The Journal of Systems & Software 195 (2023) 111539

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Automating Feature Model maintainability evaluation using machine learning techniques[☆]

Públio Silva[a], Carla Bezerra[a,*], Ivan Machado[b]

[a] Department of Computer Science, Federal University of Ceará, Av. José de Freitas Queiroz, 5003 Cedro, Quixadá, Brazil
[b] Institute of Computing, Federal University of Bahia, Av. Adhemar de Barros s/n - Ondina, Salvador, Brazil

A R T I C L E I N F O

Article history:
Received 18 February 2022
Received in revised form 15 July 2022
Accepted 6 October 2022
Available online 17 October 2022

Keywords:
Quality Assessment
Software Product Lines
Feature Model
Machine Learning

A B S T R A C T

Context: Software Product Lines (SPL) are generally specified using a Feature Model (FM), an artifact designed in the early stages of the SPL development life cycle. This artifact can quickly become too complex, which makes it challenging to maintain an SPL. Therefore, it is essential to evaluate the artifact's maintainability continuously. The literature brings some approaches that evaluate FM maintainability through the aggregation of maintainability measures. Machine Learning (ML) models can be used to create these approaches.
Objective: This work proposes white-box ML models intending to classify the FM maintainability based on 15 measures.
Methods: To build the models, we performed the following steps: (i) we compared two approaches to evaluate the FM maintainability through a human-based oracle of FM maintainability classifications; (ii) we used the best approach to pre-classify the ML training dataset; (iii) we generated three ML models and compared them against classification accuracy, precision, recall, F1 and AUC-ROC; and, (iv) we used the best model to create a mechanism capable of providing improvement indicators to domain engineers.
Results: The best model used the decision tree algorithm that obtained accuracy, precision, and recall of 0.81, F1-Score of 0.79, and AUC-ROC of 0.91. Using this model, we could reduce the number of measures needed to evaluate the FM maintainability from 15 to 9 measures. Furthermore, we created a mechanism to suggest FM refactorings to improve the maintainability of this artifact. FM maintainability evaluation and the refactoring suggestion mechanism were automated in the DyMMer tool.
Conclusion: We conclude this work by presenting a way to combine FM maintainability assessment with FM refactorings. The results of this work provide to domain engineers inputs that will allow them to carry out a continuous improvement of an SPL.

1. Introduction

A software product line (SPL) is a family of software systems created and developed from a common set of features (Clements and Northrop, 2002). A feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option (Kang et al., 1990; Batory, 2005; Soares et al., 2018). Features are a central concept within the SPL paradigm, and one of the artifacts used to specify them is the feature model (FM) (Kang et al., 1990; Apel et al., 2013). FM is a graphical representation that defines SPL features and the relationships and constraints between them. From an FM, it is possible to retrieve all valid combinations between the features (Acher et al., 2013; Bürdek et al., 2016). As the design of an FM occurs in the domain engineering phase, in the early stages of the SPL development, errors in this artifact can spread across the SPL (Montagud et al., 2012; Bezerra et al., 2017).

Considering that an FM defines all the features of an SPL, this artifact can quickly become too complex over time, making it challenging to maintain and evolve the SPL (Passos et al., 2013; Bezerra et al., 2016b; Marques et al., 2019; Lima et al., 2020; Greenwell et al., 2020). Such increased complexity occurs because changes underwent by the SPL (e.g., changes in requirements that result in the addition or removal of features) generally imply changes in the FM (Bürdek et al., 2016). Therefore, it is essential to evaluate the quality of this artifact during the evolution of an SPL so that the model remains maintainable (Bezerra et al., 2017; Bagheri and Gasevic, 2011; Oliveira and Bezerra, 2019; Silva et al., 2020).

☆ Editor: Raffaela Mirandola.
* Corresponding author.
E-mail address: carlailane@gmail.com (C. Bezerra).

https://doi.org/10.1016/j.jss.2022.111539
0164-1212/© 2022 Elsevier Inc. All rights reserved.

One of the strategies commonly used to evaluate the FM maintainability is using quality measures (Bezerra et al., 2017; Bagheri and Gasevic, 2011; Oliveira and Bezerra, 2019; Silva et al., 2020; Berger and Guo, 2014a; El-Sharkawy et al., 2019b; Rocha et al., 2020). In Bezerra et al. (2017), the authors carried out a systematic mapping study to identify FM quality measures from the literature. As a result, the authors leveraged a catalog of 32 measures to evaluate the quality of an FM. However, it is still difficult to assess the general quality of an FM by using quality measures. Each measure focuses on a specific characteristic of the model and not on this as a whole. Also, the ranges of values of the measures are broad, and there is no clear indication of which values can be considered adequate or inadequate (Oliveira and Bezerra, 2019)—the greater the number of measures, the greater the time and effort required for evaluation. Manual evaluation becomes even more difficult. Several studies have used metrics, datasets, and machine learning (ML) techniques to predict software maintainability in single-system software development (Jha et al., 2019; Alsolai and Roper, 2020; Maggio, 2013).

A feasible strategy to handle the problems mentioned above would be to aggregate measures in a single value that indicates the general maintainability of an FM. In Oliveira and Bezerra (2019), the authors used such a strategy, applying fuzzy logic to a set of 15 quality measures to aggregate them to produce a single index capable of indicating the degree of maintainability of an FM. This strategy also allows the classification of FM maintainability in the Likert scale (in the work of Oliveira and Bezerra (2019) the values of the Likert scale were *very low, low, medium, high* and *very high*), based on the index value. In our previous work (Silva et al., 2020), we used another strategy to group the values of maintainability measures into a single value. Clustering algorithms, a type of ML algorithm, were applied to group FMs according to the values of a set of maintainability measures. Therefore, it was also possible to classify the FM maintainability by using a Likert scale (in our previous work the values of the Likert scale were *very bad, bad, moderate, good* and *very good*).

When using the ML approach, it is necessary to obtain a pre-classified FM data set as it is a supervised learning problem (Marsland, 2015). One way to classify the data is to use SPL experts. However, a large number of experts is needed to obtain a reasonably sized dataset, which makes this option unfeasible. Hence, we decided to use an automatic classification approach of FM maintainability to pre-classify the training dataset of the algorithms.

FM maintainability classifying improves an SPL by indicating likely problems in an FM. However, it is unclear what actions should be taken to improve FM maintainability. Some studies in the literature describe FM refactorings (Tanhaei et al., 2016; Alves et al., 2006; Gheyi et al., 2008). However, they do not address the relationship between the FM maintainability evaluation and refactorings. In this work, we were able to relate the FM maintainability evaluation with FM refactorings to make it explicit to the domain engineer the actions to improve FM maintainability.

This work is an extension of a previous work (Silva et al., 2021) which aimed to employ ML algorithms to classify FM maintainability. When compared to prior work, the novel contributions include: (i) a performance measurement in the FM maintainability evaluation algorithm to evaluate scalability, (ii) the interpretation of the decision tree model results provides suggestions for changes in the values of the maintainability measures to increase the maintainability of the FM, (iii) the addition of FM refactoring suggestions extracted from the literature based on the suggestions of changes in the values of the maintainability measures obtained through the interpretation of the ML model results, and (iv) the integration of the FM maintainability assessment and automated refactoring suggestions in the DyMMer (Dynamic feature Model tool based on Measures) tool (Bezerra et al., 2016a, 2021).

Four research questions guided our investigation:

- **RQ**$_1$: *What are the most representative measures in the MiniCOfFEE catalog to evaluate the FM maintainability?*
- **RQ**$_2$: *Among the ML models created in this work, which one is the best to assess FM maintainability?*
- **RQ**$_3$: *How possible is it to recommend FM refactoring rules based on an ML model created in this work?*
- **RQ**$_4$: *How scalable is the FM maintainability assessment using an ML model proposed in this work?*

RQ$_1$ aims to investigate which maintainability measures are needed to evaluate FM maintainability. This reduction can increase the performance and scalability of ML models. **RQ**$_2$ aims to investigate the best ML model among those produced in this work in terms of accuracy, precision, recall, F1, and AUC-ROC. The best ML model will be made available as a result of this work. **RQ**$_3$ aims to investigate whether it is possible to use the result produced by an ML model to recommend refactorings in FM to improve maintainability. **RQ**$_4$ aims to investigate whether the FM maintainability evaluation using the ML model selected in the previous research question is scalable (e.g., whether the assessment remains viable in terms of performance when evaluating increasingly larger FMs and more complex).

Among the main contributions of this study we could highlight the following:

- Comparing two FM maintainability classification approaches described in the literature;
- Creating an ML-based approach to classifying the FM maintainability using white-box algorithms;
- Finding out the nine most significant measures to assess FM maintainability;
- Using the ML model's result to suggest FM refactorings to improve maintainability;
- Adding FM maintainability evaluation and FM refactoring suggestions to the DyMMer tool.

The remainder of this article is organized as follows. Section 2 presents the theoretical basis to support the understanding of this study and related work. Section 3 describes the study design. Section 4 presents the achieved results. Section 5 discusses the main threats to validity. Finally, Section 6 concludes the article.

2. Background

2.1. FM maintainability assessment

In Bezerra et al. (2014), the authors carried out a systematic mapping study and identified 32 quality measures for FM in a catalog called COfFEE. In a later study (Bezerra et al., 2017), the authors analyzed the measures from the COfFEE catalog and identified correlations between some of these measures. In addition, they found that it was possible to characterize the quality of an FM by using only 15 out of the 32 measures. Therefore, they created a new catalog describing these 15 measures and named it MiniCOfFEE (Bezerra et al., 2017) (see Table 1). In this current study, we aggregated the measures from the MiniCOfFEE (Bezerra et al., 2017) catalog by using ML models to allow the assessment of FM maintainability. The DyMMer tool will support collecting measures automatically (Bezerra et al., 2016a, 2021).

Table 1
MiniCOfFEE: subset of 15 maintainability measures for FMs selected from the COfFEE catalog (Bezerra et al., 2014, 2018).

ID	Name	Description
NF	Number of Features	Number of features in the model
NM	Number of Mandatory Features	Number of mandatory features in the model
NLeaf	Number of Leaf Features	Number of features without children
NTop	Number of top Features	Number of features direct descendant of the root
CogC	Cognitive Complexity	Number of variants points
FEX	Feature Extensibility	NLeaf + SCDF + MCDF
FoC	Flexibility of Configuration	(Number of optional features)/NF
DTMax	Maximum Depth of Tree	Number of features of the longest path from the root of the feature model
NVC	Number of Valid Configurations	Number of possible and valid configurations of the feature model
RoV	Ratio of Variability	NVC/NF^2
RDen	Coefficient of connectivity-density	Average number of (non-parent) features referenced in the constraints of a feature
NGXOr	Number of XOr groups	Number of variation points with relationship XOr
NGOr	Number of Or groups	Number of variation points with relationship Or
SCDF	Number of Features dependent on unique cycles	The sum of all features that participate in child feature constraints that have variant points with [1..1]
MCDF	Multiple feature-dependent cycles	The sum of all features that participate in child feature constraints that have cardinality variant points [1..*]

2.2. DyMMer 2.0 tool

DyMMer 2.0 tool is an extension (and Web version) of the DyMMer desktop tool, which allows runtime modeling, storage, and evaluation of the SPLs and DSPLs feature models from the thresholds of the measures.[1] The tool was developed in JavaScript and the code is available on GitHub[2] (Bezerra et al., 2021). It is licensed under the MIT Open Source.[3] Fig. 1 represents the start panel that summarizes the main information about the tool, showing additional information about feature modeling. The main features of the DyMMer 2.0 tool are (i) modeling of FMs from SPLs and DSPLs, (ii) calculation of quality measures to evaluate the FM maintainability, (iii) development of an adaptation mechanism for FM of DSPLs, (iv) repository of FMs, (v) inclusion of thresholds for measures, and (vi) user authentication. The tool's evaluation feature allows a user to select up to 40 measures for FM evaluation at modeling time. Table 1 shows all measures the DyMMer 2.0 tool calculates.

2.3. Machine learning

ML is a subarea of artificial intelligence (AI) that brings together a set of methods that allow computers to learn from data to improve predictions (Alpaydin, 2020). ML is a branch of computational algorithms that are still evolving (El Naqa and Murphy, 2015). Several areas, e.g., finance, health, and advertising, to name a few, have applied ML methods and techniques to handle particular problems (Bailey et al., 2018). Many studies available in the literature employ ML with SPL for several purposes, such as constraints inferring (Temple et al., 2016) and quality measures aggregation (Silva et al., 2020). There are four "learning" types: supervised learning, unsupervised learning, reinforcement learning, and evolutionary learning. The former is the most commonly employed one. For the resolution of ML problems (especially in supervised and unsupervised learning), a generally adopted process encompasses the following stages: data collection and preparation, selection of independent variables, choice of algorithms, selection of parameters and models, and evaluation (Marsland, 2015).

Supervised learning problems comprise two types of problems: (i) classification problems, in which the dependent variable is categorical; and (ii) regression problems, in which the dependent variable is numeric (Marsland, 2015). In this work, we handle the classification of FMs by considering their maintainability values. We take the set of 15 FM maintainability measures and return FM maintainability as a Likert scale item. This scenario encompasses a classification problem (supervised learning) as the intended result is categorical. There are several algorithms for the classification problem, among which we can highlight:

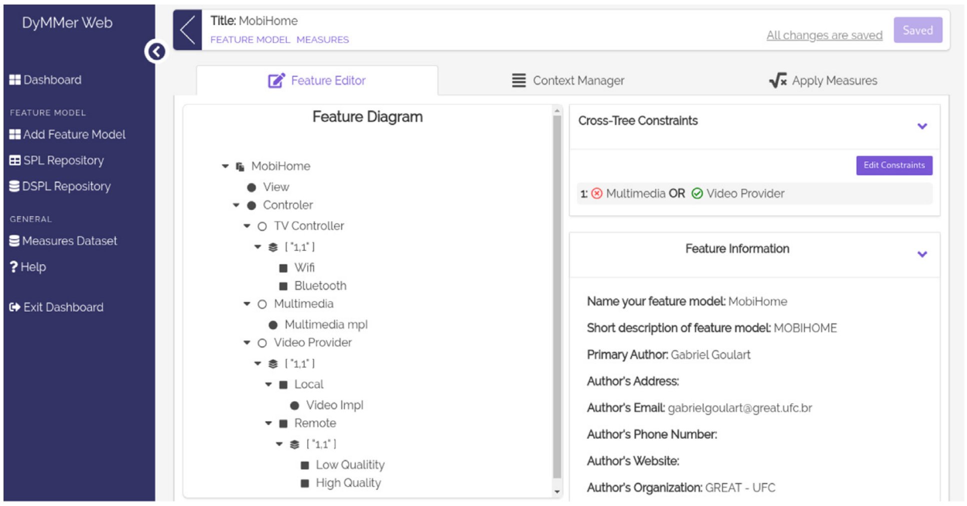
- **Naive Bayes**. It is a family of algorithms that calculate the probability of a given result according to a set of conditions, using the theorem of Bayes (Bonaccorso, 2017). Naive Bayes is a powerful and easy to train classifier (Bonaccorso, 2017). It receives the name naive because it assumes that the independent variables have no dependencies (i.e., they are not correlated), which may not be the case in certain classification problems (Sen et al., 2020).
- **Logistic Regression**. Despite having regression in the name, this is a classification algorithm based on the probability of a given sample belonging to a certain class (Bonaccorso, 2017). The logistic regression algorithm has similarities with the linear regression algorithm (used in regression problems). The difference is that in the logistic regression, the output value of the function must be between (0, 1). Therefore, the sigmoid function is introduced, whose output value must be between (0, 1).
- **Decision Tree**. It is a type of logic-based algorithm to use for regression and classification problems (Sen et al., 2020). In classification problems, the algorithm forms a binary tree structure that allows a sequential decision process to discover the class of a given sample (Bonaccorso, 2017). It is possible to evaluate a given independent variable from the tree root and decide whether the next node will be either on the right or on the left. This process repeats until reaching a leaf node, which indicates the resulting class (Bonaccorso, 2017).

