Classifying Changes to LabVIEW and Simulink Models via Changeset Metrics

Saheed Popoola[1*], Xin Zhao[2], Jeff Gray[3] and Antonio Garcia-Dominguez[4]

[1*]School of Information Technology, University of Cincinnati, Cincinnati, USA.
[2]Department of Computer Science, Seattle University, Seattle, USA.
[3]Department of Computer Science, University of Alabama, Tuscaloosa, USA.
[4]Department of Computer Science,University of York, York, UK.

*Corresponding author(s). E-mail(s): saheed.popoola@uc.edu;
Contributing authors: xzhao1@seattleu.edu; gray@cs.ua.edu;
a.garcia-dominguez@york.ac.uk;

Abstract

Automated classification of software changes can help to understand the reason why a change was made. Support for the classification of changes can also guide the adoption of quality control practices as bugfix trends are observed, and cluster related sets of changes for similar management of the changed artifacts, thereby reducing maintenance efforts. A number of change classification techniques have been developed based on information extracted from the change author, change message, change size, or changed file. However, most of these approaches have targeted textual general-purpose programming languages. Furthermore, some of these approaches are computationally expensive because they often require the analysis of the whole source code, while others rely on the developers' ability to describe a commit via a well-written message. In this paper, we present an approach to classify changes to models into the appropriate maintenance type via a set of metrics that are extracted from the version history of models. We developed seven metrics related to changes applied to models and model elements. We then conducted an empirical study involving 10 classifiers to determine the classifier that offers the best performance for automating the change classification process. These classifiers were trained on over 300 changesets extracted from the version history of 28 Simulink repositories, and 60 changesets from 10 LabVIEW repositories. The results of the study show that the Random Forest classifier offers the best performance for Simulink models, while the Bayes Net offers the best performance for LabVIEW models. The Random Forest classifier has also been evaluated by comparing its results with labels extracted from the discussions within the issues reported in a similar time frame. The evaluation results show that the Random Forest classifier is able to achieve an F-1 score of 0.83, thereby showing its ability to classify changes into the appropriate categories intended by the original developers.

Keywords: change classification, changeset metrics, LabVIEW, Simulink, classifier

1 Introduction

Models continuously evolve to satisfy new requirements, improve the performance of the system being modelled, or correct some anomalies. This evolution involves the changes to models and model elements. The analysis of these changes is important to guide the software development process and reduce maintenance efforts. For example, change analysis has been used to predict future changes, identify defect-prone sections of code, group related and dependent software components for similar maintenance activities, and help in decision-making activities [32, 38, 48]. A major task in change analysis is the classification of related changes [56]. In this paper, we define the aggregate set of changes in a version as a *changeset*.

The classification of changes into its maintenance types helps to understand the reason behind a change, supports diverse analysis regarding a changeset, and helps with future decision-making tasks [26, 37, 56]. For example, knowing that a changeset deals with corrective maintenance activities such as fixing a bug can help in understanding why the change set was implemented in the first place. An increase in corrective maintenance tasks might also be an indicator that it is time to focus more on quality assurance practices. Unfortunately, changesets are often not labeled with their types, and manual classification of changes is time-consuming and error-prone.

Software repositories such as GitHub provide capabilities for tracking the changes made to software. The history log of these repositories provides information related to the changes committed to the repository, such as who made the change, the changed files, a description of the change (in unstructured plain text), and other documentation. The widespread adoption and easy accessibility of these repositories often provide a rich source of data to analyze the history of changes to software. However, these repositories do not provide automated techniques for systematically classifying the changes [31].

A number of approaches have been developed for classifying changesets. These approaches use diverse information, such as changed files, change hunks, change author, and change message [4, 26]. The information related to the authors of a change is often not enough to classify a changeset, hence this information is often used to augment other change classification approaches. A well-written change message often conveys the reason behind a change and can significantly help the change classification process [43]. However, change messages are subjective and may contain trivial or irrelevant information. Change hunk metrics involve information related to the structure of a change, such as the number of loops, methods, variables, and null pointers affected by the change. Unfortunately, change hunk metrics and change files have only been validated when classifying changes related to text-based general-purpose programming languages. Furthermore, many of the existing approaches have focused on one or two maintenance categories, e.g., whether the changes are corrective or non-corrective [32].

This paper presents an approach for classifying changes over the history of a model repository. We extracted 7 sets of primitive change metrics from the version history of model repositories. The metrics include model changes (adding/removing models), changes to model elements (adding/removing/reordering elements), and changes to the attributes of the model elements. These change metrics have been used to train a set of classifiers to label a changeset related to a specific commit in the model repository history with its type of maintenance. This paper makes the following contributions:

1. We present an approach for classifying changes to model repositories into their maintenance types.
2. We conducted an empirical study, where we trained 10 classifiers to assess the performance of the classifiers in predicting the type of a change. The 10 classifiers have been chosen from the WEKA toolset [17].
3. We introduce a novel evaluation method by comparing the performance of a change classifier with the labels associated with discussion in the issue logs.

This paper is an extended version of the conference paper [46]. The original analysis targeted only Simulink models [2]. This paper expands the analysis to LabVIEW models [30]. An overview of LabVIEW models is provided in Section 2.2. LabVIEW models have distinct syntax and semantics from Simulink models, and both modeling languages tend to target different applications. We also explore the performance of the classifiers on LabVIEW models when trained on Simulink models. We added two new research questions in Section 3. Hence, we have enhanced the scope and the context of the original analysis.

The remaining sections of the paper are as follows. Section 2 introduces the background tools used in this paper to mine Simulink repositories, while Section 3 provides an overview of the research questions that motivated this research. Section 4 discusses how the background tools have been used to extract changeset metrics across successive versions of Simulink models. Section 5 discusses our empirical study to assess the performance of 10 classifiers in predicting the maintenance type associated with each changeset. Section 6 presents our approach to evaluate a change classifier by comparing its performance with the manual classification by the original contributors via the discussions in the issue logs. Section 7 summarizes the limitations of this study and the possible threats that may affect the validity of the results presented in this paper. Section 8 overviews the related work and also highlights how the research presented in this paper extends existing change classification approaches. Finally, Section 9 concludes this paper and highlights our future research directions.

2 A Toolchain for Change Extraction

This section introduces the five tools that have been used in this work. These tools are Simulink for modeling dynamic systems, Massif [28] for converting Simulink models to EMF-based models, LabVIEW for modeling test instruments, Eclipse Hawk [18] for indexing the version history of fragmented models stored in file-based repositories, and EMF Compare [50] for extracting the differences across two sets of models.

2.1 Simulink and Massif

Simulink by MathWorks is an extensible, graphical programming environment that supports the analysis, modeling and simulation of dynamic systems. Simulink is part of the MATLAB tool and it also provides support for diverse model management tasks such as automated code generation, testing, and model verification. Simulink provides a graphical editor and extensible libraries that can be used to design models of systems and run different simulations on the models. Fig. 1 is a graphical representation of a Simulink model that captures the motion of a car after the speed pedal is pressed and the resulting position of the car relative to its starting point. The model contains five blocks connected by lines. The pulse generator produces the input signal to indicate the pressing of the speed pedal, while the gain block multiplies the input signal by a defined factor to indicate the resultant effect of pressing the speed pedal on the car's acceleration. The second-order integrator performs the integration of the input signal twice to calculate the relative position of the car based on the acceleration. The two output blocks designate the position of the car relative to its starting point.

