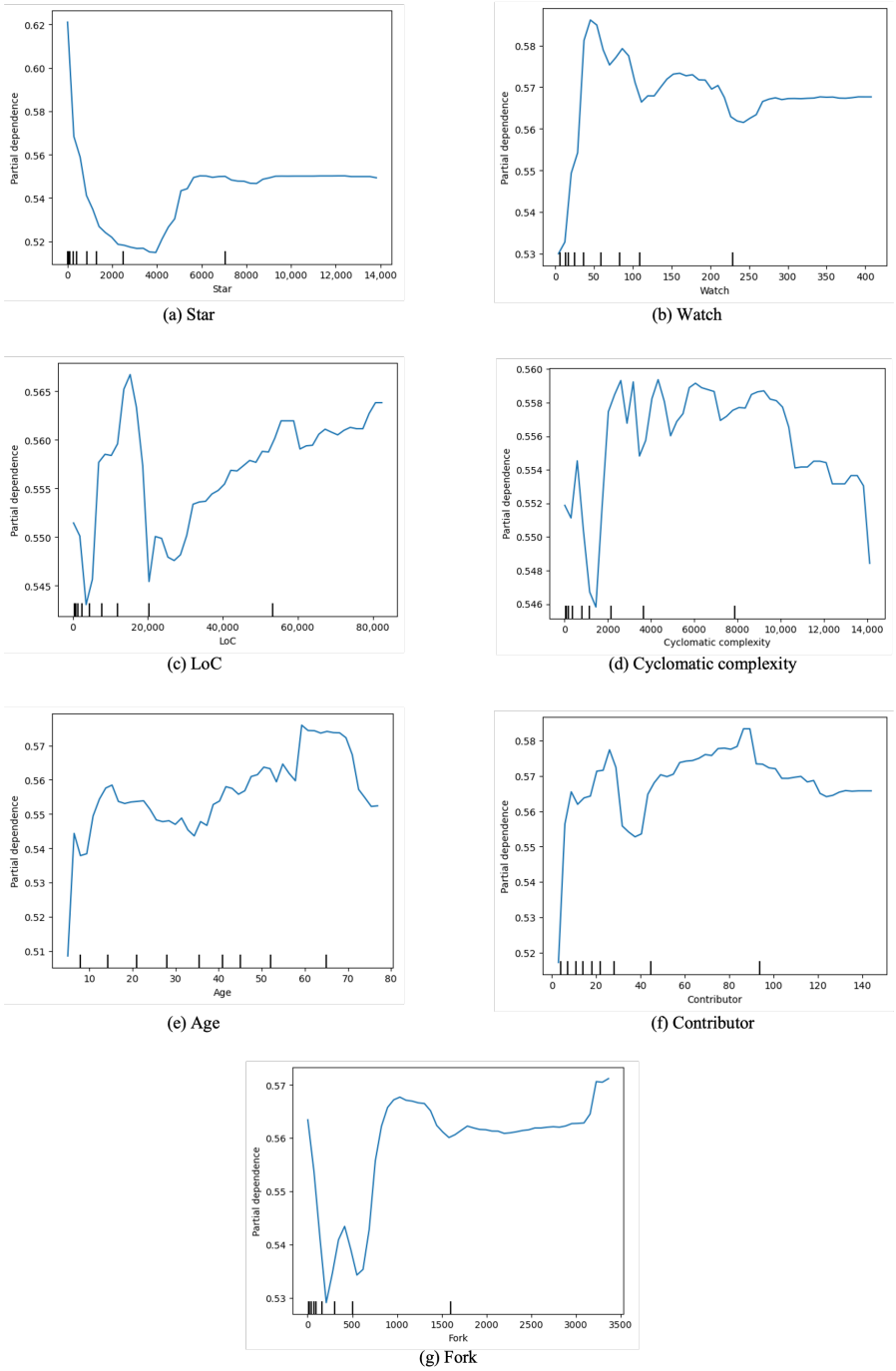


Figure 7. Partial dependence plot analysis results for features.

The LoC, representing the size of the project's codebase, shows an upward trend in the issue retention rate as the x-axis value increases from 0 to 20,000, as shown in Figure 7c. The y-axis value rises from approximately 0.545 to 0.565, indicating that larger codebases may be more prone to unresolved issues. Beyond 20,000 LoC, the retention rate fluctuates slightly but generally remains between 0.550 and 0.565, suggesting increased maintenance challenges with larger codebases.