There are several measures to measure the quality of a classification model. In general, measures consider true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) values. Some of the main classification measures are:

- **Accuracy**. It is measured by the ratio of the number of correct classifications to the total number of samples (Chicco and Jurman, 2020). The formula below expresses accuracy:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

[1] https://dymmerufc.github.io/.
[2] https://github.com/dymmerufc/Dymmer-Web.
[3] https://opensource.org/licenses/mit-license.php.



Title: MobiHome

1: Multimedia OR Video Provider

Name your feature model: MobiHome

Short description of feature model: MOBIHOME

Primary Author: Gabriel Goulart

Author's Address:

Author's Email: gabrielgoulart@great.ufc.br

Author's Phone Number:

Author's Website:

Author's Organization: GREAT - UFC

Fig. 1. FM viewer in DyMMer 2.0.

- **Precision**. The precision measure has a great emphasis on catching false-positive errors. It can be measured by the ratio of the number of correct positive ratings to the total number of positive ratings (Chicco and Jurman, 2020). Precision is expressed by the formula below.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

- **Recall**. Unlike precision, the recall measure has a great emphasis on catching false negative errors. It can be measured by the ratio between the number of correct positive ratings and the number of correct positive ratings plus the number of false-negative ratings (Chicco and Jurman, 2020). Recall is expressed by the formula below.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

- **F1 Score**. This measure is obtained from the calculation of the harmonic mean between the precision and recall (Chicco and Jurman, 2020). F1-Score is expressed by the formula below.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

- **AUC-ROC**. This is a performance measure for the classification problem that tells how much a model is able to distinguish between the classes (Narkhede, 2018). ROC (Receiver Operating Characteristic) is a probability curve plotted with TPR (True Positive Ratio) versus FPR (False Positive Ratio) (Narkhede, 2018). AUC is the area under the ROC curve. TPR and FPR are expressed by the formulas below.

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$FPR = \frac{FP}{TN + FP} \tag{6}$$

2.4. Related work

Bezerra et al. (2014) carried out a systematic mapping to identify measures to assess the quality of FMs. The authors then proposed COfFEE, a catalog of 32 FM quality measures, and validated with FMs extracted from the SPLOT repository (Mendonca et al., 2009). The measures identified in this work were implemented for automatic collection in the DyMMer tool (Bezerra et al., 2016a). In a later work (Bezerra et al., 2017), the authors carried out an analysis of the correlation between the measures in the COfFEE catalog. As a result, the authors found that a subset of the measures in the original catalog was enough to assess the quality of FMs. The authors then proposed a new catalog called MiniCOfFEE (Bezerra et al., 2017), consisting of 15 out of the 32 measures in the COfFEE catalog. In this work, the measurements from the MiniCOfFEE (Bezerra et al., 2017) catalog will serve to train ML models to assess the maintainability of FMs. Also, the measures in the MiniCOfFEE (Bezerra et al., 2017) catalog are analyzed to see if it is still possible to reduce the number of measures necessary to assess the maintainability of an FM. The analysis considers the correlations between the 15 FM maintainability measures and the importance of each measure for a set of ML models.

El-Sharkawy et al. (2019b) carried out a systematic literature review to identify measures of SPL variability. The authors analyzed 42 papers in which 52 measures for variability models were identified, 80 for code artifacts and 10 for both artifacts. In subsequent works, the authors manage to automate the measures in a tool called MetricHaven (El-Sharkawy et al., 2019a) and evaluate the automated structure from a set of metrics in the Linux Kernel (El-Sharkawy et al., 2020). Despite the large number of measures supported by this tool, most of them require code artifacts, which we did not use in this work. Also, this tool does not support the calculus of all measures in the MiniCOfFEE catalog. Therefore, in this work, we prefer to use the DyMMer tool (Bezerra et al., 2016a).

The literature presents some approaches to evaluate the FM maintainability using maintenance measures. Two out of them use the same catalog of measures that we employ in this work: Silva's approach described in our previous work (Silva et al., 2020), and Oliveira's approach (Oliveira and Bezerra, 2019). This work compares the two approaches and selects the best one to pre-classify the ML training dataset.

Silva's approach (Silva et al., 2020) uses clustering algorithms (a type of ML algorithm) to assess the maintainability of the FM. We employed the Vale method (Vale et al., 2019) to analyze each group of FMs and assign each one a label indicating the maintainability of the group members. These algorithms enable the generation of clusters (or groups) from uncategorized or unclassified data considering the characteristics and similarities of the samples of the dataset (Alloghani et al., 2020). After generating the clusters, labels were assigned to each, indicating their maintainability. In this way, it is possible to know the maintainability of an FM only by the cluster to which the FM belongs. In the FM maintenance classification problem, Silva's approach obtained an accuracy of 0.61 and a precision of 0.70. We employed supervised learning models to assess the FM maintainability instead of unsupervised ones. Besides, we seek to reduce the number of measures necessary to evaluate the maintainability of the FM.

Oliveira's approach (Oliveira and Bezerra, 2019) uses fuzzy logic to aggregate the measures from the MiniCOfFEE (Bezerra et al., 2017) catalog to produce a Maintainability Index for Feature Models (MIFM). Fuzzy logic is a type of logic that admits that a premise is partly true and partly false (Klir and Yuan, 1995). The maintainability index ranges from 0 to 100. It allows classifying the FM maintenance in the Likert scale according to the MIFM value. Whereas Oliveira and Bezerra (2019) used fuzzy logic, in this current work, we used ML models. We also analyze the importance of maintainability measures to reduce the number of measures used in the FM evaluation.

In Pereira et al. (2021), a systematic literature review is developed to identify works that used learning techniques in the context of highly configurable software. With the analysis of the selected works, the authors were able to leverage learning techniques employed in six scenarios, namely: pure prediction, interpretability of configurable systems, optimization, dynamic configuration, mining constraint, and evolution. The present work fits into at least four of these scenarios, as follows: pure prediction (we were able to predict the maintainability classification of an FM); interpretability of configurable systems (we used only white box algorithms, which allowed us to interpret the model results of ML and based on that to suggest precise changes in the values of the maintainability measures); dynamic configuration (since what we produced in this work can be applied both in the context of conventional SPL and in the context of dynamic SPL); and evolution (after evaluating the maintainability of the FM, we were also able to create a mechanism capable of suggesting FM refactorings, which the domain engineer can use on a process of continuous improvement of the SPL). Furthermore, in the works analyzed in Pereira et al. (2021), we could not find discussions surrounding the combination of maintainability assessment with refactoring suggestions. This contribution provides fundamental inputs to perform a continuous improvement process of the SPL.

3. Study settings

This work investigates the use of ML techniques to classify the maintainability of FM based on measures. Among the issues to explore is whether this FM assessment strategy is scalable. For this, we use the MiniCOfFEE (Bezerra et al., 2017) measures catalog, composed of 15 maintainability measures. Although this catalog reduces to a more extensive catalog, we want to verify if it is still possible to identify the most representative measures to evaluate FM maintainability. In addition, we also want to use the results obtained with the ML techniques to suggest FM refactorings so that it becomes clear for domain engineers the necessary measures to improve maintainability.

3.1. Research questions (RQ)

RQ$_1$: *What are the most representative measures in the MiniCOfFEE catalog to evaluate the FM maintainability?* This RQ seeks to investigate which measures are most representative to evaluate the FM maintainability of the set 15 measures described in the MiniCOfFEE (Bezerra et al., 2017) catalog. To answer **RQ**$_1$ we use two techniques for selecting independent variables (which in this work are the measures of maintainability).

RQ$_2$: *Among the ML models created in this work, which one is the best to assess FM maintainability?* This RQ seeks to investigate the best ML model produced in terms of accuracy, precision, recall, F1-Score, and AUC-ROC. To answer **RQ**$_2$ we calculate the measures cited for each of the models produced.

RQ$_3$: *How possible is it to recommend FM refactoring rules based on an ML model created in this work?* This RQ investigates whether the results of an ML model application could aid in recommending FM refactoring rules. To answer **RQ**$_3$ we analyzed the artifacts produced by the ML model selected in the previous RQ and studied ways to derive FM refactoring rules based on these artifacts and the result of the ML model.

RQ$_4$: *How scalable is the FM maintainability assessment using an ML model proposed in this work?* This RQ seeks to investigate the scalability of the FM maintainability evaluation using an ML model, that is, how the time to assess an FM varies according to its size. To answer **RQ**$_4$, we used the ML model selected in the previous RQ to evaluate the maintainability of FMs of different sizes, and we calculated the time taken to perform the evaluation. As the ML model already receives the calculated FM maintainability measure values, we did not consider the time to extract the measures in the performance evaluation.

The performance evaluation does not consider the time to extract the maintainability measurements of the FM as the ML model used to evaluate the FM maintainability already receives the calculated measure values.

3.2. Study steps and procedures

In this section, we describe the steps we took in the empirical evaluation.

Step 1: Collect data and perform an empirical study with experts. To build the dataset for training ML algorithms, we extracted 342 FMs[4] from the repository SPLOT (Mendonca et al., 2009). The base criteria we used to form the dataset was choosing FMs of varying sizes and complexity. We use the DyMMer tool (Bezerra et al., 2016a) to calculate the measures from the MiniCOfFEE (Bezerra et al., 2017) catalog for each of the samples. The dataset[4] is composed of FMs of different sizes, ranging from 10 to 290 features, with an average of approximately 30 features per FM. The Fig. 2 shows the distribution of the 15 maintainability FM measures of MiniCOfFEE on the full dataset. After calculating the 15 measures for all the FMs from the dataset, we applied both Silva et al. (2020) and Oliveira and Bezerra (2019) approaches to classify the maintainability of the 342 FM in the dataset. We passed the values of the 15 measures as input to them[4]. Both Silva et al. (2020) and Oliveira and Bezerra (2019) approaches use five labels (a standard Likert scale) to classify the FM maintainability. In the first one, the five labels are Very Low, Low, Medium, High, and Very High. In the second one, they are Very Bad, Bad, Moderate, Good, and Very Good. To standardize the results, we converted the labels of the Oliveira approach to the ones used in the Silva one. These five labels represent the independent variable of the FM maintainability classification problem we addressed in this work.

We compared the classifications of both approaches with an oracle formed by classifications performed manually by experts in SPL. To create the oracle that would be the basis for the comparison between the Silva et al. (2020) and Oliveira and Bezerra (2019) approaches, we chose a subset of the 342 FMs

[4] https://github.com/publiosilva/JSSSPLC2021.

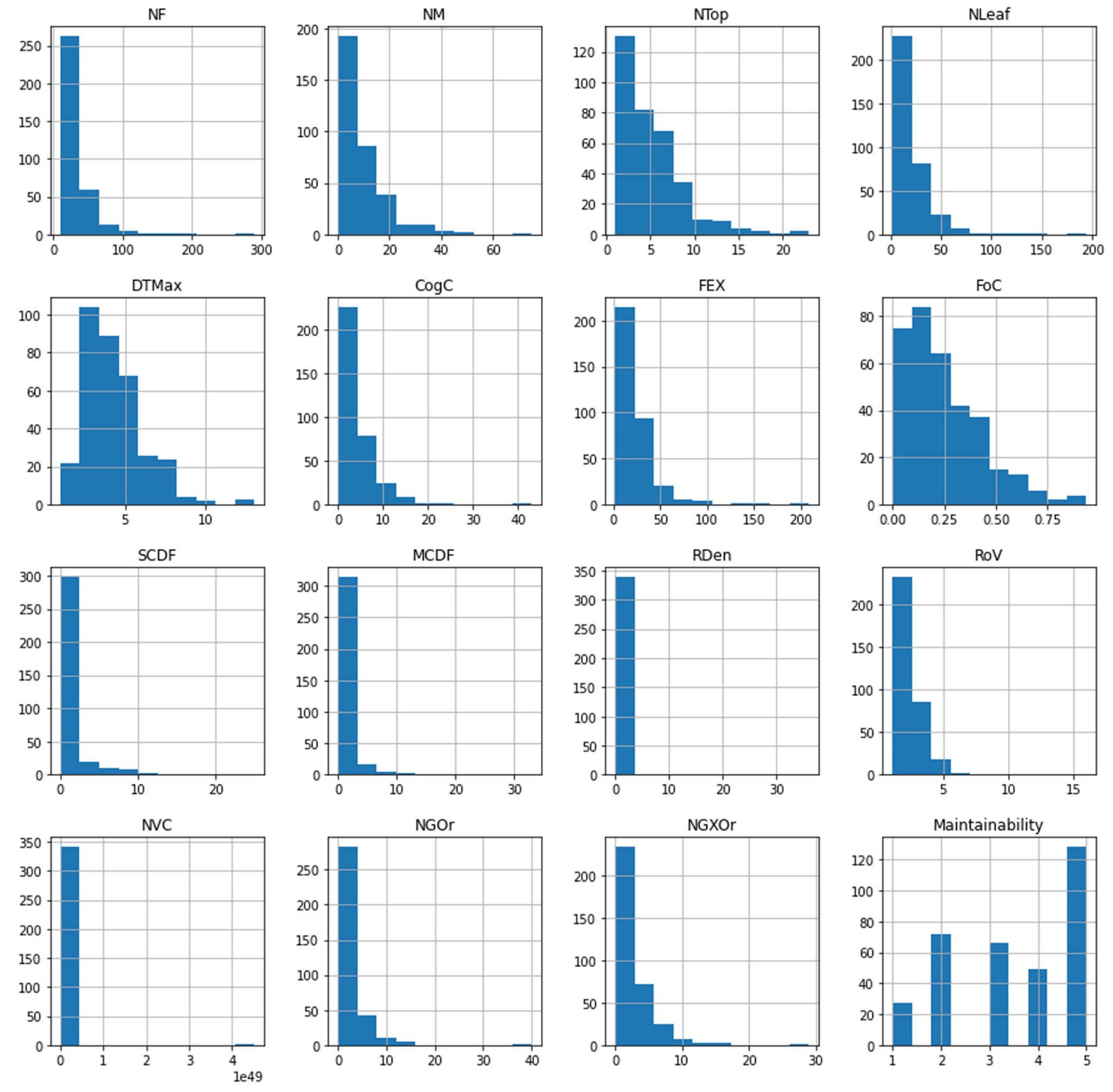


Fig. 2. Histograms showing the distribution of the 15 maintainability FM measures of MiniCOfFEE on the full dataset.

that comprised the dataset to be classified in terms of maintainability manually by a group of experts in SPL. For the selected subset to be representative concerning the complete dataset, we grouped the 342 FMs by similarity, considering the values of the 15 maintainability measures. To do this grouping, we use the K-means clustering algorithm, a well-known and widely used grouping/clustering algorithm (Jain, 2010). This algorithm is available in the scikit-learn.[5] The main parameter of this algorithm is the number of clusters. To find a good value for this parameter, we use a well-known heuristic, the elbow method (Syakur et al., 2018). After grouping, 50 FMs[4] of the different groups generated, thus ensuring that the FMs the experts would evaluate have other characteristics and are representative of the complete dataset. Fig. 3 shows the distribution of the 15 maintainability FM measures of MiniCOfFEE on the reduced dataset.[6] The distribution of all measures in the reduced dataset is very similar to that of the full dataset (Fig. 2).