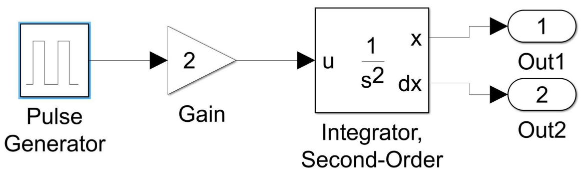

Fig. 1: A Simulink Model of a Car Acceleration System

Massif integrates the Eclipse-based EMF tools and the MATLAB/Simulink framework via Java commands in the MATLAB API. The tool supports bi-directional transformations from Simulink models to EMF-based models, as well as user-based configurations to guide the transformations. Each transformation is achieved in Massif by first connecting to the MATLAB Engine, then sending a series of commands to the MATLAB API to either retrieve useful information about the Simulink model in order to construct an appropriate EMF representation of the model, or to construct a new Simulink model based on the properties of an EMF model. The EMF models that are supported by the Massif tool must conform to an Ecore-based metamodel that captures most of the essential elements in a Simulink model. Figure 2 is the graphical representation of Massif's EMF representation of the Simulink model in Figure 1.

2.2 LabVIEW

The Laboratory Virtual Instrument Engineering Workbench (LabVIEW) [30] by National Instruments is an extensible, graphical programming environment that supports the development, analysis and validation of software systems that require fast access to hardware data. LabVIEW is widely adopted by hundreds of thousands of users

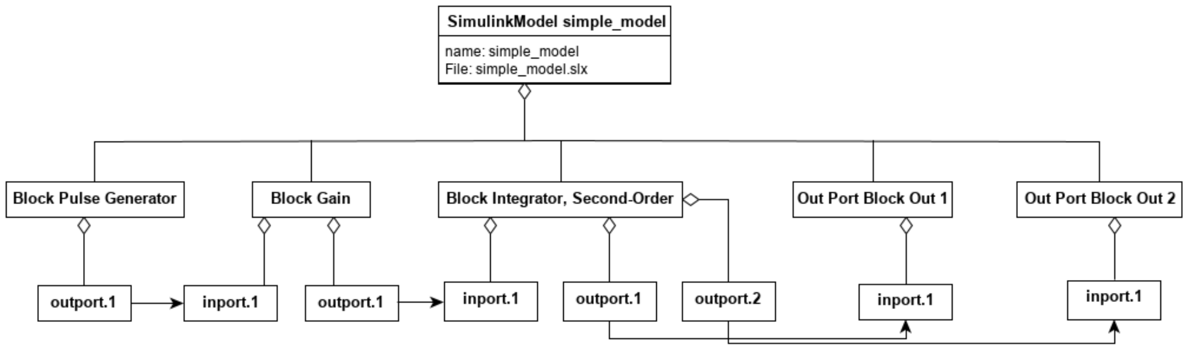

Fig. 2: Massif EMF's Representation of a Simulink Model

in more than 26,000 companies to develop software that spans diverse domains [1, 14]. LabVIEW supports many third-party applications and provides capabilities for extending the tool for custom interfaces and automated user commands. A LabVIEW model is built out of Virtual Instruments (VIs) and contains two main parts: a front panel that provides the user interface and a block diagram where users write the graphical code. Figure 3 is the front-end and block-diagram of a sample VI that converts the temperature value in Fahrenheit to its appropriate value in Celsius.

Traditionally, LabVIEW models were stored in a proprietary binary format, thereby making it challenging to analyze LabVIEW models via third-party automated tools. A recent version (LabVIEW NXG) stores a model in XMI format, thereby making it easier for third-party tools to analyze such models. This paper focuses on LabVIEW NXG models.

2.3 Hawk

Hawk [7] is a scalable model indexing framework that can be used to query models that are distributed over a large number of files, such as those that are stored in file-based repositories such as the Git version control system (VCS). A VCS like Git can track changes that are made to files, support a large number of users, and handle conflicts in the changes made to files by different users. However, a VCS does not provide support for managing the relationships between the model elements stored in the files, leaving that complexity to the user [8]. Although a number of model-centric repositories have been developed, they have not been widely adopted by practitioners: some lack features such as branching and tagging, while others re-implement similar functionality (e.g., user management, locking, checking out a model), or only work seamlessly with their own modeling tool. These repositories tend to lack the widespread tool support (e.g., continuous integration systems) of traditional file-based VCS.

Hawk provides a model-centric layer over file-based repositories, thereby combining the model-centric querying and navigation capabilities of model repositories with the maturity and widespread use of file-based repositories. Hawk can answer arbitrary queries in a dialect of the Epsilon Object Language [40], report changes across files in the repository, and present views of the indexed models as read-only EMF models. Hawk follows a component-based architecture that can be extended in various ways. Fig. 4 provides a graphical overview of Hawk. The version control manager component in Hawk retrieves file-based changes from a VCS, and the model resource factory component parses the files into an in-memory model abstraction (inspired by EMF, but independent from it). The model updater component compares the in-memory models against the current state of the backend component (currently, one of several database management systems), and performs an incremental update on the backend to reflect the latest state of the models. For large models fragmented across files, Hawk will save memory by first indexing each fragment separately and then reconnecting the fragments. After the backend is updated, it is ready to answer questions via query engine components.

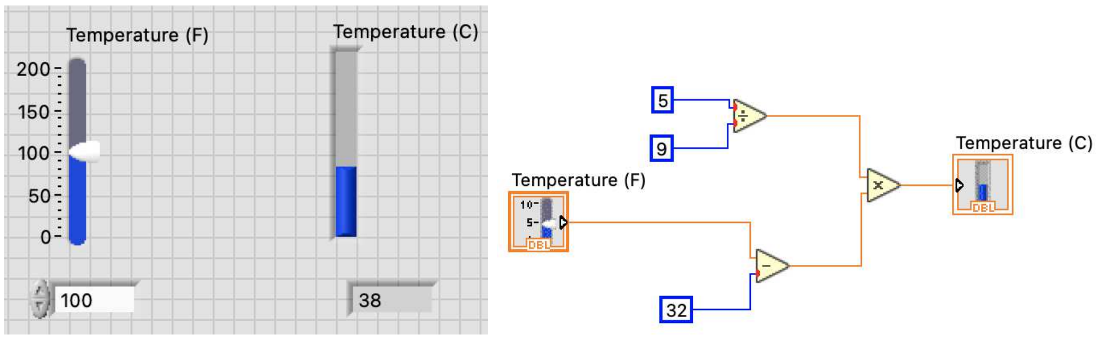

Fig. 3: A Front Panel and Block Diagram for Temperature Conversion

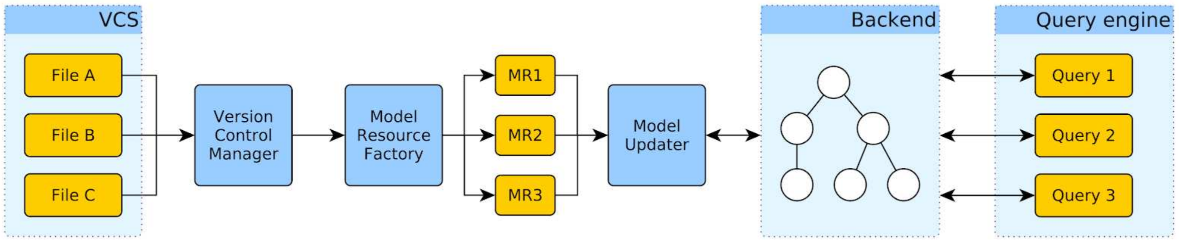

Fig. 4: Overview of the Eclipse Hawk workflow [6]

In its default configuration, Hawk only records the most recent version of the models in the repository. Hawk can be configured to use time-aware versions of the above components [18], providing the ability to record the full history of the indexed models into a temporal graph [23]. Each version of the model is associated with a time point that corresponds to when the change was made. A time-aware dialect of EOL is provided to query historic information about the models, such as when the models started exhibiting some behavior, or when a model element was first added to the repository. These time-aware extensions allow for efficient investigation and analysis of the evolution of models in file-based repositories.