The cyclomatic complexity, a measure of code complexity, exhibits a sharp increase in the issue retention rate as the x-axis value grows from 0 to 2000, with the y-axis rising from 0.545 to 0.562 (Figure 7d. However, beyond 6000, the rate stabilizes or slightly decreases, suggesting that projects with high initial complexity may establish better maintenance structures over time.

The project age demonstrates an increasing issue retention rate as the x-axis value grows from 10 to 20, with the y-axis rising from approximately 0.52 to 0.54 (Figure 7e). Around 60 months, the y-axis value increases to approximately 0.55–0.58, indicating that older projects may face challenges in resolving issues promptly, likely due to maintenance difficulties or technical debt.

Contributors displays a similar pattern. As shown in Figure 7f, the y-axis value decreases slightly when the x-axis value increases from 20 to 40, indicating a temporary improvement in issue resolution. However, beyond 40 contributors, the y-axis begins to rise again, suggesting that larger contributor counts do not necessarily enhance a project's problem-solving capacity. Maintaining an optimal size may be more beneficial for effective management.

Forks, a measure of community contributions, exhibits a declining issue retention rate as the x-axis value grows from 0 to 500, with the y-axis decreasing from approximately 0.56 to 0.53 (Figure 7g). This indicates that higher contributions generally accelerate issue resolution. However, beyond 500 forks, the retention rate stabilizes, and after 2000 forks, it slightly increases again, suggesting that excessive contributions may introduce additional complexities, potentially hindering issue resolution.

4.2.3. Insights into Feature Influence on OSS Survival

The combined results of the feature importance and PDP analyses in this study indicate that OSS project survivability is determined by the complex interplay of various factors. While community activity and popularity positively contribute to reducing issue retention rates and enhancing survivability, project age and code complexity could negatively impact survivability. Moreover, some features tend to exert significant influence only within specific ranges, highlighting the need for additional interpretation.

From the perspective of community activity, features such as stars and watchers, which reflect project popularity, were found to correlate with faster issue resolution and lower issue retention rates. For instance, the PDP analysis of stars revealed a steep decline in the issue retention rate as the star count increased up to 4000, indicating that greater community attention and support attract more contributors, fostering quicker issue resolution and higher long-term survivability. Conversely, the PDP analysis of forks showed an increase in issue retention rates as fork counts rose, suggesting that while more contributors might join the project, difficulties in coordination and management complexity could slow issue resolution.

LoC and contributor count also appeared as factors influencing OSS project survivability. For LoC, larger codebases were associated with slightly higher issue retention rates, indicating that increased code volume adds maintenance burdens and potentially delays issue resolution. Contributor count showed an initial increase in issue retention rates as contributor numbers rose but stabilized beyond a certain threshold. This finding suggests that while an increase in contributors can initially benefit project maintenance, exceeding a certain level may introduce challenges in coordination and management.

Additionally, code complexity and project age negatively impact survivability. Higher cyclomatic complexity and older project age are linked to higher issue retention rates. Increased code complexity likely hampers maintenance efforts, as seen in the upward trend in issue retention rates with rising cyclomatic complexity. Similarly, older projects show a tendency for unresolved issues to persist longer, indicating that aging projects may face growing maintenance and issue resolution challenges. These findings emphasize the significant role of obsolescence and management burdens in affecting OSS survivability.

To enhance the long-term survivability of OSS projects, strategies focusing on fostering community participation and increasing project popularity in the early stages are essential.

Additionally, the systematic management of code complexity, codebase size, contributor count, and project aging is necessary. Early-stage strategies such as community engagement initiatives and marketing play a critical role in attracting attention, while ensuring balanced growth through complexity management and maintaining a sustainable level of contributors and code size will support long-term survivability. These comprehensive interpretations provide actionable guidelines for OSS managers, offering a meaningful foundation for devising strategies to ensure the success and sustainability of future OSS projects.

5. Conclusions

This study focused on analyzing issue retention rates to effectively evaluate the survivability of OSS projects. OSS projects thrive on the voluntary participation of contributors and users, but sustaining them in the long term requires active community involvement and problem-solving capabilities. Thus, the duration and retention rate of issues serve as critical indicators for assessing the long-term viability of OSS projects. This research measured issue retention rates based on resolution status and resolution time to analyze the long-term survivability of OSS projects.