We then contacted several SPL experts via e-mail and WhatsApp. In this first contact, we sent a questionnaire to gather profile data (name, e-mail, and education) and a declaration of consent to participate in the second stage of the survey. In all, 15 experts agreed to participate in the second stage of the survey. These experts were sent a second questionnaire where they would have to rate the 50 FMs. It encompasses background questions to gather the participants' profiles and questions specific to the classification task. Each expert classified the maintainability of 10 FMs considering five categories and each FM was classified by 3 different experts. For each of the ten FMs, the participants could access an external document to view the structure of the FM tree and the values of the 15 maintainability measures. Two videos were made available with instructions for answering the questionnaire in the form delivered to the participants. With that, five forms were created so that the 50 FMs could be evaluated. Appendix B shows a sample form. Table 2 shows the profile of the 15 experts.

Since each FM was rated by three different experts, one FM could receive three different maintainability classifications. Thus, to increase the degree of confidence in the ratings obtained with the responses, we decided that we would only consider the ratings that obeyed one of the following constraints: (i) if there was

5 https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

6 https://github.com/publiosilva/JSSSPLC2021/blob/main/datasets/reduced-dataset.csv.

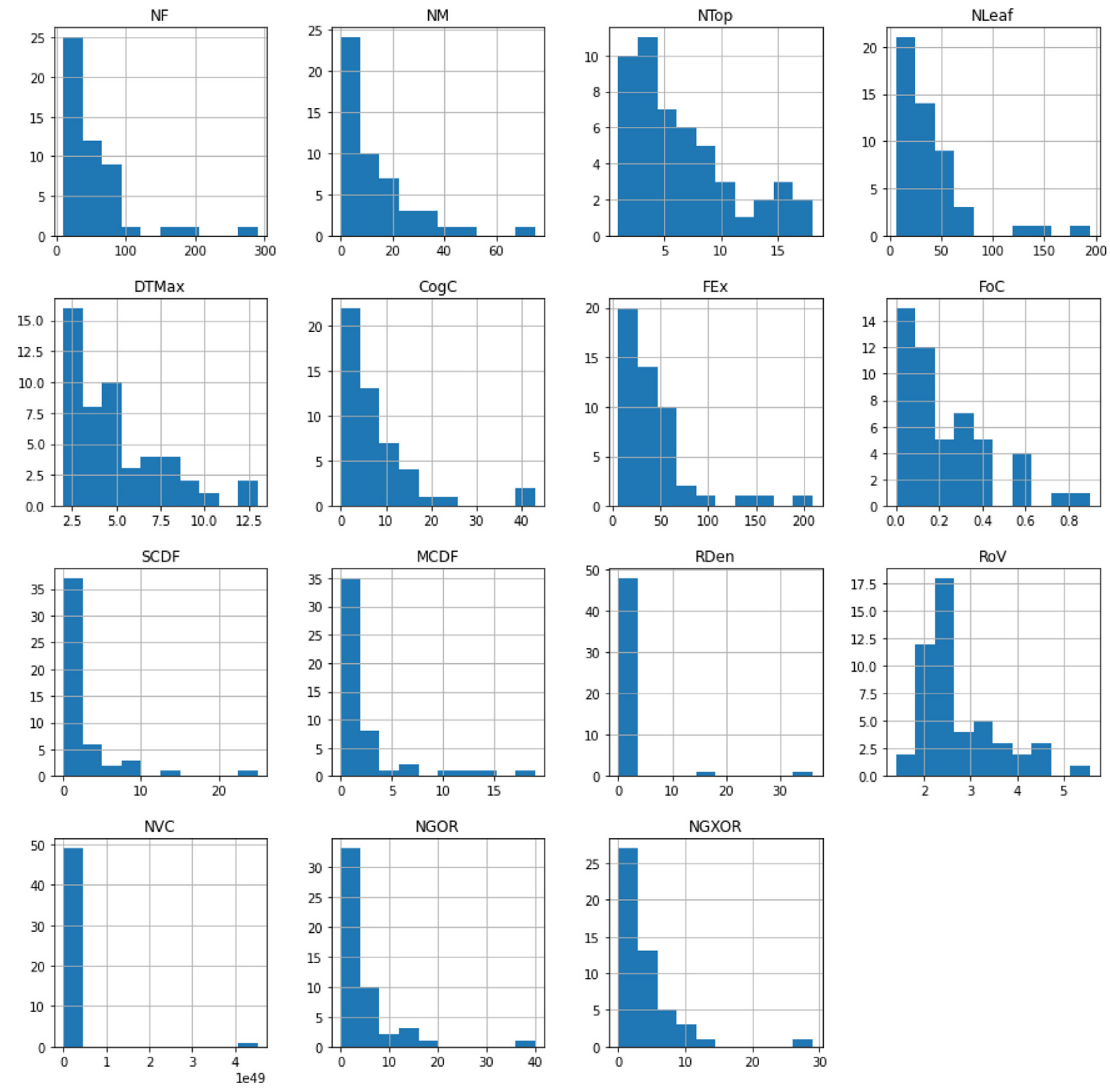


Fig. 3. Histograms showing the distribution of the 15 maintainability FM measures of MiniCOfFEE on the reduced dataset.

consensus among the three participants as to the classification, and (ii) if any two participants agreed on the classification and the third one defined a neighboring class to that defined by the other two experts (for example, two participants classify an FM as good and the third as very good). Therefore, we would consider the classification chosen by most of the participants. We selected 28 FM of the first 50 to calculate the comparison measures by this exclusion criteria.

We proposed three measures to manually compare the results obtained with the Silva et al. (2020) and Oliveira and Bezerra (2019) approaches through the experts' classification, forming the comparison oracle. The three measures defined are described below:

- **Accuracy of the classifications of an approach to the classifications of SPL experts (ACAC).** To calculate this measure, one must obtain the number of times the approach classified an FM with the same class that the experts used to classify the same FM. After that, this value is divided by the total number of FMs in the comparison oracle. This gives a value that can vary from 0 to 1, with 0 being the worst value and 1 being the best value. This measure is represented by the formula below:

$$\frac{Number\ of\ ratings\ equal\ to\ that\ of\ experts}{Total\ number\ of\ FMs} \tag{7}$$

- **Rate of optimism and pessimism of an approach (ROPA).** To calculate this measure, one must: (i) obtain the number of times that an approach classified an FM with a higher class than the experts used to classify the same FM; and, (ii) obtain the number of times that an approach classified an FM with a lower class than the experts used to classify the same FM. After that, the first is subtracted by the second value, and the result is divided by the total number of FMs in the comparison oracle. The obtained value can be positive, negative, or zero. A zero value indicates that the approach classifies FMs with pessimism and optimism at the same frequency. A negative value indicates that the approach makes more pessimistic than optimistic classifications. A positive value indicates that the approach does more optimistic than pessimistic ratings. This measure is represented by the formula below:

$$\frac{Number\ of\ optimistic\ ratings - Number\ of\ pessimistic\ ratings}{Total\ number\ of\ FMs} \tag{8}$$

Table 2
Profiles of experts who participated in the evaluation of the FM.

Expert	Experience with SPL (years)	Context that worked with SPL	Education degree	Level of knowledge about FMs	Level of knowledge about FM maintainability	Evaluated the FM maintainability?
1	1 to 5	Academy	Master	Has general knowledge but almost never uses	Hears about but never used	No
2	5 to 10	Academy and Industry	Doctorate	Has average knowledge and sometimes uses	Has general knowledge but almost never uses	No
3	1 to 5	Academia	Bachelor	Has average knowledge and sometimes uses	Has average knowledge and sometimes uses	Yes
4	1 to 5	Academia	Master	Has average knowledge and sometimes uses	Has general knowledge but almost never uses	Yes
5	0 to 1	Academy	Master	Has general knowledge but almost never uses	Hears about but never used	No
6	5 to 10	Academia	Doctorate	Is an expert on the subject and uses almost every day	Has general knowledge but almost never uses	No
7	1 to 5	Academia	Master	Has average knowledge and sometimes uses	Has average knowledge and sometimes uses	No
8	1 to 5	Academia	Bachelor	Has average knowledge and sometimes uses	Hears about but never used	No
9	5 to 10	Academia	Doctorate	Has average knowledge and sometimes uses	Hears about but never used	No
10	1 to 5	Academia	Master	Has average knowledge and sometimes uses	Has general knowledge but almost never uses	No
11	1 to 5	Academia	Master	Has average knowledge and sometimes uses	Has average knowledge and sometimes uses	Yes
12	5 to 10	Academia	Doctorate	Has average knowledge and sometimes uses	Has average knowledge and sometimes uses	Yes
13	More than 10	Academia	Doctorate	Is an expert on the subject and uses almost every day	Has average knowledge and sometimes uses	Yes
14	5 to 10	Academia	Master	Has average knowledge and sometimes uses	Hears about but never used	No
15	1 to 5	Academia	Master	Has average knowledge and sometimes uses	Has general knowledge but almost never uses	Yes

- **Average distance from the ratings of an approach to the ratings of SPL experts (ADRA).** The measure considers the distances between the ratings of an approach and the ratings the SPL experts attributed (e.g., if an approach classified an FM as moderate and the experts classified the same FM as very good, the distance is 5). In the end, this sum is divided by the total number of FMs in the comparison oracle. The lower the value, the greater the similarity between the classifications. The higher the value, the smaller the similarity between the classifications. This measure is represented by the formula below:

$$\frac{\textit{Distance for the experts classification}}{\textit{Total number of FMs}} \tag{9}$$

Table 3 shows the results of the defined comparison measures. The results showed that Oliveira's approach ranks with the same class defined by SPL experts more often than Silva's approach. Both approaches classify with more pessimism than optimism regarding the experts' classification, favoring both approaches. The results also showed that the Oliveira approach's classifications are, on average, closer to the classifications defined by SPL experts than the classifications obtained with the Silva approach. Therefore, Oliveira's approach stands to Silva's approach compared with the results obtained with a group of experts. Therefore, the training dataset of ML algorithms considered the ratings of the Oliveira approach.

Step 2: Select independent variables. As we earlier stated in this article, we verified whether it was possible to reduce the dimensionality of the addressed ML problem by using a smaller subset of the MiniCOfFEE (Bezerra et al., 2017) catalog to classify the FM maintainability with good precision. This verification occurred during the selection phase of the independent variables, where we selected the independent variables of most importance to address the problem. We employed two approaches to select independent variables: in the first one, we analyzed the correlation between the 15 maintainability measures from the MiniCOfFEE (Bezerra et al., 2017) catalog and the FM maintainability classification in the dataset we described in the previous section. In the second approach, we used an importance attribute of the independent variables that some ML algorithms provide, such as the logistic regression and the decision tree, to select only the independent variables with greater significance for each ML model. We did not use the Naive Bayes algorithm in the second approach as it does not directly provide an importance attribute. However, it is possible to extract this data using model agnostic methods such as the Feature Permutation Importance (Greenwell et al., 2020).

Table 3
Comparison of measures between Silva et al. (2020) and Oliveira and Bezerra (2019) approaches.

Measures	Silva's approach	Oliveira's approach
ACAC	0.28	0.39
ROPA	−0.28	−0.10
ADRA	1.25	0.89

Step 3: Train ML models. Naive Bayes, Logistic Regression, and Decision Tree algorithms, described in Section 2, were used to create three ML models to classify FM maintainability.[4] We chose to use only white-box algorithms, so we discard options like K-Nearest-Neighbor (KNN), which is black-box (Loyola-González, 2019) and Support Vector Machine (SVM), whose conventional approaches are boxed black (Ma et al., 2017). We also rule out the use of the Random Forest algorithm. Although using tree

structures such as those used by the decision tree algorithm, it is considered a black-box in some studies due to the high number of operations involved in the prediction mechanism (Bénard et al., 2021) and the size of random forests (Zhang and Wang, 2009).

ML models were trained using the dataset described in the previous sections. This procedure was done in parallel to evaluating ML models, described in the next section. In this step, it is necessary to adjust the specific hyperparameters of each algorithm so that the best values are obtained in each of the defined evaluation measures. We made a few adjustments to the hyperparameters of the three ML models in an iterative way to improve the accuracy, precision, recall, F1, and AUC-ROC metrics. The most notable adjustment example was limiting the maximum tree depth of the decision tree model to 5 levels. This limitation was based on the analysis of the accuracy of the ML model with different values for the maximum depth of the tree.

Step 4: Evaluate ML models. We evaluated the three ML models for classification of FM maintainability produced using the measures accuracy, precision, recall, F1-Score, and AUC-ROC, described in Section 2. To check how generalizable the ML models were, we performed 10-step cross-validation. At each step, we use 90% of the data for training and 10% for testing (Refaeilzadeh et al., 2016). Cross-validation is a model validation technique for evaluating how the results of a statistical analysis will generalize to an independent dataset (Browne, 2000). It attempts to estimate how accurately a learning method would perform in practice. The values of the five measures were calculated in the ten steps. In the end, we calculated the average of each measure in the ten steps.