2.4 EMF Compare

EMF Compare is an Eclipse-based project that supports two- and three-way differencing and merging of EMF-based models. EMF Compare provides distinct phases of the comparison process that can be extended and customized toward user-specific needs. For example, while the default method of detecting if elements in different models are the same is based on the similarities of their features, EMF Compare can be told to use static identifiers to match elements.

EMF Compare provides two approaches for extracting differences between models: a two-way differencing approach that extracts differences directly between two sets of models, and a three-way differencing approach that extracts the differences between two models based on their evolution from a common third base model. EMF Compare also supports four kinds of differences between two models: ADD to show the addition of new elements, DELETE to indicate the removal of elements, MOVE to show the reordering of elements, and CHANGE to indicate changes in the attributes of elements.

3 Research Questions

The main goal of this research is to automate the classification of changes to LabVIEW and Simulink models in order to provide insights on why the change was made. We did this by extracting changes from open-source LabVIEW and Simulink repositories on GitHub, developing a set of metrics on each change, and using the developed metrics to automatically determine the change type via a set of classifiers. We specifically aim to answer the following four research questions.

1. **RQ1:** Which classifier(s) best predict the maintenance type for changes in Simulink models?
2. **RQ2:** Which classifier(s) best predict the maintenance type for changes in LabVIEW models?
3. **RQ3:** Can a classifier trained on Simulink models accurately predict the maintenance type for changes in LabVIEW models (and vice-versa)?
4. **RQ4:** What is the performance of the best-performing classifier in terms of precision, recall, and F1-score, over the labels associated with some of the design decisions in the issues reported in the repositories?

The original paper was focused on RQ1 and RQ4. This paper extends the classification analysis with RQ2 and RQ3. Section 5.3 answers RQ1, RQ2, and RQ3, while Section 6 answers RQ4.

4 Mining Repositories and Change Extractions

This section discusses how we extracted changes from the evolution history of a set of Simulink models in 28 GitHub repositories and LabVIEW models in 10 repositories. First, we search for relevant Simulink repositories on GitHub via the keyword "simulink." We then manually searched through each repository in the search results and extracted the repositories with at least 10 commits, and the original search results were filtered down to approximately 100 repositories. We used Hawk to query Simulink models in each repository, and finally selected models with more than five version histories. A version is defined as a commit that involves changes to at least one Simulink file. This exclusion criteria was established to ensure that the selected repositories have sufficient evolution history to capture diverse types of changes, and not only the addition of new models. The final selection consists of 28 repositories with 537 versions across many industrial and academic domains. The same process was repeated for LabVIEW repositories (except that the search keyword was "labview nxg"). However, most LabVIEW repositories contained LabVIEW models persisted in the traditional binary format, and only a few repositories were associated with the NXG format. Unfortunately, the current version of the extended Hawk tool supports only the NXG version. Therefore, we selected almost all of the NXG-based LabVIEW repositories.

Table 1 and Table 2 provide a statistical overview of the selected repositories. The tables show the total number of commits, the average number of commits, the minimum number of commits in any of the repositories, the maximum number of commits in a single repository, the average number of commits across all of the repositories, and the median number of commits. The same set of statistical data was also extracted for the number of versions, models, model elements, branches, and contributors. It can be seen from the tables that this study analyzes 451 models containing an aggregate of over four million model elements for Simulink and 76 models containing an aggregate of over thirteen thousand model elements for LabVIEW.

Fig. 5 provides a graphical summary of the domains of the Simulink repositories. The figure shows that the robotics and avionics domains have the highest number of repositories. Furthermore, we could not assign two repositories to a particular domain; therefore, we labeled them as others. In total, the 28 repositories span across 12 different domains, thereby showing the heterogeneity of the repositories used in this study, which offers a more diverse dataset for the classification. We could not verify the domains for the LabVIEW repositories.

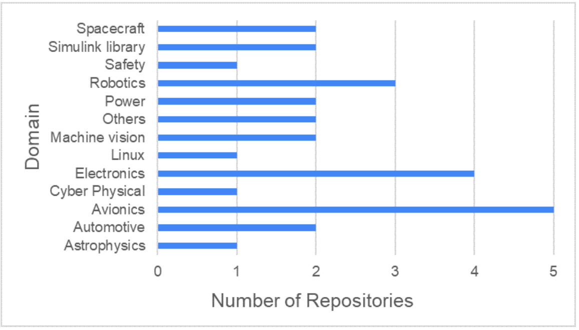

Fig. 5: Domain Distribution in the Simulink Repositories

Properties \Variables	Commits	Version	Models	Model elements	Branches	Contributors
Total	5310.0	537.0	451.0	3061796.0	139.0	66.0
Average	189.6	19.3	16.1	109349.8	4.9	2.4
Minimum	20.0	5.0	1.0	911.0	1.0	1.0
Maximum	1168.0	90.0	174.0	830193.0	48.0	8.0
Median	69.0	11.0	8.0	57085.0	1.0	1.0

Table 1: Overview of Selected Simulink Repositories

Properties \Variables	Commits	Version	Models	Model elements	Branches	Contributors
Total	744.0	52.0	76.0	13246.0	13.0	31.0
Average	74.4.6	5.2	7.6	1324.6	1.3	3.1
Minimum	4.0	2.0	1.0	171.0	1.0	1.0
Maximum	593.0	11.0	15.0	5537.0	2.0	18.0
Median	12.5	3.5	7.0	667.0	1.0	1.0

Table 2: Overview of Selected LabVIEW Repositories

The full details about each of the repositories considered and the final repositories selected for this research are provided here [1]. The remaining part of this section discusses how we mined LabVIEW and Simulink models from the selected GitHub repositories via the integration of Hawk and Massif, with the extraction of the changes via EMF Compare.

4.1 Mining/Extracting Data with Hawk and Massif

Figure 6 shows how the time-aware version of Hawk has been extended to support the indexing of LabVIEW and Simulink models. The extension includes a model parser to parse LabVIEW and Simulink files, and a language to support high-level queries over model versions. The model parser detects LabVIEW and Simulink files via their respective file extensions. For LabVIEW files, the parser extracts the model elements and converts them into an EMF resource. The resulting EMF resource is validated against a LabVIEW metamodel developed in a previous work [45]. For Simulink files, the parser transforms Simulink models into an appropriate EMF resource via the integration of Hawk with the Massif framework. The generated EMF resource for LabVIEW and Simulink files is then passed to Hawk for indexing and storage in a temporal graph database. Successive versions of the same model can also be passed to EMF Compare to generate additional change data between the versions. The query language extends the default time-aware query language provided by Hawk to offer new capabilities for querying the indexed models to extract the necessary change data needed for change classification tasks. The set of extensions provided by the query language include the following:

1. Model-level queries. This provides information about the models (via the file name) that have been added, deleted or modified. Sample queries include a query to return the total number of models added at a particular timepoint.
2. EMF Compare changes. This provides information about changes between two successive versions based on the results produced by EMF Compare. Sample queries include a query to extract all the changes between two timepoints (which may contain multiple versions).