This research was conducted through the following experimental stages: First, data from 218 OSS projects were collected using the GitHub API. This dataset included various project characteristics, such as the number of stars, forks, contributors, and code complexity. Additionally, issue creation and resolution timestamps were gathered, enabling the measurement of issue retention durations.

Second, Kaplan–Meier survival analysis was employed to quantify the distribution of unresolved issue durations, which provided insights into project survivability. Kaplan–Meier survival analysis visualized the proportion of unresolved issues over time, thereby revealing trends in issue resolution rates within OSS projects. The issue retention rates derived from Kaplan–Meier survival functions allowed for a quantitative assessment of each project's maintenance status and community activity.

Third, a random forest model was applied to analyze the impact of collected features on issue retention rates. The random forest model effectively captured the nonlinear relationships among features, identifyingthe key factors influencing issue retention. To optimize the model's performance, hyperparameter tuning was conducted using RandomizedSearchCV.

The experimental results identified stars and watchers as the most influential factors of OSS survivability. PDP analysis revealed that as the number of stars increased, community engagement improved, leading to faster issue resolution and lower retention rates. For instance, as the number of stars increased from 1 to 4000, the issue retention rate decreased significantly from approximately 0.62 to 0.52. This finding indicates that strong community interest contributes to enhanced issue resolution and long-term project survivability. Conversely, the number of forks showed a tendency to increase issue retention rates, rising from approximately 0.53 to 0.56 when the number of forks exceeded 500. This suggests that while a high number of contributors can indicate interest, it may also introduce challenges such as coordination difficulties and management complexity.

Other variables, such as lines of code (LoC) and the number of contributors, also influenced OSS survivability. LoC showed an upward trend in issue retention rates, increasing from approximately 0.545 to 0.565 as the code size exceeded 20,000 lines. This implies that larger codebases have greater maintenance burdens, slowing issue resolution. The number of contributors initially had a positive impact, reducing issue retention rates, but after reaching approximately 40 contributors, the effect plateaued, suggesting that excessive contributor numbers can lead to management difficulties.

Code complexity and project age were also identified as factors negatively impacting survivability. Both metrics showed an increase in issue retention rates as their values

grew. For example, a cyclomatic complexity exceeding 2000 led to an increase in retention rates from approximately 0.552 to 0.565. Similarly, the project age showed a sharp rise in retention rates beyond 60 months, highlighting the challenges faced by aging projects and the need for systematic management.

This study makes significant contributions by introducing a novel methodology for quantitatively evaluating OSS survivability based on issue retention rates. The Kaplan–Meier survival analysis provides a visual representation of OSS maintenance capabilities, while the random forest model empirically identifies key factors influencing survivability. These findings offer OSS managers actionable insights for developing long-term maintenance strategies, emphasizing the importance of community engagement and the need for systematic management of code complexity, contributor numbers, and project aging.

Several limitations should be noted. First, the data were restricted to GitHub, excluding OSS projects from other platforms. This limits the scope of analysis to the GitHub ecosystem. Second, this study focused on selected project characteristics, community activity, and technical factors, potentially overlooking external influences such as legal considerations or competition with commercial software. Third, while the Kaplan–Meier model effectively analyzed average trends in issue retention rates, it lacked the granularity to capture detailed changes over specific time intervals or irregular distributions.

To address these limitations, future research will incorporate data from diverse platforms and refine models to account for more complex feature interactions. Advanced techniques, such as combining machine learning and deep learning, will be employed to analyze the intricate nonlinear relationships and examine how issue types and severity influence retention rates. These advancements aim to provide a comprehensive analysis of the OSS ecosystem and enhance the precision of survivability predictions.

Author Contributions: Conceptualization, S.P. and G.K.; methodology, S.P.; resources, S.P.; software, S.P.; validation, G.K.; writing—original draft preparation, S.P.; writing—review and editing, G.K.; visualization, S.P.; supervision, G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00122, Safety Analysis and Verification Tool Technology Development for High Safety Software Development).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Feller, J.; Fitzgerald, B. *Understanding Open Source Software Development*; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 2002.
2. Lee, G.K.; Cole, R.E. From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organ. Sci.* **2003**, *14*, 633–649. [CrossRef]
3. Shah, S.K. Motivation, governance, and the viability of hybrid forms in open source software development. *Manag. Sci.* **2006**, *52*, 1000–1014. [CrossRef]
4. Cheng, J.; Guo, J.L.C. Activity-based analysis of open source software contributors: Roles and dynamics. In Proceedings of the 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), Montreal, QC, Canada, 27 May 2019; pp. 11–18.