Step 5: Create a mechanism to suggest FM refactorings. We used the result of the ML model that obtained the best results in the previous step to produce suggestions for changes in the values of the quality measures so that the evaluation result would improve. After that, we identified in the literature FM refactorings capable of causing the desired changes in the values of the quality measures (Temple et al., 2016; Tanhaei et al., 2016; Alves et al., 2006). Therefore, we built a mechanism to evaluate the FM maintainability and suggest FM refactorings to improve maintainability. We also implemented this mechanism in the DyMMer tool.

Step 6: Assess scalability of FM maintainability evaluation using ML. We used the FM generator provided by SPLOT to generate 5 FM of different sizes and complexities. After that, we used the ML model that obtained the best result in step 4 to evaluate the maintainability of the 5 FM. We measured the time taken to evaluate each of the models. After that, we analyze how the evaluation time varies as the FM size increases to determine if this FM evaluation strategy is scalable.

4. Results

4.1. What are the most representative measures in the MiniCOfFEE catalog to evaluate the FM maintainability? (RQ$_1$)

To answer **RQ$_1$** we used two independent variable selection approaches.[7] In the first one, we analyzed the correlation between the 15 measures in the MiniCOfFEE (Bezerra et al., 2017) catalog and the FM maintainability classification in the first approach. Among the existing correlation coefficients, we could highlight Pearson and Spearman. The former is suitable for normally distributed data. If data distribution is non-normal, the latter is preferable (Schober et al., 2018). We analyzed the distribution of the 15 maintenance measures of the MiniCOfFEE (Bezerra et al., 2017) catalog. We observed that the measures and maintenance of FMs do not follow a normal distribution, as Fig. 2 shows. Therefore, we used the Spearman correlation coefficient. Spearman's correlation coefficient ranges from −1 to +1, where 0 indicates no correlation between the variables involved, and values closer to −1 or +1 indicate a greater correlation between the variables involved (Schober et al., 2018). Table 4 shows the value of the correlation coefficient between the 15 maintainability measures and the FM maintainability classification (the full correlation matrix is available on Github[7]). We needed to transform data gathered from the Likert scale to numerical values (from 1 to 5).

Table 4
Spearman's correlation coefficient between the 15 maintainability measures and the FM maintainability rating.

NF	NM	NTop	NLeaf	DTMax	CogC	FEX	FoC	SCDF	MCDF
−0.8	−0.43	−0.25	−0.77	−0.62	−0.62	−0.74	0.17	−0.12	−0.14
RDen	RoV	NVC	NGOr	NGXOr					
−0.12	−0.047	−0.64	−0.39	−0.38					

Salkind and Rainwater (2006) provided a scale that was used in Berger and Guo (2014b) to intuitively understand Spearman's correlation coefficient values. The scale considers that values of:

- 0.8 to 1.0 or −0.8 to −1.0 indicate a very strong correlation.
- 0.6 to 0.8 or −0.6 to −0.8 indicate a strong correlation.
- 0.4 to 0.6 or −0.4 to −0.6 indicate a moderate correlation.
- 0.2 to 0.4 or −0.2 to −0.4 indicate a weak correlation.
- 0.0 to 2.0 or 0.0 to −0.2 indicate a very weak correlation or no correlation.

We applied this same scale to select the most correlated maintainability measures to classify FM maintainability. We only set the measures for which the absolute value of the correlation coefficient was greater than or equal to 0.6, thus remaining only with the measures with a strong or very strong correlation with the FM maintainability classification. The NF, NLeaf, DTMax, CogC, FEX, and NVC measurements were selected using this procedure.

In the second approach to selecting independent variables, we used an attribute of the importance of independent variables provided by some ML algorithms, such as the logistic regression and decision tree algorithms. This attribute is the coefficient associated with each independent variable in the logistic regression function in the logistic regression algorithm. An attribute is explicitly provided in the decision tree algorithm, indicating the importance of each independent variable for the FM. For the model that used the logistic regression algorithm, we used the utility *SelectFromModel*.[8] It is available at the scikit-learn library, and automatically selects a set of independent variables based on the importance attribute the ML algorithm provides. We obtained a set of maintenance measures formed by NF, NM, NTop, NLeaf DTMax, CogC, and FEX.

In the model that used the decision tree algorithm, we manually analyzed the importance attribute of the independent variables. The measures NGXOr, NGOr, NVC, RoV, MCDF, and FEX are not important for the ML model and, therefore, were excluded, leaving then the measures SCDF, CogC, RDen, NM, FoC, NTop, NLeaf, NF and DTMax. We did not use the second approach in the model that used the Naive Bayes algorithm. It does not provide an attribute that indicates the importance of independent variables for the model.

The two approaches mentioned above enabled obtaining three different sets of maintainability measures. Table 5 shows these

[7] https://github.com/publiosilva/JSSSPLC2021/blob/master/notebooks/Feature-Selection.ipynb.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html.

Table 5
Maintainability measures obtained when selecting independent variables.

Set #	How it was obtained	Maintainability measures
1	MiniCOfFEE (Bezerra et al., 2017)	DTMax, NF, NLeaf, CogC, NTop, NTop, RDen, FoC, NM, FEX, SCDF, MCDF, RoV, NVC, NGOr, NGXOr
2	Correlation analysis	NF, NLeaf, DTMax, CogC, FEX, NVC
3	Importance of measures for the logistic regression model	NF, NM, NTop, NLeaf, DTMax, CogC, FEX
4	Importance of measures for the decision tree model	SCDF, CogC, RDen, NM, FoC, NTop, NLeaf, NF, DTMax

measures together with another set containing all the maintainability measures from the MiniCOfFEE (Bezerra et al., 2017) catalog. We trained the three ML models with all the sets of measures presented in Table 5. The average accuracy of each ML model in a 10-step cross-validation process served as a criterion for selecting each model's ideal set of measures. The set of measures #3 was the best for the model that used the Naive Bayes algorithm, the set of measures #1 was the best for the logistic regression model, and the set of measures #4 was the best for the decision tree model. Therefore, in all models, except for the logistic regression, it was possible to decrease the number of independent variables and increase the mean accuracy of the ML model.

The measures most representative to evaluate the FM maintainability are: SCDF, CogC, RDen, NM, FoC, NTop, NLeaf, NF and DTMax. Among these, NF and DTMax stand out as the most representative.

4.2. Among the ML models created in this work, which one is the best to assess FM maintainability? (RQ_2)

To answer **RQ_2**, we performed 10-step cross-validation for the 3 ML models created (using the Naive Bayes algorithm,[4] logistic regression,[4] and decision tree[4]). We used the fourth set of measures shown in Table 5. We calculated the five measures described in Section 2 in each step, and at the end, we obtained the minimum value, the maximum value, the average, and the standard deviation of each measure in the ten steps. Accuracy is the only measure to apply directly to binary classification problems and multiclass problems among the classification measures used. On the other hand, binary classification problems typically use precision, recall, F1, and AUC-ROC measures. Since we are dealing with a multiclass problem, we use the weighted variation of these measures that generate the weighted average of the values of the measures for each class. The weighted variation of these measures considers the imbalance in each class's number in the dataset.

Tables 6–8 show the achieved results. The model that obtained the best results for all measures was the one that used the decision tree algorithm, followed by the Naive Bayes algorithms and logistic regression. All measures were close to or above 0.80 for the decision tree model, including measures for accuracy and precision. The results indicate that the correctness rate of this ML model is higher than that of the FM maintainability classifier generated in our previous work (Silva et al., 2020). Table 9 shows the results of the accuracy, recall, F1-measure, and AUC-ROC measures for each class in the decision tree model.

By comparing the precision, recall, and F1-measure values of the three ML models, it is possible to state that the decision tree model is the least susceptible to false positive and false negative errors. By gazing at the values of the AUC-ROC measure, it is possible to state that among the three ML models, the one that

Table 6
Values of each classification measure in a 10-step cross-validation for the ML model that used the Naive Bayes algorithm.

Measures	Minimum	Maximum	Average	*Std Dev*
Accuracy	0.5	0.67	0.61	0.04
Precision	0.5	0.75	0.65	0.06
Recall	0.5	0.67	0.61	0.04
F1	0.47	0.67	0.61	0.05
AUC-ROC	0.84	0.93	0.88	0.02

Table 7
Values of each classification measure in a 10-step cross-validation for the ML model that used the Logistic Regression algorithm.

Measures	Minimum	Maximum	Average	*Std Dev*
Accuracy	0.52	0.64	0.59	0.03
Precision	0.37	0.68	0.54	0.09
Recall	0.52	0.64	0.59	0.03
F1	0.43	0.57	0.50	0.04
AUC-ROC	0.78	0.91	0.84	0.03

Table 8
Values of each classification measure in a 10-step cross-validation for the ML model that used the Decision Tree algorithm.

Measures	Minimum	Maximum	Average	*Std Dev*
Accuracy	0.73	0.94	0.81	0.06
Precision	0.64	0.95	0.81	0.08
Recall	0.73	0.94	0.81	0.06
F1	0.68	0.94	0.79	0.07
AUC-ROC	0.81	0.96	0.91	0.04

Table 9
Average of the measures precision, recall, F1 and AUC-ROC for each class with the ML model that used the decision tree algorithm.

Class	Precision	Recall	F1	AUC-ROC
Very Bad	0.51	0.64	0.55	0.76
Bad	0.78	0.82	0.79	0.87
Moderate	0.65	0.64	0.64	0.88
Good	0.77	0.78	0.73	0.90
Very Good	0.98	0.92	0.94	0.96

best split the five maintenance classes is also the decision tree model. Cross-validation revealed that the decision tree model has the best hit rate in datasets composed of FMs with different characteristics. Fig. 4 shows the accuracy of each ML model in the 10 stages of cross-validation. The tree model remains with good values at all stages and with better values than the other two models, as Fig. 4 shows. Therefore, we could infer that the decision tree model is generalizable.

Hence, we conclude that the ML model that used the decision tree algorithm is the best among the three ML models for the classification of FM maintainability. It obtained the best values in all five classification measures used. In addition, the decision tree model proved to be the most generalizable, as it obtained the best values for all five classification measures in the cross-validation process.

The best ML model to classify FM maintainability is the one that uses the decision tree algorithm.

4.3. How possible is it to recommend FM refactoring rules based on an ML model created in this work? (RQ_3)

To answer **RQ_3**, we analyzed the main artifact produced by the decision tree model, which is the decision tree. The generated decision tree is a binary tree where each non-leaf node contains a logical decision that indicates whether to go to the right or left node. Each leaf node has a possible classification for the samples.

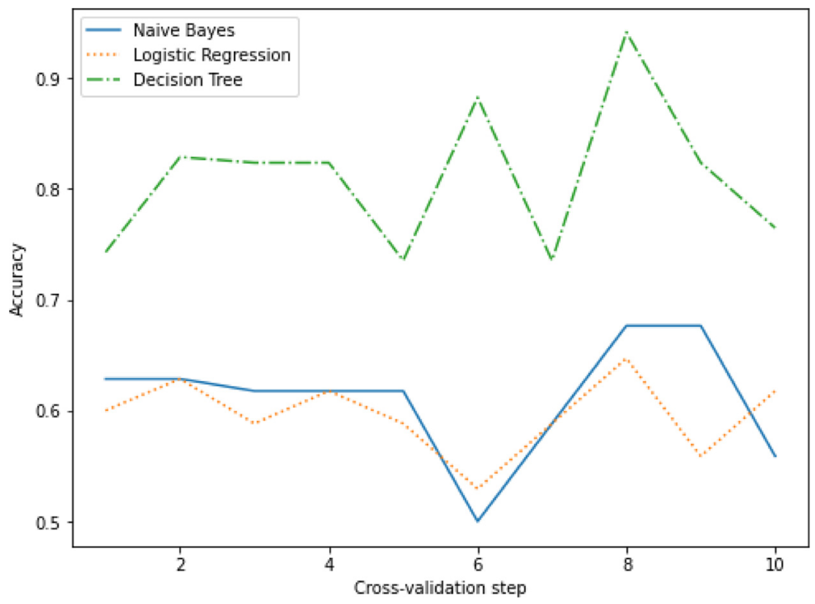


Fig. 4. Accuracy of the 3 ML models at each stage of cross-validation.

Table 10
Actions necessary to respond to each suggestion of a change in the values of the quality measures.

Suggestion of change in the values of quality measures	Actions required to respond to the suggestion
NF $>$ THRESHOLD	Increase the number of features
NF $\leq$ THRESHOLD	Decrease the number of features
NM $>$ THRESHOLD	Increase the number of mandatory features
NM $\leq$ THRESHOLD	Decrease the number of mandatory features
NTop $>$ THRESHOLD	Increase the number of features that descend directly from the root of the tree
NTop $\leq$ THRESHOLD	Decrease the number of features that descend directly from the root of the tree
NLeaf $>$ THRESHOLD	Increase the number of features in the last level of the tree
NLeaf $\leq$ THRESHOLD	Decrease the number of features in the last level of the tree
DTMax $>$ THRESHOLD	Increase the number of tree levels
DTMax $\leq$ THRESHOLD	Decrease the number of tree levels
CogC $>$ THRESHOLD	Increase the number of clusters of type Or or XOr
CogC $\leq$ THRESHOLD	Decrease the number of clusters of type Or or XOr
FoC $>$ THRESHOLD	Increase the number of optional features or decrease the number of features
FoC $\leq$ THRESHOLD	Decrease the number of optional features or decrease the number of features
SCDF $>$ THRESHOLD	Increase the number of child features of XOr clusters
SCDF $\leq$ THRESHOLD	Decrease the number of child features of XOR clusters
RDen $>$ THRESHOLD	Increase the number of features referenced in constraints
RDen $\leq$ THRESHOLD	Decrease the number of features referenced in constraints

Logical decisions at each non-leaf node take INDEPENDENT VARIABLE $\leq$ THRESHOLD. If the decision result is true, proceed to the left node; otherwise, proceed to the right node.

We created an algorithm that identifies the path from the tree root to the leaf node that determines the FM maintainability rating. In addition, the algorithm identifies a new path that results in a better maintainability ranking, continually looking for the least number of changes in the original path. With this new path, it is possible to say which changes in the values of the independent variables are necessary for the maintainability of the FM to improve. Such information enabled us to make suggestions for changes in the values of the independent variables (which are the FM quality measures) following the forms "Make QUALITY MEASURE $\leq$ THRESHOLD" and "Make MEASUREMENT OF QUALITY is $>$THRESHOLD". Therefore, as we used nine quality measures, there are 18 possible types of suggestions for changes in the values of the independent variables.