4.2 Extracting Changes with EMF Compare

A change analyzer has been developed and integrated with the extended Hawk framework in order to support the extraction of changes across the version history of models. Change has always

[1]https://bit.ly/ChangeAnalysisSAM

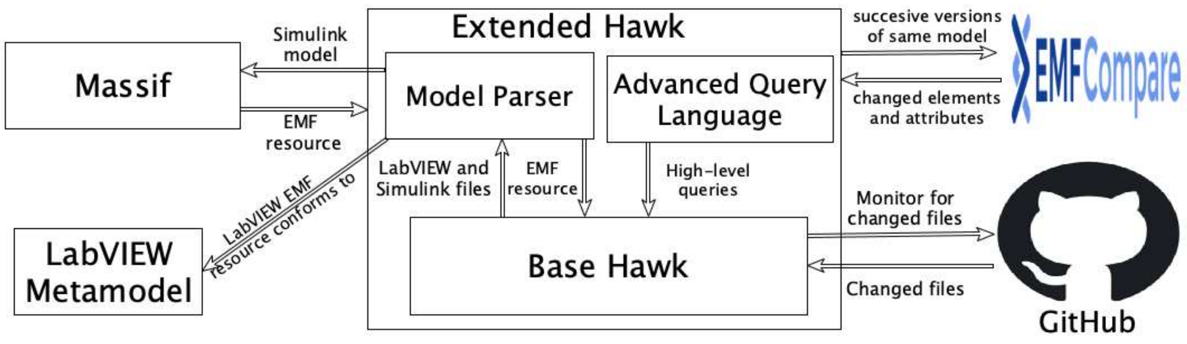

Fig. 6: The Extended Hawk Framework

```
Let A= set of models in prior version
Let B= set of models in following version
If A intersection B is empty
    Added models= All B
    Deleted Models = All A
Else
    ChangedModel = (A union B) - (A intersection
      B)
    Added models = ChangedModel intersection B
    Deleted models = ChangedModel intersection A
```

Listing 1: Differencing for model-level changes

been recognized as an important factor in software evolution and many sets of metrics have been developed to quantify and analyze changes to software systems. We adapted existing change metrics defined by Wen et al. [54] to fit changes to models. In particular, we considered six operations that characterize the evolution of a set of models from version A to version B. These operations are grouped into three categories: changes to the models (adding/removing models), changes to model elements (adding/removing/reordering elements), and changes to the attributes of the model elements.

The changes were extracted in two phases. The first phase involves checking for the addition or deletion of models, while the second phase involves extraction of the changes in successive versions of models via EMF Compare. In the first phase, we extracted the high-level changes to models in the repository by comparing the models via the file name property of the models that is captured by the Simulink metamodel provided by Massif. The sets of added and deleted models between each pair of adjacent versions are computed as in Listing 1.

The second phase involves extracting changes at the level of model elements and attributes via the integration of EMF Compare with our extended Hawk framework. In this phase, the set of common models in two adjacent versions are entered as inputs into EMF Compare. The kind of differences (i.e., ADD, DELETE, CHANGE, or MOVE) and the affected model element/attributes are then extracted from the results produced by EMF Compare. If any difference is detected during this phase, the model is also added to a list of modified models. Table 3 shows the total number of extracted changes to the models and model elements in the selected Simulink and LabVIEW repositories. The table captures the total number of changesets, added model elements, deleted model elements, changed or modified model elements, moved model elements, added models, changed models, and deleted models.

Figure 7 is a simplified model of an hospital that has been modified from an initial version to the final version. The initial version of the model is composed of two patients and one doctor. In the final version, one patient has been deleted and the role of the doctor has been modified. For the initial version, the change extraction process will capture the addition of one model and three elements. For the final version, the process will capture the modification of one model, the deletion of one model element (Patient Sam), and the modification of one model element (Doctor Jeff). It should be noted that the processes described in Section 4.1

Environment\ Variables	Change-set	Added Element	Deleted Element	Changed Element	Moved Element	Added Model	Deleted Model	Changed Model
Simulink	360	5146360	3533164	249584	431368	228	76	812
LabVIEW	44	20176	369	884	174	120	63	55

Table 3: Summary of Total Extracted Changes

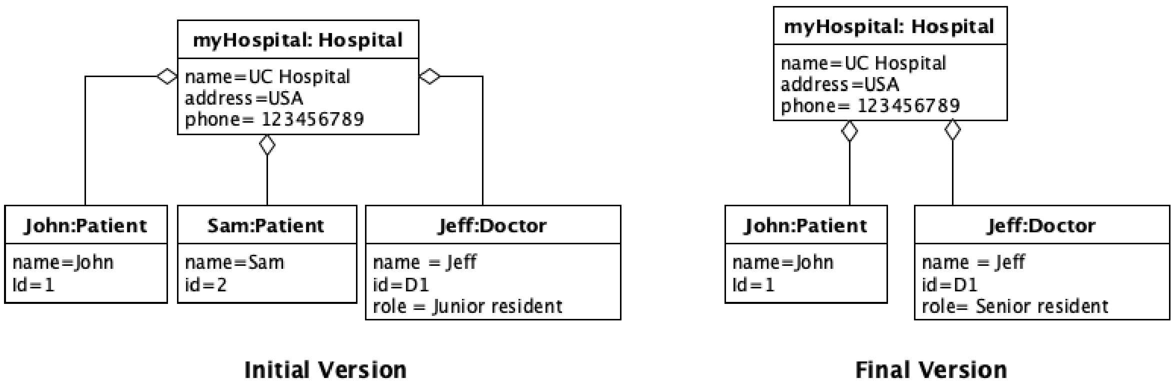

Fig. 7: A Sample Model of an Hospital

and Section 4.2 are automated, and the complete code is publicly available in a Github repository[2].

5 Classification of the Extracted Changes

This phase of our work describes how we classify the extracted change sets discussed in the previous section. To classify a changeset, we used a set of classifiers from the Waikato Environment for Knowledge Analysis (WEKA) toolset [17]. The WEKA tool set is open-source software that provides access to state-of-the-art machine learning and classification algorithms. These algorithms can be accessed easily by users to execute many machine-learning tasks on user-defined data. The WEKA toolset was used to classify the changes recovered from the previous section into four categories of changes associated with software maintenance, as specified by the ISO/IEC 17464 standard [29]. The set of classifiers used for the classification process includes OneR [27], Random Forest [9], Multi-class Classifier [36], Classification via Regression [17], KStar [12], SMO [44], LibSVM [10], Bayes Net [42], Naive Bayes [33], and J48 [47]. The categories of changes considered in this work include the following:

1. Corrective: This is a change associated with fixing an issue that negatively affects the optimal functioning of the software. These changes are associated with correcting bugs or mistakes in the models.
2. Adaptive: This is a change associated with enhancing the functionalities of the software to satisfy new requirements or specifications. These changes are often associated with adding new features, satisfying new requirements or adapting to new external factors.
3. Perfective: This is a change associated with optimizing the functionalities or features of software. These changes include reorganizing the code base, improving the software documentation, and refactoring the models to improve maintainability.
4. Preventive: This is a change that is done in order to address a possible future need. These kinds of changes include testing components of the models.

[2]https://github.com/sopopoola/hawk

5.1 Dataset Collection and Analysis

The training data for the supervised learning algorithm was developed by manually annotating over 300 changes from 28 Simulink repositories and 60 changes from 10 LabVIEW repositories via the process described in Section 4.2. The manual annotation was independently done by the first two authors using the labels and descriptions adapted from Murgia et al. [39]. At the end of the classification process, 45.4% of the changes were classified as perfective, 39.3% were classified as adaptive, 12.3% were classified as corrective, and 3% were classified as preventive. Fig. 8 and Fig. 9 summarize the distribution of the change types across the 28 Simulink repositories and 10 LabVIEW repositories.

For each changeset, we extracted seven primitive metrics and matched them to the manually annotated label for that changeset. The seven metrics are the total number of elements added, total number of elements deleted, total number of elements with at least one attribute changed, total number of elements that have been reordered, total number of newly added models, total number of models deleted, and the total number of existing models that have at least one element or attribute changed. These metrics were chosen because they capture all the basic atomic (non-derived) set of metrics related to changes between two models. The class labels used for the classification are the types of changes manually annotated for each changeset and they are adaptive, predictive, corrective, and preventive. It is possible that a particular commit belongs to more than one classification. Therefore, we adopt a binary classification where each commit is either classified as one of the class labels or not. This means that we have four identical files for each type: the first file classifies the changesets into adaptive or non-adaptive, and the second file divides the changesets into corrective or non-corrective. The third and fourth files also follow a similar pattern for the perfective and preventive categories.