5. Forte, A.; Lampe, C. Defining, understanding, and supporting open collaboration: Lessons from the literature. *Am. Behav. Sci.* **2013**, *57*, 535–547. [CrossRef]
6. Russo Latona, G.; Gote, C.; Zingg, C.; Casiraghi, G.; Verginer, L.; Schweitzer, F. Shock! Quantifying the impact of core developers' dropout on the productivity of OSS projects. In Proceedings of the Companion Proceedings of the ACM on Web Conference 2024, Singapore, 13–17 May 2024; pp. 706–709.
7. Coelho, J.; Valente, M.T. Why modern open source projects fail. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 186–196.
8. Miller, C.; Kästner, C.; Vasilescu, B. "We feel like we're winging it": A study on navigating open-source dependency abandonment. In Proceedings of the ESEC/FSE '23: ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, San Francisco, CA, USA, 3–9 December 2023; pp. 1281–1293.
9. Roumani, Y.; Nwankpa, J.K.; Roumani, Y.F. Adopters' trust in enterprise open source vendors: An empirical examination. *J. Syst. Softw.* **2017**, *125*, 256–270. [CrossRef]
10. Dahlander, L.; Magnusson, M. How do firms make use of open source communities? *Long Range Plan.* **2008**, *41*, 629–649. [CrossRef]
11. Alshomali, M.A.A.; Albayati, W. Mining GitHub for factors that affect open source software sustainability. *Int. J.* **2023**, *10*, 122–133. [CrossRef]
12. Jarczyk, O.; Jaroszewicz, S.; Wierzbicki, A. Surgical teams on GitHub: Modeling performance of GitHub project development processes. *Inf. Softw. Technol.* **2018**, *100*, 32–46. [CrossRef]
13. Ait, A.; Izquierdo, J.L.C.; Cabot, J. An empirical study on the survival rate of GitHub projects. In Proceedings of the 19th International Conference on Mining Software Repositories, Pittsburgh, PA, USA, 23–24 May 2022; pp. 365–375.
14. Han, J.; Deng, S.; Xia, X.; Wang, D.; Yin, J. Characterization and prediction of popular projects on GitHub. In Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 15–19 July 2019; Volume 1, pp. 21–26.
15. Zhou, H.; Ravi, H.; Muniz, C.M.; Azizi, V.; Ness, L.; de Melo, G.; Kapadia, M. Gitevolve: Predicting the evolution of GitHub repositories. *arXiv* **2020**, arXiv:2010.04366.
16. Eluri, V.K.; Sarkani, S.; Mazzuchi, T.A. Open source software survivability prediction using multi layer perceptron. In Proceedings of the 28th International Conference on Software Engineering and Data Engineering, San Diego, CA, USA, 30 September–2 October 2019; Volume 64, pp. 148–157.
17. Decan, A.; Constantinou, E.; Mens, T.; Rocha, H. GAP: Forecasting commit activity in Git projects. *J. Syst. Softw.* **2020**, *165*, 110573. [CrossRef]
18. Avelino, G.; Constantinou, E.; Valente, M.T.; Serebrenik, A. On the abandonment and survival of open source projects: An empirical investigation. In Proceedings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Recife, Brazil, 19–20 September 2019; pp. 1–12.
19. Dabbish, L.; Stuart, C.; Tsay, J.; Herbsleb, J. Social coding in GitHub: Transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, Seattle, WA, USA, 11–15 February 2012; pp. 1277–1286.
20. Borges, H.; Hora, A.; Valente, M.T. Predicting the popularity of GitHub repositories. In Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering, Ciudad Real, Spain, 7 September 2016; pp. 1–10.
21. Borges, H.; Hora, A.; Valente, M.T. Understanding the factors that impact the popularity of GitHub repositories. In Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, USA, 2–7 October 2016; pp. 334–344.
22. Ebert, C.; Cain, J.; Antoniol, G.; Counsell, S.; Laplante, P. Cyclomatic complexity. *IEEE Softw.* **2016**, *33*, 27–29. [CrossRef]
23. Collett, D. *Modelling Survival Data in Medical Research*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2023.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.