To facilitate the domain engineer's work in improving the quality of the FM, we have cataloged the necessary actions (which must be applied in the FM) to meet each suggestion of a change in the values of the independent variables (see Table 10). In some cases, the actions are apparent, e.g., for the suggestion NF $>$ THRESHOLD, the action to increase the number of features in the FM. However, in other cases, these actions are not so obvious. For example, for the FoC $>$ THRESHOLD suggestion, there are two possible actions: increase the number of optional features or decrease the number of features.

In our previous work, we provided the domain engineer with suggestions for changes to the values of the quality measures to improve FM maintainability and the actions needed to respond to each suggestion. However, we have not yet been able to provide refactoring suggestions in FM that clearly say what the domain engineer should change in the FM to improve its maintainability.

Several works in the literature describe FM refactorings. In Tanhaei et al. (2016), a set of FM refactorings was defined using the ATL model transformation language. The refactorings described in Tanhaei et al. (2016) use the most typical FM components (features, relationships, groupings, and constraints) but also use new concepts, such as Lowest Common Ancestor (LCA), Mandatory From Ancestor (MFA), and Mandatory From Lowest Common Ancestor (MFLCA). LCA of two features A and B is the lowest, with A and B as descendants. A feature A MFA of a

Table 11
FM refactoring rules with the refactorings that gave rise to them and the additional conditions.

Refactoring rule	Original refactoring rule and additional conditions
R1	Refactoring 5 (Alves et al., 2006)
R2	Refactoring 12 (Alves et al., 2006)
R3	Remove Local Dead Feature 1 (Tanhaei et al., 2016)
R4	Remove Local Dead Feature 2 (Tanhaei et al., 2016)
R5	Remove Local Dead Feature 3 (Tanhaei et al., 2016)
R6	Pull Down 3 (Tanhaei et al., 2016) (If C is optional)
R7	Pull Up 2 (Tanhaei et al., 2016) (If C is optional)
R8	Become Mandatory (Tanhaei et al., 2016)
R9	Migration (Tanhaei et al., 2016) (If C is optional)
R10	Law 3 (Gheyi et al., 2008) (right to left), B-refactoring 3 (Alves et al., 2006) (right to left)
R11	Law 3 (Gheyi et al., 2008) (left to right), B-refactoring 3 (Alves et al., 2006) (left to right)
R12	Refactoring 4 (Alves et al., 2006)
R13	Refactoring 7 (Alves et al., 2006)
R14	Pull Up 1 (Tanhaei et al., 2016)
R15	Pull Up 2 (Tanhaei et al., 2016)
R16	Extract From Or Group (Tanhaei et al., 2016) (If R is root)
R17	Refactoring 5 (Alves et al., 2006) (If A is root)
R18	Refactoring 12 (Alves et al., 2006) (If A is root)
R19	Refactoring 9 (Alves et al., 2006) (If A is root)
R20	Pull Down 1 (Tanhaei et al., 2016)
R21	Pull Down 2 (Tanhaei et al., 2016)
R22	Pull Down 3 (Tanhaei et al., 2016)
R23	Remove Local Dead Feature 1 (Tanhaei et al., 2016) (If B descends directly from the root)
R24	Remove Local Dead Feature 2 (Tanhaei et al., 2016) (If R is root)
R25	Refactoring 10 (Alves et al., 2006) (If A is root)
R26	Pull Down 1 (Tanhaei et al., 2016) (If A and C are leaves)
R27	Pull Down 2 (Tanhaei et al., 2016) (If A and C are leaves)
R28	Pull Down 3 (Tanhaei et al., 2016) (If A and C are leaves)
R29	Remove Local Dead Feature 1 (Tanhaei et al., 2016) (If B is leaf)
R30	Remove Local Dead Feature 2 (Tanhaei et al., 2016) (If C is leaf)
R31	Refactoring 10 (Alves et al., 2006) (If B and C are leaves)
R32	Pull Up 1 (Tanhaei et al., 2016) (If C is leaf)
R33	Pull Up 2 (Tanhaei et al., 2016) (If C is leaf)
R34	Refactoring 9 (Alves et al., 2006) (If C is leaf and only daughter of B)
R35	Pull Down 1 (Tanhaei et al., 2016) (If A is the leaf of the longest path from the root)
R36	Pull Down 2 (Tanhaei et al., 2016) (If A is the leaf of the longest path from the root)
R37	Pull Down 3 (Tanhaei et al., 2016) (If A is the leaf of the longest path from the root)
R38	Migration (Tanhaei et al., 2016) (If A is the leaf of the longest path from the root)
R39	Refactoring 10 (Alves et al., 2006) (If B is the leaf of the longest path from the root)
R40	Refactoring 12 (Alves et al., 2006) (If A is the leaf of the longest path from the root)
R41	Pull Up 1 (Tanhaei et al., 2016) (If C is on the longest path from the root)
R42	Pull Up 2 (Tanhaei et al., 2016) (If C is on the longest path from the root)
R43	Remove Local Dead Feature 1 (Tanhaei et al., 2016) (If B is on the longest path from the root)
R44	Remove Local Dead Feature 2 (Tanhaei et al., 2016) (If C is on the longest path from the root)
R45	Migration (Tanhaei et al., 2016) (If C is on the longest path from the root)
R46	Refactoring 9 (Alves et al., 2006) (If C is on the longest path from the root)
R47	Law 2 (Gheyi et al., 2008) (right to left), B-refactoring (Alves et al., 2006) (right to left)
R48	Refactoring 4 (Alves et al., 2006)
R49	Law 2 (Gheyi et al., 2008) (left to right), B-refactoring (Alves et al., 2006) (left to right)
R50	Refactoring 6 (Alves et al., 2006)
R51	Refactoring 8 (Alves et al., 2006)
R52	Extract From Or Group (Tanhaei et al., 2016)
R53	Refactoring 2 (Alves et al., 2006)
R54	Refactoring 3 (Alves et al., 2006)
R55	Law 1 (Gheyi et al., 2008) (right to left), B-refactoring (Alves et al., 2006) (right to left)
R56	Law 1 (Gheyi et al., 2008) (left to right), B-refactoring (Alves et al., 2006) (left to right)
R57	Refactoring 1 (Alves et al., 2006)
R58	Add Extra Constraints (Tanhaei et al., 2016)
R59	Removes Extra Constraint 1 (Tanhaei et al., 2016)
R60	Removes Extra Constraint 2 (Tanhaei et al., 2016)

feature R, if A is forced to be present in that configuration when selecting R in a configuration (it is represented as MFA(A, R)). A feature A is MFLCA of a feature B if when selecting the LCA of A and B in a configuration, A is forced to be present in that configuration. The refactorings described in Tanhaei et al. (2016) are unidirectional. In Alves et al. (2006), the authors described a set of 16 sound refactorings; 12 of them are unidirectional, and 4 are bidirectional. Some of the refactorings described in Alves et al. (2006) end up resulting in a structure that is not a tree and is therefore valid only for theoretical reasoning and not in practice. In Gheyi et al. (2008), a set of FM refactorings is also described. There are seven refactorings, and all of them are bidirectional. As with Alves et al. (2006), some of the refactorings described in Gheyi et al. (2008) result in a structure that is not a tree and is therefore not useful in practice.

We analyzed the refactorings described in the three studies cited above (Tanhaei et al., 2016; Alves et al., 2006; Gheyi et al., 2008). For each suggestion of changes in the values of quality measures, we chose those that perform the actions described in Table 10. In some cases, adding additional conditions to the refactorings was necessary to ensure that the desired actions were performed. However, these conditions do not add new refactorings, and they only restrict the possibilities that the refactoring already allows. To facilitate understanding, we transformed the bidirectional refactorings into unidirectional ones, creating two unidirectional refactors for each bidirectional refactoring. In addition, we describe how new refactorings for refactorings were required to attach additional conditions. In this way, we obtained a set of 60 refactorings that meet all the suggestions for changes in the values of the quality measures. Figs. A.8–A.11 show the 60 refactoring rules and Fig. A.7 presents the legend for these figures.

Table 11 describes the 60 refactoring rules, the original refactorings that created them, and the conditions added to the original refactorings. Table 12 describes the relationship between the 60 refactoring rules and suggestions for changes in the values of quality measures.

We implemented the FM maintainability evaluation, the suggestions for changing the values of the quality measures, and the refactoring suggestions in the FM based on the suggestions for changing the values of the quality measures in the DyMMer tool[1]. Using the DyMMer tool, the domain engineer will be able to, while editing/modeling the FM, assess the artifact's maintainability and receive refactoring suggestions to improve the maintainability of the FM. This feature allows for a continuous improvement process in the FM, allowing corrective solutions to take from this early stages of the SPL life cycle.

Table 12
Refactorings that can be used to address each suggestion of a change in the values of quality measures.

Suggestion of change in the values of quality measures	Refactoring rule
NF > THRESHOLD	R1, R2
NF ≤ THRESHOLD	R3, R4, R5
NM > THRESHOLD	R5, R6, R7, R8, R9, R10
NM ≤ THRESHOLD	R11, R12, R13
NTop > THRESHOLD	R14, R15, R16, R17, R18, R19
NTop ≤ THRESHOLD	R20, R21, R22, R23, R24, R25
NLeaf > THRESHOLD	R1, R2, R32, R33, R34
NLeaf ≤ THRESHOLD	R26, R27, R28, R29, R30, R31
DTMax > THRESHOLD	R35, R36, R37, R38, R39, R40
DTMax ≤ THRESHOLD	R41, R42, R43, R44, R45, R46
CogC > THRESHOLD	R47, R48
CogC ≤ THRESHOLD	R5, R49, R50, R51
FoC > THRESHOLD	R3, R4, R5, R11, R13, R52, R49, R50, R51
FoC ≤ THRESHOLD	R1, R10, R6, R8, R9, R47, R53, R54
SCDF > THRESHOLD	R1, R55
SCDF ≤ THRESHOLD	R4, R5, R51, R54, R56, R57
RDen > THRESHOLD	R11, R14, R49, R56, R58
RDen ≤ THRESHOLD	R3, R4, R8, R10, R20, R47, R55, R59, R60

Table 13
Time to evaluate each FM.

No. features	Time
10	0.823 ms
100	0.835 ms
1000	0.824 ms
10,000	0.829 ms
100,000	0.839 ms

4.4. How scalable is the FM maintainability assessment using an ML model proposed in this work? (RQ_4)

To answer **RQ_4**, we used the decision tree model to evaluate the maintainability of FMs of different sizes. As for the performance test, the semantics of the FM was not relevant, but only the size and complexity of the FM were important. Therefore, we decided to generate five FMs using the FM generator available in the SPLOT repository. They had 10, 100, 1,000, 10,000, and 100,000 features each. Then we used the DyMMer tool to calculate the values of the nine maintainability measures used by the decision tree model for the 5 FMs.

After that, we used the decision tree ML model to assess the maintainability of the five FM and collect the time taken to each evaluation. We performed the evaluation using a workstation with a Windows 10 operating system, a 7th generation Intel Core I5 processor, and 8 GB of RAM. All the artifacts that we used in the evaluation (including the 5 FMs evaluated and the algorithm used in the measurement of the evaluation time) can be found on GitHub.[9]

Table 13 presents the measurement results. As can be seen, the time taken to evaluate each of the 5 FMs was practically constant. In our initial assumptions, we expected this result since after training the decision tree ML model, we needed to use the decision tree (a static artifact that remains unchanged until we train the model with new samples) to assess maintainability based on maintainability measures. Furthermore, the evaluation algorithm already receives the calculated measurement values, and because of that, the evaluation does not consider the time to extract the maintainability measurements. The results indicate that the decision tree model is scalable and can evaluate the maintainability of FMs of the most varied sizes without performance loss.

9 https://github.com/publiosilva/JSSSPLC2021/tree/main/performance-measurement.

4.5. Using the ML model to classify FM maintainability

We integrated the FM maintainability evaluation and the refactoring suggestions into the DyMMer tool. To evaluate an FM in the DyMMer tool, enter the FM editing screen and navigate to the "Quality Assessment" tab. As you can see in Fig. 5, this tab shows the maintainability of the FM, a suggestion of a change in the value of a quality measure, and also FM refactoring suggestions to cause this change in the value of the quality measure and consequently improve the maintainability of the FM. After making a change in the FM (in the "Feature Editor" tab), the user can navigate back to the quality evaluation tab and click on the "Re-evaluate" button. The DyMMer tool will re-evaluate the FM considering the changes made.

This FM quality assessment mechanism built into the DyMMer tool allows domain engineers to quickly and efficiently assess the serviceability of an FM. This allows corrective action to be taken early in the life cycle of an SPL, preventing errors from spreading across the entire product line. After modifying an FM, one domain engineer can quickly and effectively assess the FM again to verify if the FM maintainability has improved. In doing so, the domain engineer also receives new refactoring suggestions for the FM that enable a continuous improvement process of the SPL.

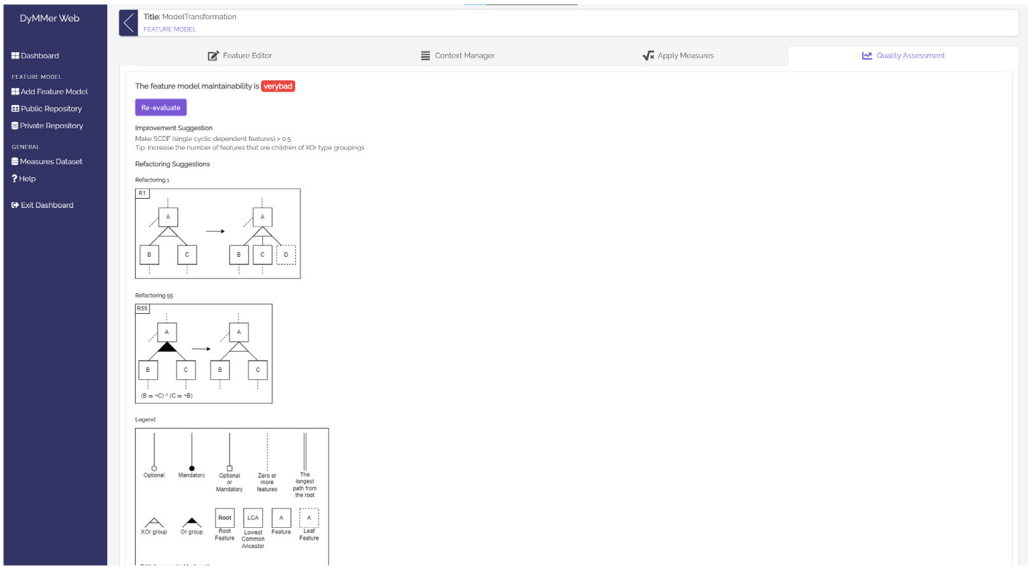
4.6. Discussion

We created 3 ML models to classify the maintainability of FMs using the white box ML algorithms: Naive Bayes, logistic regression, and decision tree. The 3 ML models created were compared using classification accuracy, precision, recall, F1, and AUC-ROC. Also, 10-step cross-validation was performed to verify how generalizable the models created are. We calculated the five classification measures in 10 steps for each ML model, and at the end, we obtained the average of the classification measures in the cross-validation steps. Considering the average of the values of each measure in the ten stages of cross-validation, the ML model that obtained the best results was the one that used the decision tree algorithm, which reached an accuracy of 0.81, precision of 0.81, recall of 0.81, F1 0.79 and AUC-ROC of 0.91. These values indicate that the model is less susceptible to false-negative and false-positive errors. It also manages to better separate between the five maintenance classes of FMs. These values also demonstrate that the FM maintainability classification approach developed in this work obtained better accuracy and precision than the approach developed in our previous work (Silva et al., 2020), where we got an accuracy of 0.61 and precision of 0.7. It is not possible to make the same comparison with Oliveira's approach, as the work of Oliveira and Bezerra (2019) does not provide accuracy or precision values.