In summary, we have four identical files for each of the change categories with over 400 changesets. This also means that the classification process was executed four times with each of the classifiers mentioned previously. Finally, each entry to the dataset used for the classification process in WEKA contains seven numeric attributes, representing each of the seven extracted metrics and a class label of either positive or negative for the change type that is under consideration.

5.2 Methodology for Validating the Classifiers

The ten-fold cross-validation technique has been used to assess the performance of the classification algorithm used in our work. A ten-fold cross-validation technique is a popular mechanism for assessing the performance of machine learning models and algorithms [2]. In a 10-fold cross-validation process, the dataset is divided into 10 equal parts (called folds) and ten rounds of validation are conducted. For each validation round, one part or fold of the data set is used as test data and the remaining 9 folds are used as a training set. For the next round of validation, a new fold is used as test data and the other 9 folds (including the one used as test data in the prior round) are used as the training set. The procedure is repeated ten times, with each fold used as test data once.

Every changeset in the dataset has been assigned exactly one label, whose value is either a positive or negative for the change type under consideration. The dataset is divided into 10 parts as per the requirement of the ten-fold cross validation technique. The training data that consists of 9 parts of the data set is used to train the classification model to identify the best set of patterns and anti-patterns of changes that best match a given classification label. The testing fold is used to validate the classification model by initially excluding the manually annotated label from the training data, then allowing the classification model to predict the appropriate change classification, and comparing the results with the initial label. In this way, the validation process mimics realistic scenarios where the training set corresponds to available data that have been manually classified by humans, and the test data correspond to the set of changes that need to be classified by our classification algorithm.

The F-1 score is a standard statistical value to measure of the accuracy of a classification model. The F-1 score is the harmonic mean of a model's precision and recall. The highest possible F-1 score is one and this score indicates a model that is 100% accurate, while the worst score is zero. However, most real-world classification models have a score

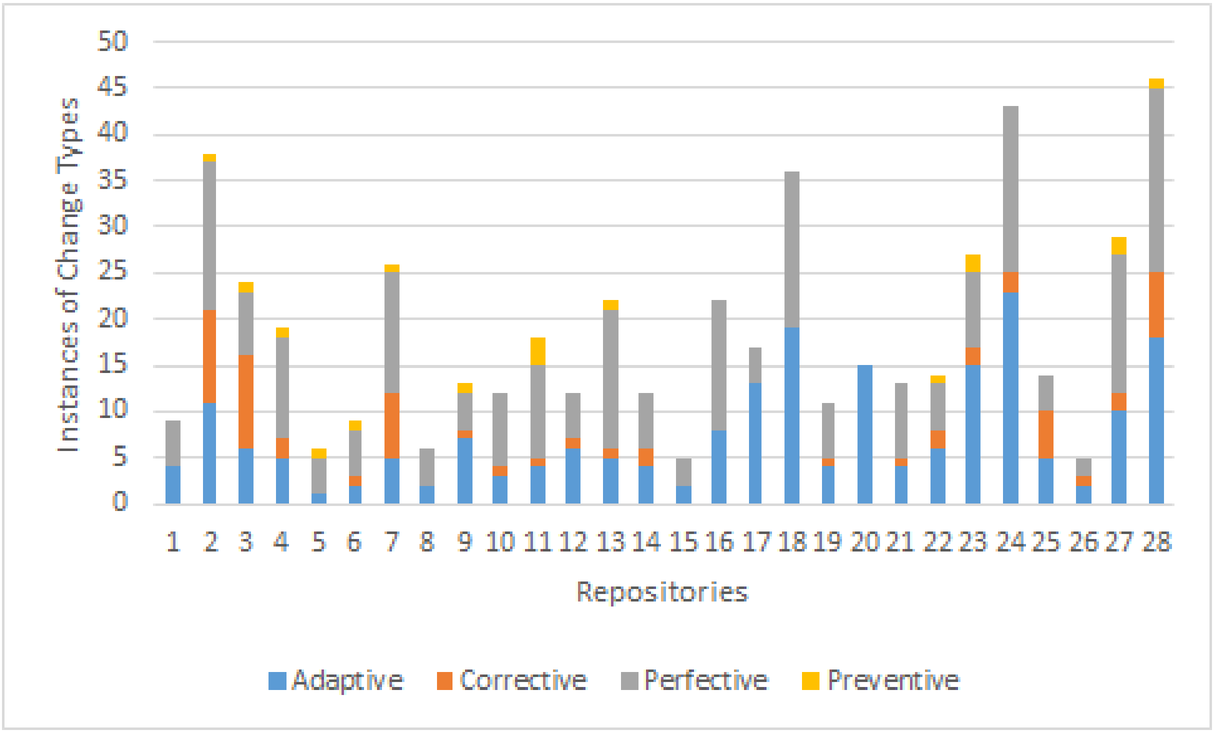

Fig. 8: Distribution of Maintenance Types across Simulink Repositories

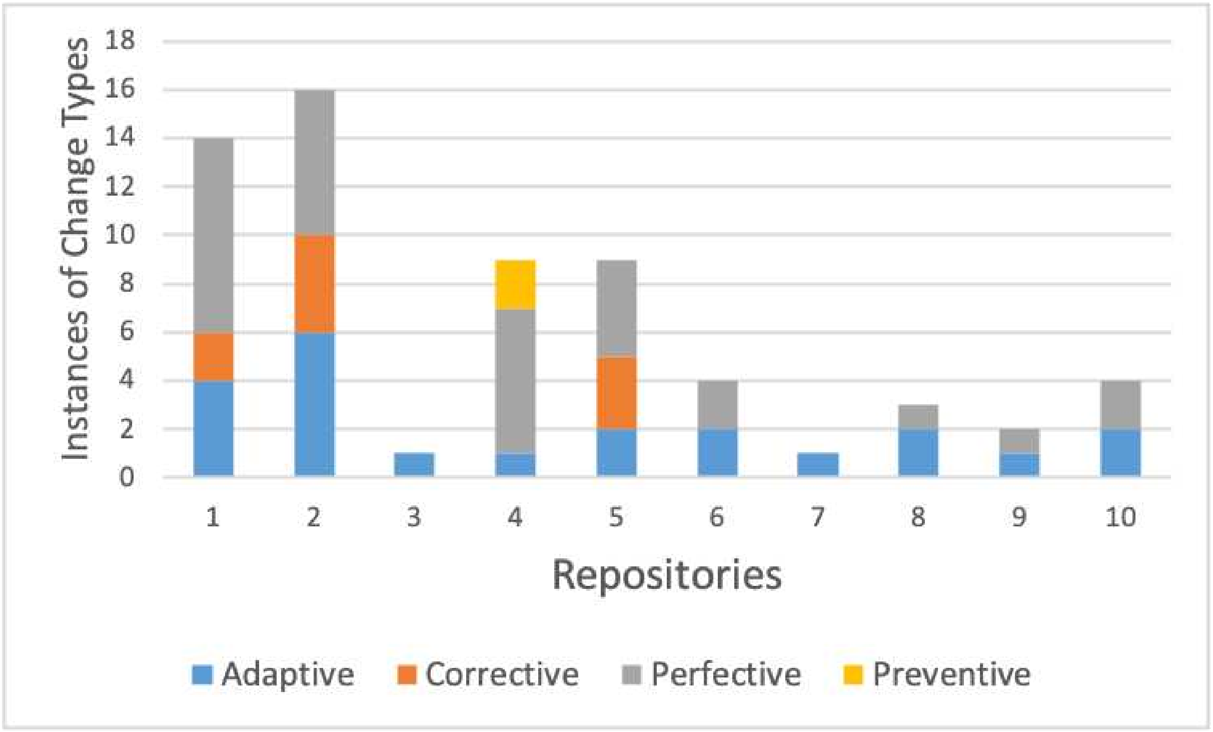

Fig. 9: Distribution of Maintenance Types across LabVIEW Repositories

greater than 0 but less than 1. The F-1 score measure has been used to assess the performance of each of the 10 classifiers used in this paper.