The decision tree model used only 9 out of the 15 maintainability measures in the MiniCOfFEE (Bezerra et al., 2017) catalog; therefore, we achieved a reduction of 6 independent variables. The decrease in the problem's dimensionality makes the ML model simpler and potentially more scalable and improves the model's performance.

ML model based on the decision tree algorithm created in this work allows domain engineers and anyone interested in improving SPL to quickly and easily assess the maintainability of an FM. This allows quick corrective actions to be taken, which are already in the early stages of an SPL's life cycle. Early evaluation can prevent errors in the FM from spreading throughout the SPL. Also, as the decision tree model is a white box, information can be extracted to improve SPL.

To further assist the domain engineer in the SPL improvement process, we use the decision tree generated by the decision tree algorithm to generate suggestions for changes in the values of



Title: ModelTransformation
FEATURE MODEL

The feature model maintainability is verybad

Improvement Suggestion
Make SCDF (single cyclic dependent features) > 0.5
Tip: Increase the number of features that are children of XOr type groupings

Refactoring Suggestions

Refactoring 1

Refactoring 55

Legend

Fig. 5. The FM quality assessment implemented in the DyMMer tool.

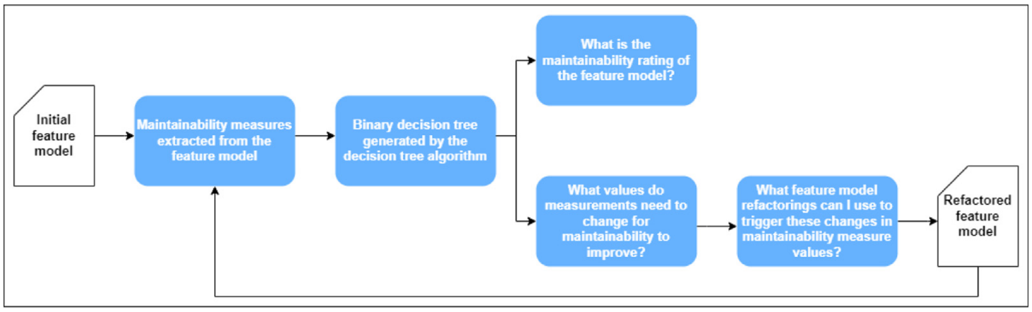

Initial feature model
Maintainability measures extracted from the feature model
Binary decision tree generated by the decision tree algorithm
What is the maintainability rating of the feature model?
What values do measurements need to change for maintainability to improve?
What feature model refactorings can I use to trigger these changes in maintainability measure values?
Refactored feature model

Fig. 6. FM evaluation and refactoring process.

the independent variables that are the quality measures. We then relate these suggested changes to the values of the quality measures with a set of refactorings. Thus, besides evaluating the degree of maintainability of the FM, we were also able to provide suggestions for refactoring the FM. With this, the domain engineer can clearly know the actions needed to improve the maintainability of the FM.

Fig. 6 shows the FM evaluation and refactoring process and clarifies how one could use it for continuously improving an FM. Nine measures of maintainability are taken from the original FM. These measurements are received by the decision tree model that returns the FM maintainability rating. Furthermore, with the analysis of the decision tree model, it is possible to interpret the results and know how the nine measures should be modified so that the maintainability of the FM improves. By knowing this, it is possible to choose FM refactorings that cause the desired changes in the values of the maintainability measures. After performing the refactorings on the original FM, a new FM is obtained that can also be reevaluated. This creates a process for evaluating and continuously improving FM maintainability.

We implemented the FM maintainability evaluation and the refactoring suggestions in the DyMMer tool. With the DyMMer tool, the domain engineer can assess the FM maintainability while editing the artifact.

We also performed a performance run we did not evaluate FM maintainability. With the evaluation, we observed that the time was practically constant for FM of different sizes (ranging from 10 to 100,000 features). With this, we conclude that this FM maintainability assessment strategy is scalable.

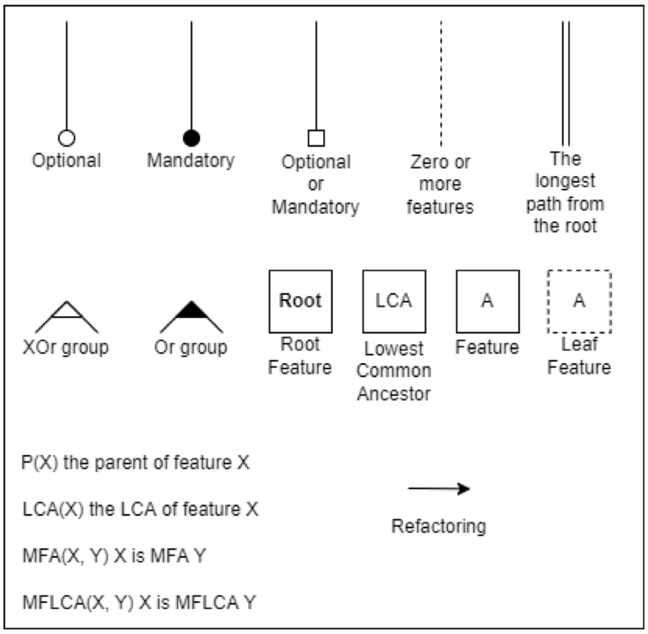


Fig. A.7. Legend for all refactorings presented in Figs. A.8, A.9, A.10, and A.11.

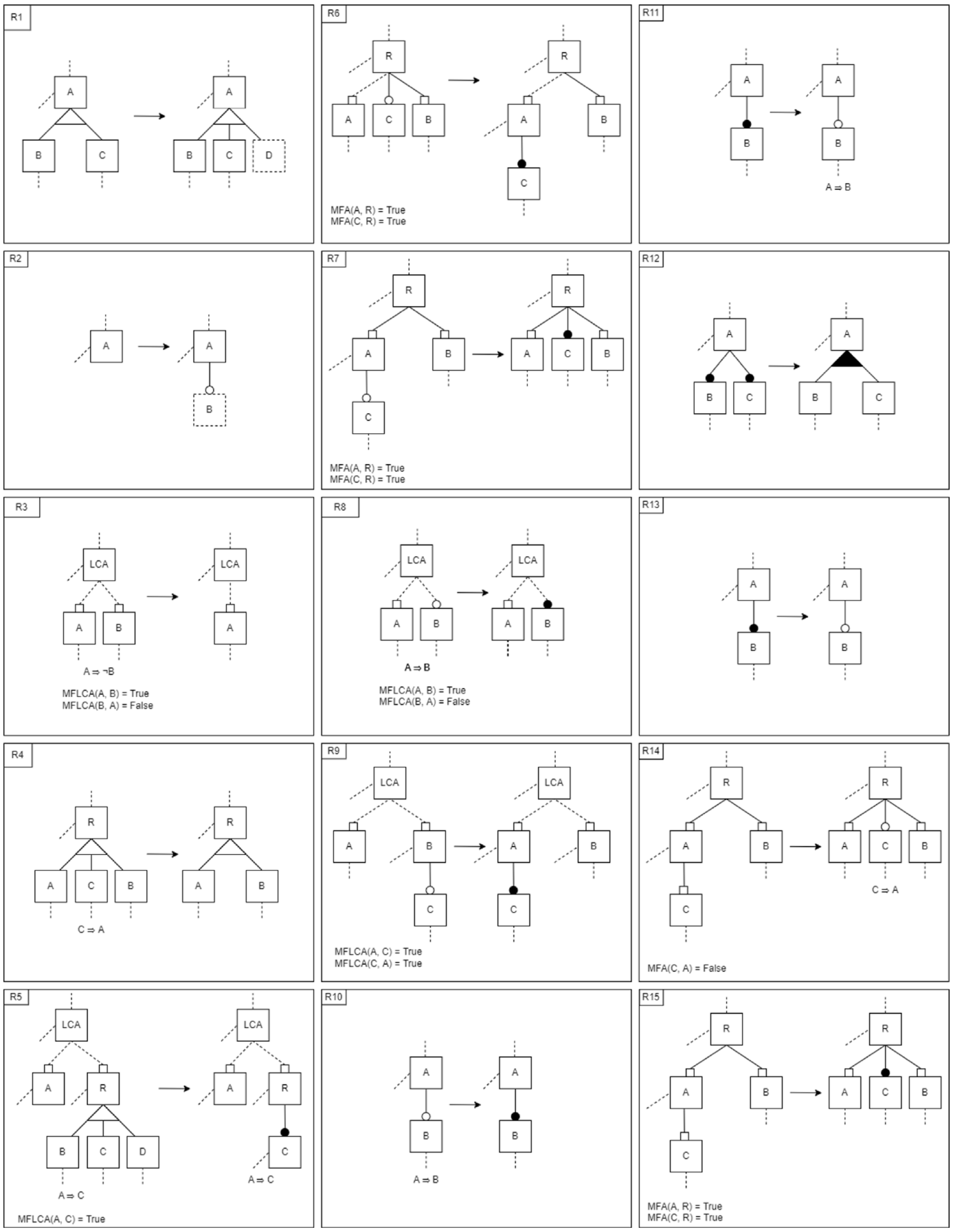


Fig. A.8. Refactorings 1 to 15.

5. Threats to validity

Internal Validity. All FM maintainability measures were automatically collected using the DyMMer tool to mitigate the threat to internal validity. Also, the maintainability classification of each sample in the dataset was collected automatically using a Java code made available by Oliveira's approach (Oliveira and Bezerra, 2019). A concern about internal validity is that we made few adjustments to the hyperparameters of ML algorithms. Therefore, we aim to make greater adjustments and optimizations in hyperparameters to increase ML models' quality as future work. Another concern is that refactoring suggestions do not necessarily affect a single quality measure. Because of this, it can happen that when implementing a refactoring, the value quality measure that you did not want to change changes. It could end up making the maintainability worse instead of better. We intend to study ways to exclude refactoring suggestions that can worsen the FM maintainability in future works.

Construct validity. The main concern to the construction validity is selecting the appropriate independent variables to classify the maintainability of the FMs. To mitigate this threat, we use two approaches to selecting independent variables. We analyzed the correlation between the FM maintainability measures and the FM maintainability classification in the first approach. In the second approach, we analyze the degree of importance of the independent variables for the created ML models. With both approaches, we obtained some sets of tested measures in the three ML models created. We selected the set of measures that maximize accuracy for each ML model, thus ensuring that we selected each case's best set of measures. When analyzing the importance

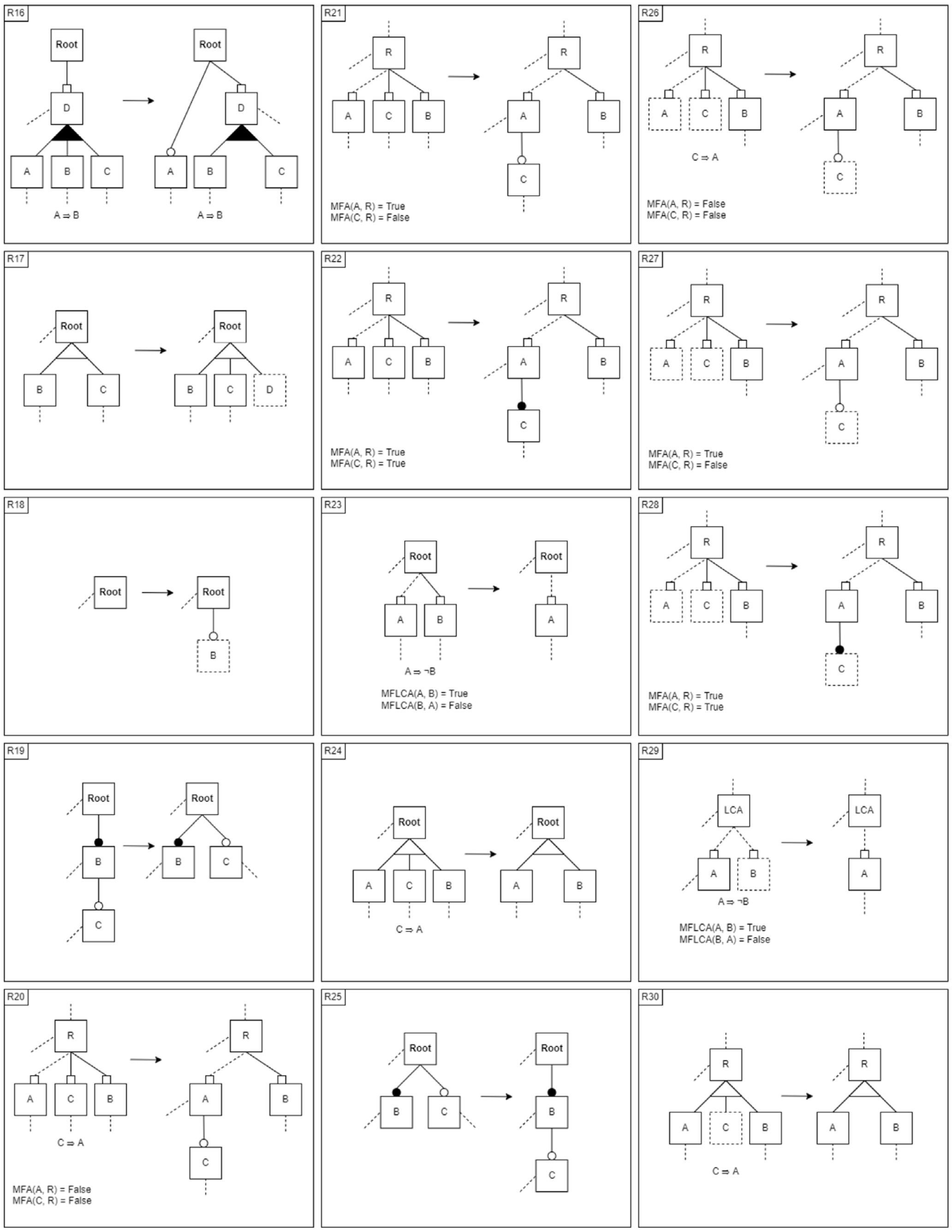


Fig. A.9. Refactorings 16 to 30.

of independent variables for ML models, we do not consider the Naive Bayes algorithm. This algorithm does not directly provide an attribute of importance for the variables. Thus, as future work, we intend to use agnostic model approaches (e.g., Feature Permutation Importance) to measure the importance of independent variables.

External Validity. External validity is concerned with the extent to which the results obtained can be generalized. To mitigate the threat to external validity, we use a dataset composed of FMs of different sizes and characteristics. Also, we use 10-step cross-validation to verify that ML are generalizable, that is, how well they behave in datasets with different characteristics. One concern is that most FM on the dataset was academic, so in future work, we plan to add FM used in industry as the 127 real-world FM provided in Knüppel et al. (2017). Another concern is that the group of experts that evaluated the FMs is composed of only 15 people and is not as heterogeneous as desired (most experts dealt with SPL only in academia). That is due to the difficulty in gathering a large and heterogeneous group of SPL experts willing to participate in the research. Besides, the analysis and evaluation of an FM is an activity that takes some time, which makes it hard to gather a large group of people willing to perform this task. Therefore, in future work we plan to include a more heterogeneous set of experts, particularly from industry.

Conclusion validity. To mitigate threats related to the validity of the conclusion and increase the confidence of the results, we compared the ML models created using five classification measures. The measures we used in this study consider different aspects of an ML model, such as the model's susceptibility to false positive or false negative errors and how well each model

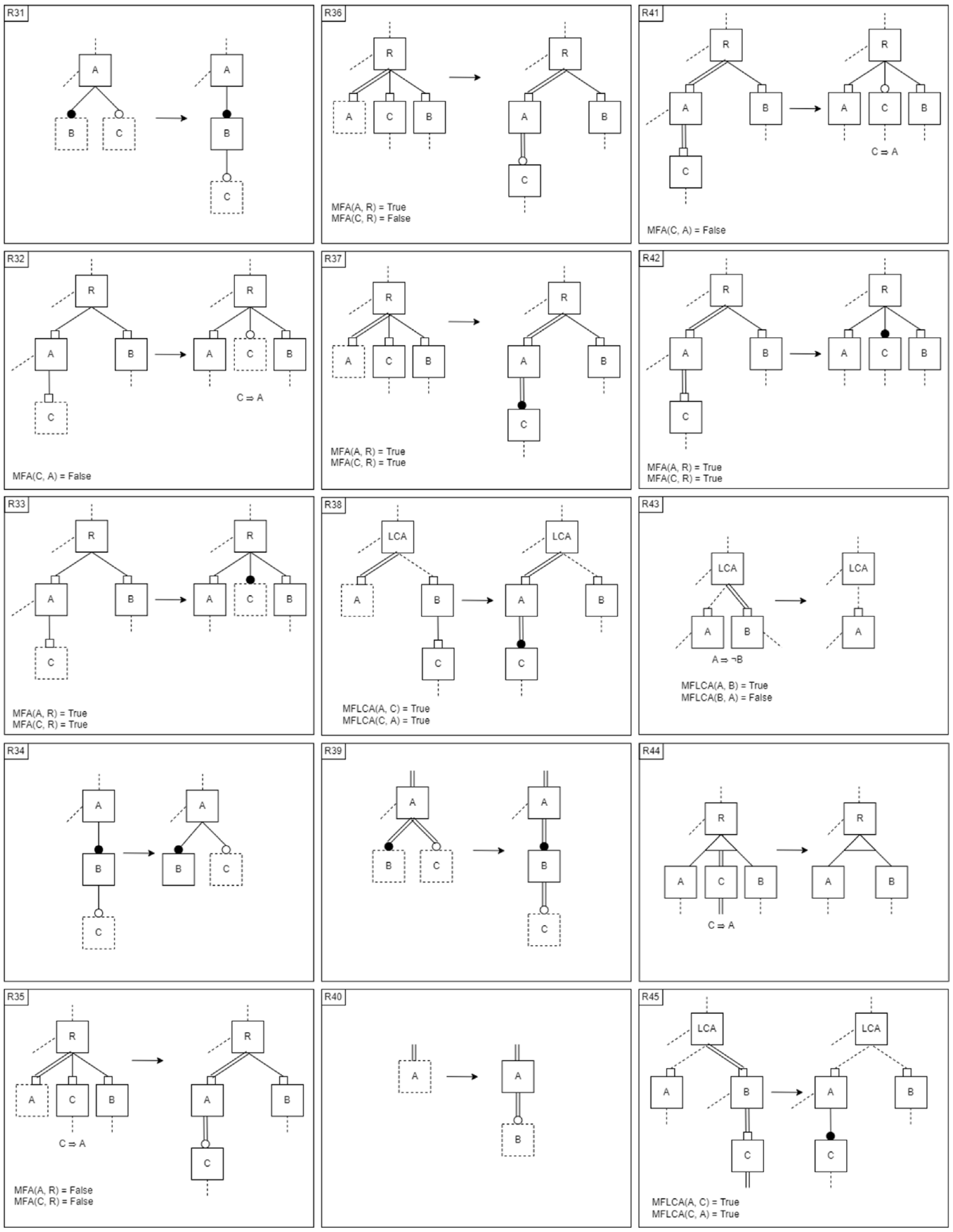


Fig. A.10. Refactorings 31 to 45.

separates the existing classes. We perform cross-validation of 10 steps, and in each step, we calculate the 5 classification measures. In the end, we calculated the average of each measure in the ten steps and used these values to compare the ML models.

6. Conclusion

This work is an extension of a previous work (Silva et al., 2021) in which we used white box ML algorithms to create ML models capable of classifying FM maintainability from a set of measures. In this work, we included an evaluation of the scalability of the use of an ML model to evaluate the maintainability of FM, the use of the results of the assessment made by the ML model to suggest FM refactorings to improve the maintainability of the FM and we implemented the FM assessment and the FM refactoring suggestions to the DyMMer tool.

We selected two approaches for automatic classification of FM maintainability as candidates to pre-classify the dataset: Silva's approach (Silva et al., 2020), and Oliveira's approach (Oliveira and Bezerra, 2019). We compared the two approaches using a comparison oracle composed of 28 FMs assessed manually by 15 SPL experts. The result of the comparison showed that: (i) the Oliveira's approach outperformed Silva's approach in terms of classification with the same class defined by the experts; (ii) when both approaches do not classify with the same class defined by experts in general, they classify with more pessimism; and, (iii) in general, the classifications obtained with the Oliveira's approach are closer to the classifications defined by the experts.

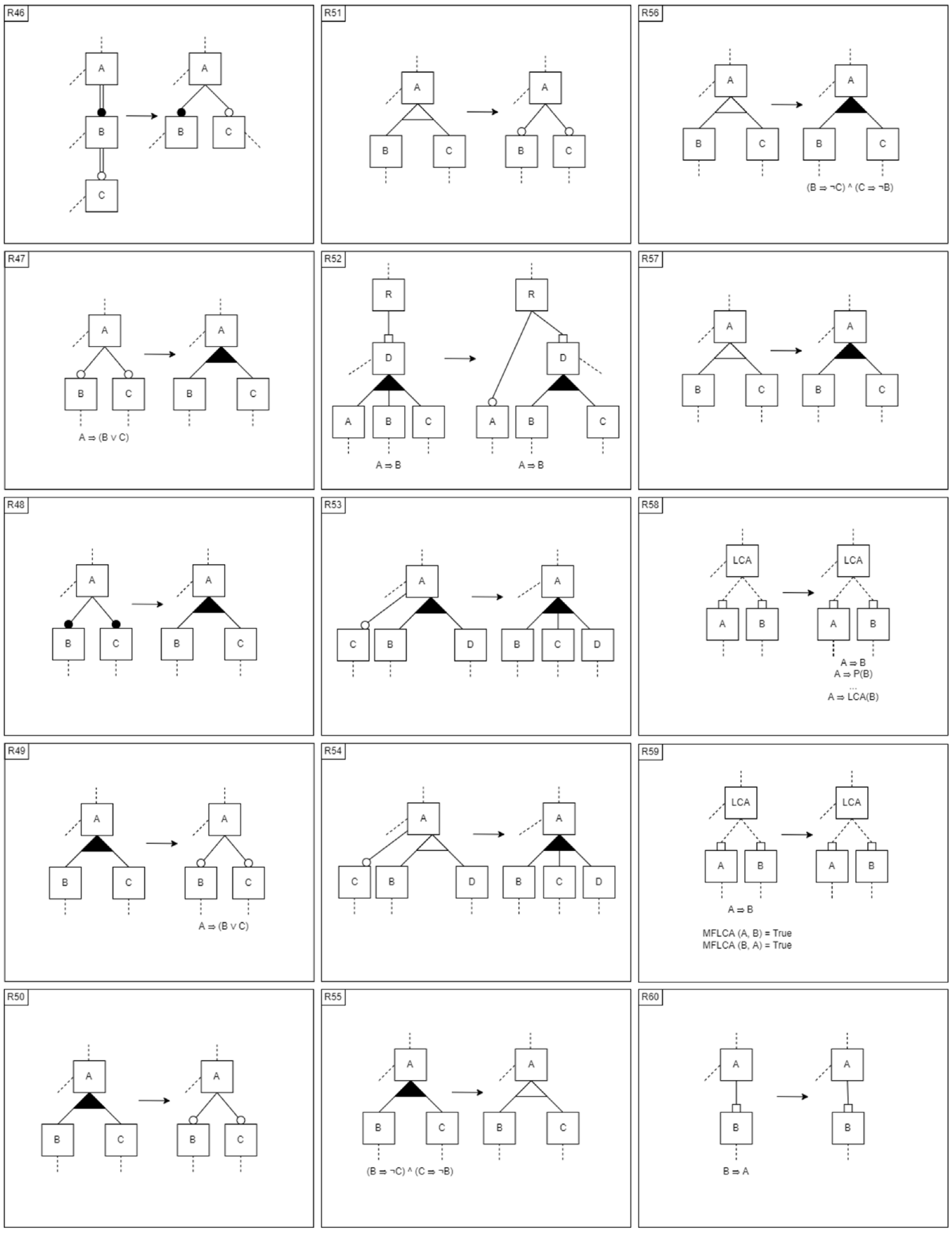


Fig. A.11. Refactorings 46 to 60.

We used Oliveira's approach to pre-classify the dataset, as it obtained better results in the comparison.

We created three ML models to classify FM maintainability: (1) Naive Bayes, (2) logistic regression, and (3) decision tree. We used five classification measures to compare the ML models: accuracy, precision, recall, F1, and AUC-ROC. We used a 10-step cross-validation process. For each step, we calculated the five measures, and at the end, their average. Using the decision tree algorithm, the ML model obtained the best results, with an average accuracy of 0.81, an average precision of 0.81, an average recall of 0.81, an average F1-score of 0.79, and an average AUC-ROC of 0.91. These values indicate that the ML model is not susceptible to false negative and false positive errors and that it can separate the five maintainability classes of FM well. With the decision tree model, it was necessary to use only nine of the fifteen measures from the MiniCOfFFE catalog, reducing six independent variables. The ML model we created allows domain engineers and anyone interested in improving the SPL to quickly assess the maintainability of an FM, facilitating the taking of corrective actions at the beginning of the cycle.

We also measured the performance of the FM maintainability evaluation algorithm and concluded that the approach is scalable. In addition, we use the decision tree to generate suggestions for changes in the values of maintainability measures and relate each suggestion to one or more FM refactoring rules. We were able to indicate to the domain engineer what changes should be made to the FM so that the maintainability of the artifact improves. We implemented both the FM maintainability evaluation and the refactoring suggestions in the DyMMer tool. It provides domain engineers with a feasible strategy to assess the maintainability and take corrective actions while editing the FM.

We envision opportunities for further studies, as follows: (i) using the information extracted from the decision tree model to create a mechanism capable of providing suggestions for improving FM maintainability; (ii) making more adjustments and optimizations in the hyperparameters of ML algorithms to increase the quality of the ML models obtained; (iii) using agnostic model approaches to measure the importance of variables in the selection of independent variables; (iv) devise an approach to exclude refactoring rules with the potential to worsen, rather than improve, FM maintainability; (v) implement automatic detection of opportunities and refactoring suggestions in FM in the DyMMer tool; and (vi) implement the FM maintainability assessment also considering the ontological aspects of the artifact.

Declaration of competing interest

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

Data availability

I have shared the link to my data/code of the attach file step.

Appendix A. Refactorings rules

See Figs. A.7–A.11.

Appendix B. Feature model evaluation form

For each question, the respondent must mark out an appropriate answer (in a 5-point Likert scale: Very Bad, Bad, Moderate, Good, and Very Good) choice that he/she thinks is correct.

Q1. Evaluate feature model "Simple Drawing Tools" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/46). Select the category in which the feature model fits in terms of maintainability.

Q2. Evaluate feature model "e-commerce" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/16). Select the category in which the feature model fits in terms of maintainability.

Q3. Evaluate feature model "Speech Recognition" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/83). Select the category in which the feature model fits in terms of maintainability.

Q4. Evaluate feature model "PhotoSahring" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/31). Select the category in which the feature model fits in terms of maintainability.

Q5. Evaluate feature model "Windows70_Contrast" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/99). Select the category in which the feature model fits in terms of maintainability.

Q6. Evaluate feature model "Owncloud" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/95). Select the category in which the feature model fits in terms of maintainability.

Q7. Evaluate feature model "USECASE_Test" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/24). Select the category in which the feature model fits in terms of maintainability.

Q8. Evaluate feature model "WebApp" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/15). Select the category in which the feature model fits in terms of maintainability.

Q9. Evaluate feature model "VMTools-RA" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/65). Select the category in which the feature model fits in terms of maintainability.

Q10. Evaluate feature model "Jogo de Tiros" for maintainability (https://publiosilva.github.io/feature-model-assessment/#/100). Select the category in which the feature model fits in terms of maintainability.

References

Acher, M., Baudry, B., Heymans, P., Cleve, A., Hainaut, J.-L., 2013. Support for reverse engineering and maintaining feature models. In: Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '13, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2430502.2430530.

Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., Aljaaf, A.J., 2020. A systematic review on supervised and unsupervised machine learning algorithms for data science. In: Berry, M.W., Mohamed, A., Yap, B.W. (Eds.), Supervised and Unsupervised Learning for Data Science. Springer International Publishing, Cham, pp. 3–21. http://dx.doi.org/10.1007/978-3-030-22475-2_1.

Alpaydin, E., 2020. Introduction to Machine Learning. MIT Press.