5.3 Analysis of the classification results

The results of the classification process shows that some of the classifiers perform better in one category while others excel in other categories. No classifier actually performed better than other classifiers in all the change categories being considered. We analyzed the results of the classification process in three distinct categories - Simulink datasets, LabVIEW datasets, a combination of Simulink and LabVIEW datasets. During the analysis, we aimed to answer the first three research questions discussed in Section 3 and stated below.

1. **RQ1:** Which classifier(s) best predict the maintenance type for changes in Simulink models?
2. **RQ2:** Which classifier(s) best predict the maintenance type for changes in LabVIEW models?
3. **RQ3**: Can a model trained on Simulink models accurately predict the maintenance categories of changes in LabVIEW models (and vice-versa)?

RQ1. Performance of Classifiers on Simulink Models

To answer the first research question, we used the 10-fold cross-validation approach on change instances from Simulink models only. The average F-1 score for all the classifiers across the four categories was 0.79. Overall, the Random Forest classifier offers the best aggregate performance across the four categories with an average F1-score of 0.84 while the LibSVM has the worst performance with an average F-1 score of 0.71. Furthermore, the best performance from all the classifiers was in the preventive category with an average F1-score of 0.95 while the perfective category has the worst score 0.65. Fig. 10 summarizes the results of the classification process.

RQ2. Performance of Classifiers on LabVIEW Models

To answer the second research question, we used the 10-fold cross-validation approach on change instances from LabVIEW models only. The average F1-score for all the classifiers across the four categories was 0.62. Overall, the Bayes Net classifier offers the best aggregate performance across the four categories with an average F1-score of 0.68 while Regression has the worst performance with an average F-1 score of 0.45. Furthermore, the best performance from all the classifiers was in the preventive category with an average F1-score of 0.94 while the perfective category has the worst score 0.49. The performance in the maintenance categories is consistent with the results in RQ1. Fig. 11 summarizes the results of the classification process.

RQ3. Performance of Classifiers on Simulink-Trained LabVIEW Models

To answer the third research question, we trained the selected classifiers on Simulink models only, and then tested the trained models on LabVIEW models. The average F-1 score for all the classifiers across the four categories was 0.58. Overall, the Naive Bayes classifier offers the best aggregate performance across the four categories with an average F-1 score of 0.74 while the SMO has the worst performance with an average F-1 score of 0.52. Furthermore, the best performance from all the classifiers was in the preventive category with an average F-1 score of 0.94 while the perfective category has the worst score 0.55. The performance in the maintenance categories is consistent with the results in RQ1 and RQ2. Fig. 12 summarizes the results of the classification process.

In conclusion, the best performance was by classifiers trained and validated on Simulink models, while classifiers from both the LabVIEW models and the Simulink-trained LabVIEW models offer a mixed performance. This suggests that a large dataset is necessary for the training, while using a generic model (in this case, trained on Simulink but test on LabVIEW models) may not offer much improvement in the performance. Fig. 13 summarizes the result of the comparison across the three categories. The "LabVIEW-Simulink" tag indicates the Simulink-Trained LabVIEW

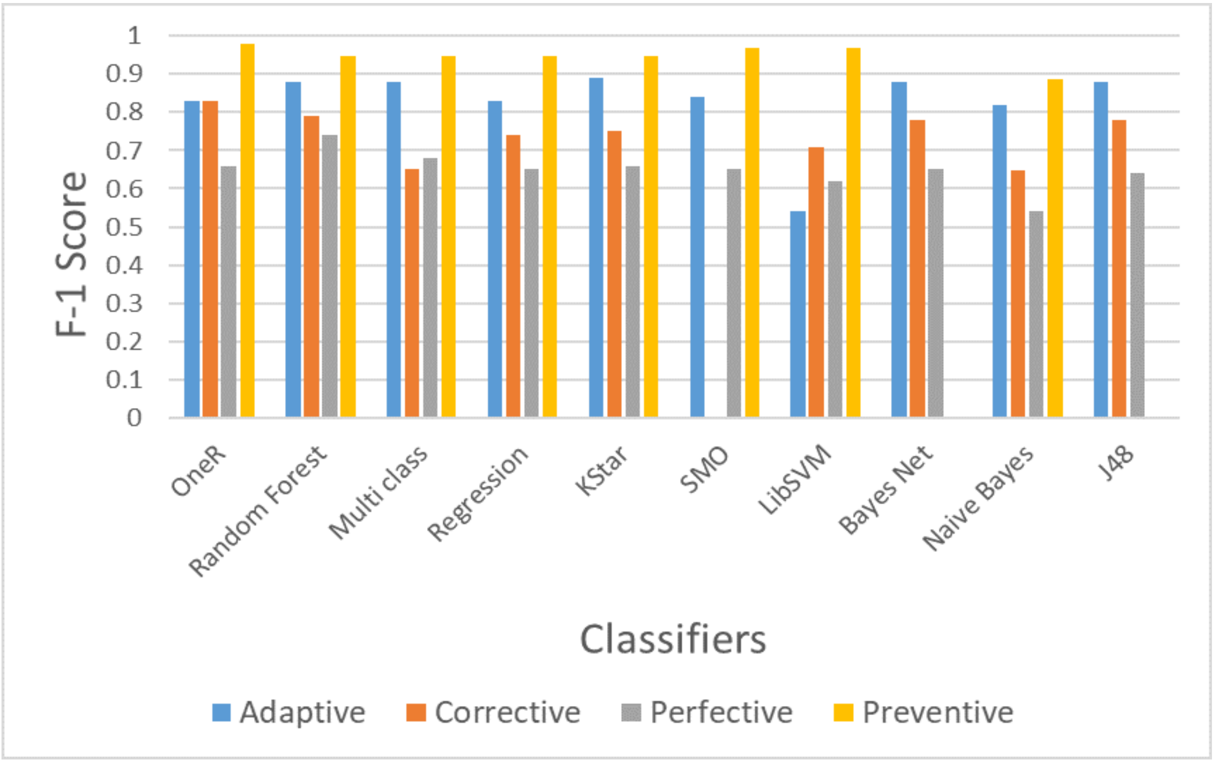

Fig. 10: Results of the Change Classification Process on Simulink Models

models. Some of the classifiers in Fig. 10, Fig. 11, and Fig. 12 do not have an F-1 score. This is because the number of true positives produced by the classifier is zero. Therefore, the calculated precision and recall will also be zero leading to an invalid F-1 score.

6 Evaluation Methodology and Results

This section describes the procedure we followed to evaluate the effectiveness of the change classification process. Specifically, we compared our results with the design decisions that have been extracted from the discussions in the issues reported in two large Simulink repositories. The logged issues that are reported in the issues section of the repository often contain some discussions on why a change is necessary, and the type of change required via labels. Therefore, a manual analysis of the discussion around an issue might be able to reveal the intended type of change behind a changeset. This makes the issues a helpful way to evaluate the change classification process.

We evaluated the accuracy of our framework on two repositories: NASA T-MATS[3] (a thermodynamic modeling package), and CJT[4] (a personal repository for modeling robotic joints). These repositories were chosen because they are open source, and they have a high number of logged issues and commits in GitHub. These repositories have also been excluded from the original 28 Simulink repositories used for the training set, in order to prevent bias when evaluating the generalizability of our approach. Table 4 gives an overview of the repositories, with number of issues, number of issues related to Simulink models, number of commits, and application domains.