Alsolai, H., Roper, M., 2020. A systematic literature review of machine learning techniques for software maintainability prediction. Inf. Softw. Technol. 119, 106214. http://dx.doi.org/10.1016/j.infsof.2019.106214, URL: https://www.sciencedirect.com/science/article/pii/S0950584919302228.

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C., 2006. Refactoring product lines. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering. GPCE '06, Association for Computing Machinery, New York, NY, USA, pp. 201–210. http://dx.doi.org/10.1145/1173706.1173737.

Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

Bagheri, E., Gasevic, D., 2011. Assessing the maintainability of software product line feature models using structural metrics. Softw. Qual. J. 19 (3), 579–612. http://dx.doi.org/10.1007/s11219-010-9127-2.

Bailey, G., Joffrion, A., Pearson, M., 2018. A comparison of machine learning applications across professional sectors. Available At SSRN 3174123.

Batory, D., 2005. Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (Eds.), Software Product Lines. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 7–20.

Bénard, C., Biau, G., da Veiga, S., Scornet, E., 2021. Interpretable random forests via rule extraction. In: Banerjee, A., Fukumizu, K. (Eds.), Proceedings of the 24th International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, vol. 130, PMLR, pp. 937–945, URL: https://proceedings.mlr.press/v130/benard21a.html.

Berger, T., Guo, J., 2014a. Towards system analysis with variability model metrics. In: Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '14, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2556624.2556641.

Berger, T., Guo, J., 2014b. Towards system analysis with variability model metrics. In: Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '14, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2556624.2556641.

Bezerra, C.I.M., Andrade, R.M.C., Monteiro, J.M.S., 2014. Measures for quality evaluation of feature models. In: Schaefer, I., Stamelos, I. (Eds.), Software Reuse for Dynamic Systems in the Cloud and beyond. Springer International Publishing, Cham, pp. 282–297.

Bezerra, C.I., Andrade, R.M., Monteiro, J.M., 2017. Exploring quality measures for the evaluation of feature models: a case study. J. Syst. Softw. 131, 366–385. http://dx.doi.org/10.1016/j.jss.2016.07.040, URL: https://www.sciencedirect.com/science/article/pii/S0164121216301340.

Bezerra, C.I.M., Andrade, R.M.C., Monteiro, J.M.S., Cedraz, D., 2018. Aggregating measures using fuzzy logic for evaluating feature models. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems. In: VAMOS 2018, Association for Computing Machinery, New York, NY, USA, pp. 35–42. http://dx.doi.org/10.1145/3168365.3168375.

Bezerra, C.I.M., Barbosa, J., Freires, J.H., Andrade, R.M.C., Monteiro, J.M., 2016a. DyMMer: A measurement-based tool to support quality evaluation of DSPL feature models. In: Proceedings of the 20th International Systems and Software Product Line Conference. SPLC '16, Association for Computing Machinery, New York, NY, USA, pp. 314–317. http://dx.doi.org/10.1145/2934466.2962730.

Bezerra, C., Lima, R., Silva, P., 2021. DyMMer 2.0: A tool for dynamic modeling and evaluation of feature model. In: Brazilian Symposium on Software Engineering. Association for Computing Machinery, New York, NY, USA, pp. 121–126, URL: https://doi.org/10.1145/3474624.3476016.

Bezerra, C.I.M., Monteiro, J.M., Andrade, R.M.C., Rocha, L.S., 2016b. Analyzing the feature models maintainability over their evolution process: An exploratory study. In: Proceedings of the Tenth International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '16, Association for Computing Machinery, New York, NY, USA, pp. 17–24. http://dx.doi.org/10.1145/2866614.2866617.

Bonaccorso, G., 2017. Machine Learning Algorithms. Packt Publishing Ltd.

Browne, M.W., 2000. Cross-validation methods. J. Math. Psych. 44 (1), 108–132. http://dx.doi.org/10.1006/jmps.1999.1279, URL: https://www.sciencedirect.com/science/article/pii/S0022249699912798.

Bürdek, J., Kehrer, T., Lochau, M., Reuling, D., Kelter, U., Schürr, A., 2016. Reasoning about product-line evolution using complex feature model differences. Autom. Softw. Eng. 23 (4), 687–733. http://dx.doi.org/10.1007/s10515-015-0185-3.

Chicco, D., Jurman, G., 2020. The advantages of the matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genom. 21 (1), 6. http://dx.doi.org/10.1186/s12864-019-6413-7.

Clements, P., Northrop, L., 2002. Software Product Lines. Addison-Wesley Boston.

El Naqa, I., Murphy, M.J., 2015. What is machine learning? In: El Naqa, I., Li, R., Murphy, M.J. (Eds.), Machine Learning in Radiation Oncology: Theory and Applications. Springer International Publishing, Cham, pp. 3–11. http://dx.doi.org/10.1007/978-3-319-18305-3_1.

El-Sharkawy, S., Krafczyk, A., Schmid, K., 2019a. MetricHaven: More than 23,000 metrics for measuring quality attributes of software product lines. In: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B. SPLC '19, Association for Computing Machinery, New York, NY, USA, pp. 25–28. http://dx.doi.org/10.1145/3307630.3342384.

El-Sharkawy, S., Krafczyk, A., Schmid, K., 2020. Fast static analyses of software product lines: An example with more than 42,000 metrics. In: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems. VAMOS '20, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3377024.3377031.

El-Sharkawy, S., Yamagishi-Eichler, N., Schmid, K., 2019b. Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. Inf. Softw. Technol. 106, 1–30. http://dx.doi.org/10.1016/j.infsof.2018.08.015, URL: https://www.sciencedirect.com/science/article/pii/S0950584918301873.

Gheyi, R., Massoni, T., Borba, P., 2008. Algebraic laws for feature models. J. UCS 14, 3573–3591.

Greenwell, B.M., Boehmke, B., Gray, B., 2020. Variable importance plots—An introduction to the vip package. R J. 12 (1), 343–366.

Jain, A.K., 2010. Data clustering: 50 years beyond K-means. Pattern Recognit. Lett. 31 (8), 651–666. http://dx.doi.org/10.1016/j.patrec.2009.09.011, URL: https://www.sciencedirect.com/science/article/pii/S0167865509002323, Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

Jha, S., Kumar, R., Hoang Son, L., Abdel-Basset, M., Priyadarshini, I., Sharma, R., Viet Long, H., 2019. Deep learning approach for software maintainability metrics prediction. IEEE Access 7, 61840–61855. http://dx.doi.org/10.1109/ACCESS.2019.2913349.

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Klir, G., Yuan, B., 1995. Fuzzy Sets and Fuzzy Logic, Vol. 4. Prentice hall New Jersey.

Knüppel, A., Thüm, T., Mennicke, S., Meinicke, J., Schaefer, I., 2017. Is there a mismatch between real-world feature models and product-line research? In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2017, Association for Computing Machinery, New York, NY, USA, pp. 291–302. http://dx.doi.org/10.1145/3106237.3106252.

Lima, L., Uchôa, A., Bezerra, C., Coutinho, E., Rocha, L., 2020. Visualizing the maintainability of feature models in SPLs. In: Anais Do VIII Workshop de VisualizaÇÃo, EvoluÇÃo E ManutenÇÃo de Software. SBC, Porto Alegre, RS, Brasil, pp. 1–8. http://dx.doi.org/10.5753/vem.2020.14522, URL: https://sol.sbc.org.br/index.php/vem/article/view/14522.

Loyola-González, O., 2019. Black-box vs. White-Box: Understanding their advantages and weaknesses from a practical point of view. IEEE Access 7, 154096–154113. http://dx.doi.org/10.1109/ACCESS.2019.2949286.

Ma, Y., Chen, W., Ma, X., Xu, J., Huang, X., Maciejewski, R., Tung, A.K.H., 2017. EasySVM: A visual analysis approach for open-box support vector machines. Comput. Vis. Media 3 (2), 161–175. http://dx.doi.org/10.1007/s41095-017-0077-5.

Maggio, V., 2013. Improving Software Maintenance using Unsupervised Machine Learning Techniques (Ph.D. thesis). University of Naples Federico II, Italy, URL: http://www.fedoa.unina.it/9079/.

Marques, M., Simmonds, J., Rossel, P.O., a Cecilia Bastarrica, M., 2019. Software product line evolution: A systematic literature review. Inf. Softw. Technol. 105, 190–208. http://dx.doi.org/10.1016/j.infsof.2018.08.014, URL: https://www.sciencedirect.com/science/article/pii/S0950584918301848.

Marsland, S., 2015. Machine Learning. CRC Press.

Mendonca, M., Branco, M., Cowan, D., 2009. S.P.L.O.T.: Software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. OOPSLA '09, Association for Computing Machinery, New York, NY, USA, pp. 761–762. http://dx.doi.org/10.1145/1639950.1640002.

Montagud, S., Abrahão, S., Insfran, E., 2012. A systematic review of quality attributes and measures for software product lines. Softw. Qual. J. 20 (3), 425–486. http://dx.doi.org/10.1007/s11219-011-9146-7.

Narkhede, S., 2018. Understanding auc-roc curve. Towards Data Sci. 26, 220–227.

Oliveira, D.C.S., Bezerra, C.I.M., 2019. Development of the maintainability index for SPLs feature models using fuzzy logic. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering. In: SBES 2019, Association for Computing Machinery, New York, NY, USA, pp. 357–366. http://dx.doi.org/10.1145/3350768.3351299.

Passos, L., Czarnecki, K., Apel, S., Wąsowski, A., Kästner, C., Guo, J., 2013. Feature-oriented software evolution. In: Proceedings of the Seventh International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '13, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/2430502.2430526.

Pereira, J.A., Acher, M., Martin, H., Jézéquel, J.-M., Botterweck, G., Ventresque, A., 2021. Learning software configuration spaces: A systematic literature review. J. Syst. Softw. 182, 111044. http://dx.doi.org/10.1016/j.jss.2021.111044, URL: https://www.sciencedirect.com/science/article/pii/S0164121221001412.

Refaeilzadeh, P., Tang, L., Liu, H., 2016. Cross-validation. In: Liu, L., Özsu, M.T. (Eds.), Encyclopedia of Database Systems. Springer New York, New York, NY, pp. 1–7. http://dx.doi.org/10.1007/978-1-4899-7993-3_565-2.

Rocha, L., Machado, I., Almeida, E., Kästner, C., Nadi, S., 2020. A semi-automated iterative process for detecting feature interactions. In: Proceedings of the 34th Brazilian Symposium on Software Engineering. SBES '20, Association for Computing Machinery, New York, NY, USA, pp. 778–787. http://dx.doi.org/10.1145/3422392.3422418.

Salkind, N.J., Rainwater, T., 2006. Exploring Research. Pearson Prentice Hall Upper Saddle River, NJ.

Schober, P., Boer, C., Schwarte, L.A., 2018. Correlation coefficients: appropriate use and interpretation. Anesth. Analg. 126 (5), 1763–1768.

Sen, P.C., Hajra, M., Ghosh, M., 2020. Supervised classification algorithms in machine learning: A survey and review. In: Mandal, J.K., Bhattacharya, D. (Eds.), Emerging Technology in Modelling and Graphics. Springer Singapore, Singapore, pp. 99–111.

Silva, P., Bezerra, C.I.M., Lima, R., Machado, I., 2020. Classifying feature models maintainability based on machine learning algorithms. In: Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse. SBCARS '20, Association for Computing Machinery, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3425269.3425276.

Silva, P., Bezerra, C.I.M., Machado, I., 2021. A machine learning model to classify the feature model maintainability. In: Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume a. Association for Computing Machinery, New York, NY, USA, pp. 35–45, URL: https://doi.org/10.1145/3461001.3471152.

Soares, L.R., Schobbens, P.-Y., do Carmo Machado, I., de Almeida, E.S., 2018. Feature interaction in software product line engineering: A systematic mapping study. Inf. Softw. Technol. 98, 44–58. http://dx.doi.org/10.1016/j.infsof.2018.01.016, URL: https://www.sciencedirect.com/science/article/pii/S0950584917302690.

Syakur, M.A., Khotimah, B.K., Rochman, E.M.S., Satoto, B.D., 2018. Integration K-means clustering method and elbow method for identification of the best customer profile cluster. IOP Conf. Ser.: Mater. Sci. Eng. 336, 012017. http://dx.doi.org/10.1088/1757-899x/336/1/012017.

Tanhaei, M., Habibi, J., Mirian-Hosseinabadi, S.-H., 2016. Automating feature model refactoring: A model transformation approach. Inf. Softw. Technol. 80, 138–157. http://dx.doi.org/10.1016/j.infsof.2016.08.011, URL: https://www.sciencedirect.com/science/article/pii/S0950584916301422.

Temple, P., Galindo, J.A., Acher, M., Jézéquel, J.-M., 2016. Using machine learning to infer constraints for product lines. In: Proceedings of the 20th International Systems and Software Product Line Conference. SPLC '16, Association for Computing Machinery, New York, NY, USA, pp. 209–218. http://dx.doi.org/10.1145/2934466.2934472.

Vale, G., Fernandes, E., Figueiredo, E., 2019. On the proposal and evaluation of a benchmark-based threshold derivation method. Softw. Qual. J. 27 (1), 275–306. http://dx.doi.org/10.1007/s11219-018-9405-y.

Zhang, H., Wang, M., 2009. Search for the smallest random forest. Stat. Interface 2 (3), 381. http://dx.doi.org/10.4310/sii.2009.v2.n3.a11, URL: https://pubmed.ncbi.nlm.nih.gov/20165560, 20165560[pmid].

Públio Silva holds a Bachelor Degree in Computer Science. He received Summa Cum Laude Bachelor degree from Federal University of Ceará (UFC), Brazil, in 2021. His research interests include Software Product Lines Engineering, Variability Management, Software Quality, and Machine Learning. He is currently a Software Engineer at Casa Magalhães, a software company based in Brazil.

Carla Bezerra is an adjunct professor in the Federal University of Ceará (UFC), Brazil, since 2010. She received her Ph.D. degree in Computer Science from UFC, in 2016. Her research interests are software product line, software quality, software evolution and maintenance, and mining software repositories. She is the Leader of the LEAN, a Software Engineering Research Group at UFC.

Ivan Machado is an Adjunct Professor at the Institute of Computing at the Federal University of Bahia (UFBA), in Salvador, Brazil. He holds a Ph.D. in Computer Science from UFBA and a M.Sc. in Computer Science from UFPE. His research interests include software product lines engineering, software testing and analysis, software startups, sustainable software engineering, and mining software repositories. He is the Leader of the ARIES LAB, a Software Engineering Research Group at UFBA. Ivan serves and have served as a program committee member of various conferences and as a reviewer for recognized journals (e.g., IEEE TSE, IEEE Access, ACM TOSEM, JSS, IST, EMSE, SQJ).