Table 4 shows that only a small fraction of the total number of issues typically affects Simulink models. To evaluate the applicability of the classified changes, we compared our results with the issues in the repository within the same time period. For example, the first version of the T-MATS repository was on Jan 31, 2014 which our

[3]https://github.com/nasa/T-MATS
[4]https://github.com/geez0x1/CompliantJointToolbox

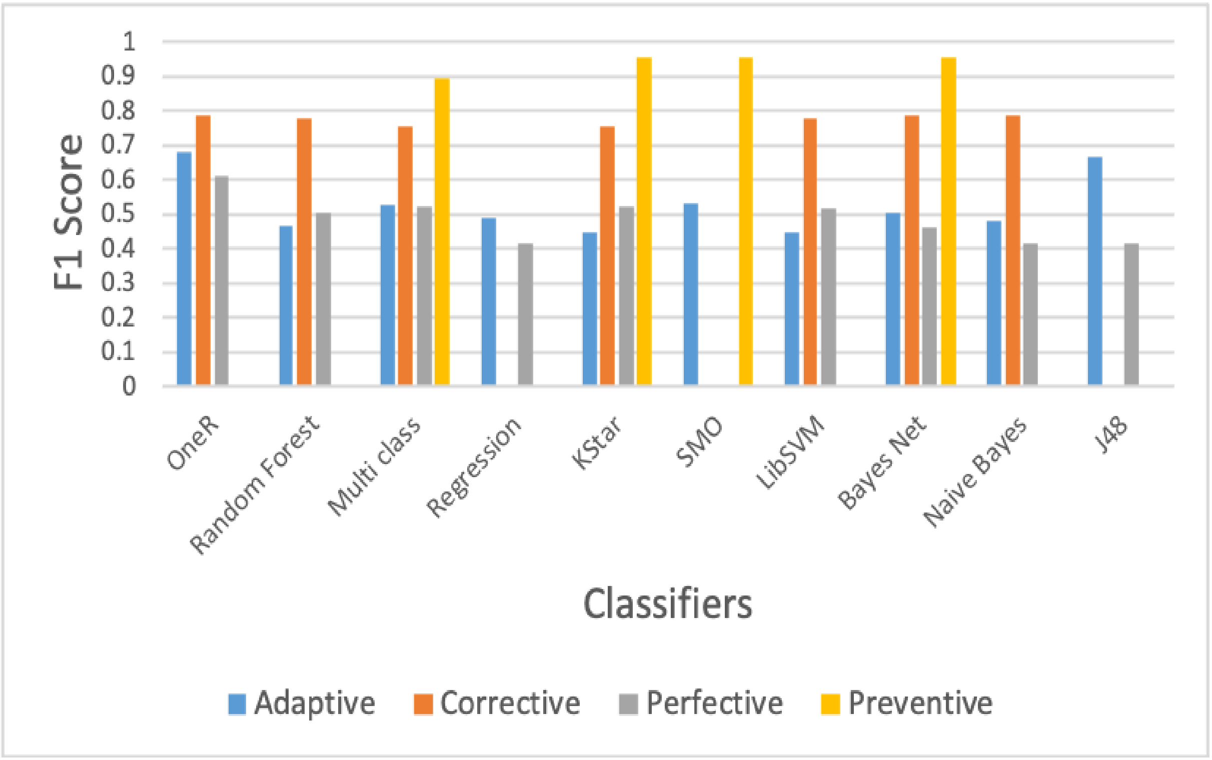

Fig. 11: Results of the Change Classification Process on LabVIEW Models

Properties \ Repositories	T-MATS	CJT
Number of issues	89	75
Number of resolved issues	86	58
Number of issues related to Simulink models	26	10
Number of commits	221	668
Domain	Thermodynamics	Robotics

Table 4: Overview of T-MATS and CJT Repositories

framework predicted to be a corrective type of change; on the same day, two issues were closed and they were labeled correction and bugs, respectively. In particular, we aim to answer research question 4 as stated below.

RQ4: *What is the performance of the best-performing classifier in terms of precision, recall, and F1-score, over the labels associated with some of the design decisions in the issues reported in the repositories?*

The Random Forest classifier offers the best aggregate performance for Simulink models from the list of classifiers discussed in Section 5. Therefore, we used the Random Forest classifier to evaluate the accuracy of predicting the type of change. To investigate the performance of the classifier, we extracted the seven attributes discussed in Section 4 for each of the commits in the two repositories used for the evaluation. Then, we considered the performance of our change classification algorithm with respect to the labels attached to some of the design decisions in the issue logs. All of the affected decisions have only four kinds of labels, and we matched these labels to appropriate change types. The four labels are the correction and bug labels that were matched to the corrective change type, the clean up label that was matched to the perfective change type and the enhancement label that can be either adaptive or perfective. It should be noted that none of the labels were associated with the preventive change type. Finally, we calculated the precision, recall and F1-score for each of the change types.

Figure 14 gives an overview of the correctness of the prediction classification with respect to the labels in the issue logs. The figure shows that

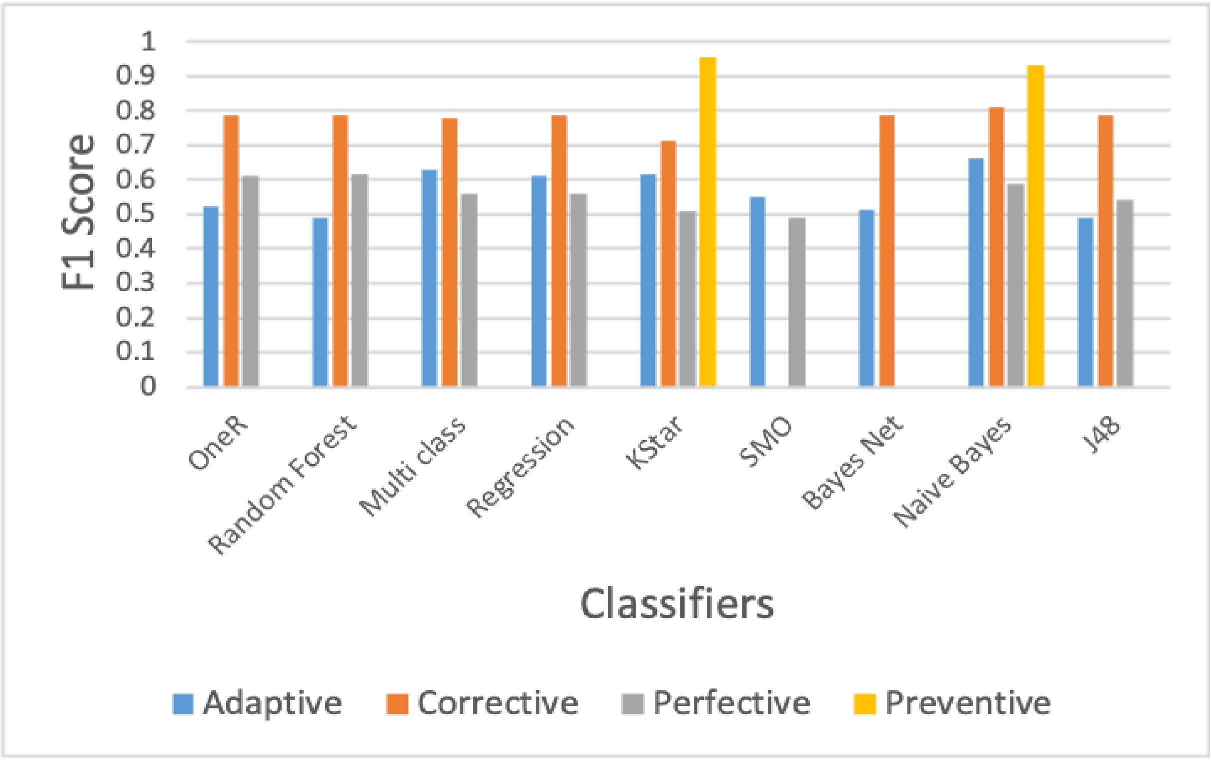

Fig. 12: Results of the Change Classification Process of Simulink-Trained LabVIEW Models

the classification algorithm performed very well for all the classes with an aggregate F-1 score of 0.83. The preventive class offers the highest performance of 0.95. This means that the Random Forest classifier could efficiently predict that none of the labels were associated with a preventive class. Furthermore, the corrective class offers the least performance of 0.74. A manual investigation reveals that the corrective cases that have been labeled incorrectly actually involved the addition of new features. However, the new features were added to fix a bug and not to satisfy new requirements. We could not repeat the same experiment for LabVIEW models due to a lack of issues in the LabVIEW repositories.

In general, an acceptable F-1 score for a machine learning model will depend on the context in which the model will be applied. For example, safety-critical systems will require very high level of accuracy compared to other domains. The Random Forest classifier used for this study produces an F-1 of more than 70% in its worst case scenario. This, in our opinion, indicates a high-level of confidence in its prediction.

7 Limitations and Threats to Validity

There are some limitations that may affect the generalization of the results presented in this paper. We analyse some of the threats to the validity of our results in the following paragraphs.

1. *Small dataset.* A small dataset can affect the generalizability of results because the dataset may not contain sufficient patterns for different scenaios. Unfortunately, the dataset used for this study involved only 28 Simulink repositories and 10 LabVIEW repositories. The major challenge in the data collection process was in finding models with rich version history, because the scope of this paper focuses on change extraction and analysis. While there are a lot of third-party models [11, 49], many of them do not capture a rich version history of the models. Additionally, we also considered extracting models from MatlabCentral, the official webpage from Mathworks for sharing Simulink files. However, the platform offers

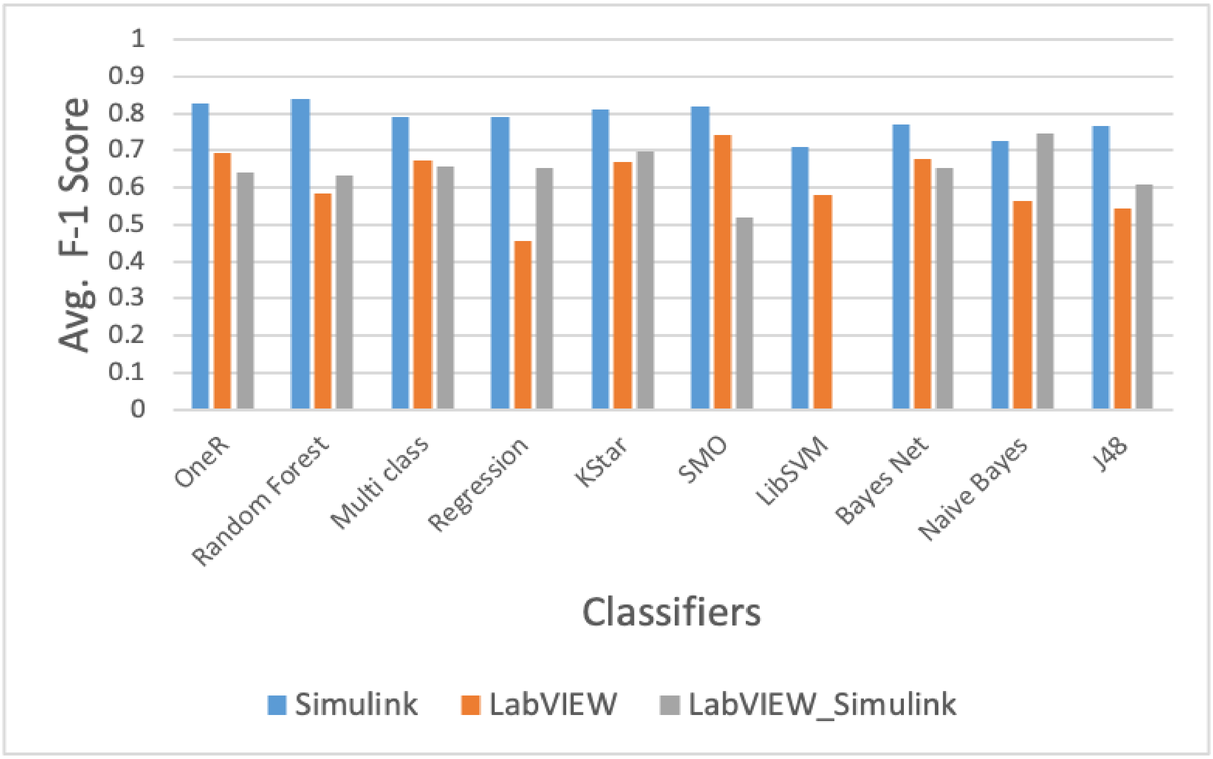

Fig. 13: Average F-1 Scores

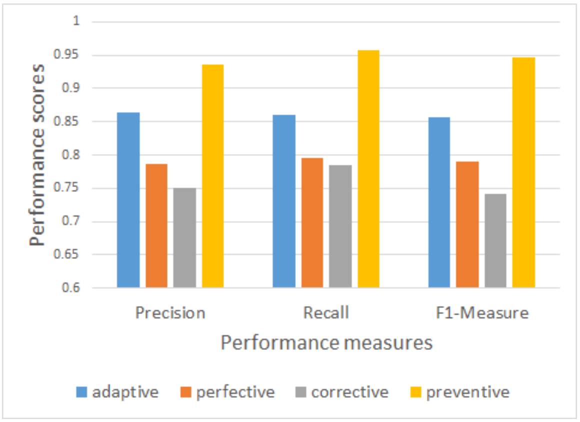

Fig. 14: Evaluation results for the Random Forest classifier

an RSS feed and the platform only offers project releases instead of commits. Due to this limitations, it was very hard to extract granular commit-level changes between successive versions of project releases. The changes between project releases are very large (as opposed to commit changes), and contain multiple categories of changes embedded in a new release. This makes it inefficient for the study and was excluded. In the future, we plan to work more on how to incorporate changes from project releases. Furthermore, LabVIEW was limited to only 10 repositories because the extended Hawk tool currently supports only the NXG version of LabVIEW. In the future, we plan to incorporate support for the traditional binary format of LabVIEW models. Finally, some studies have also shown that the difference in performance when a model is trained on small or large dataset may not be significant [5, 53]. Hence, we believe the results presented here is still very important, even though future work on larger dataset is necessary to validate the results.

2. *The datasets contain only LabVIEW and Simulink models.* The dataset used for training and evaluating the classifier contains only LabVIEW and Simulink models that have been extracted from open-source repositories. Hence,

it is possible that the results may not be generalizable to other types of models especially non-executable models.

3. *Simple LabVIEW datasets.* The LabVIEW dataset contains few models with limited changesets. This lack of sufficient data may affect the classification results.
4. *Simulink repositories do not only contain Simulink models.* Other file formats (especially MATLAB files) are often present in the same repository. For example, a bug in a Simulink model might be fixed with a code in a MATLAB file. Furthermore, we only consider the commits where changes are made to any Simulink model.
5. *This work assumes that models have the same names across versions.* Hence, if a model name was changed across two versions, the tool presented in this paper detects such scenarios as the addition of a new model in the succeeding version and the removal of another model in the previous version. Furthermore, our work primarily targets Simulink models, and the process of matching identical models uses the file attribute captured in the Simulink metamodel provided by the Massif tool. However, we believe that most aspects of our work should be generalizable to any EMF-based framework. We plan to address these limitations in the future by using additional VCS capabilities and metadata: Git can be asked to report renames, for instance.
6. *The functionality described in this work depends on the correctness of results produced by the three main tools*: Hawk, Massif, and EMF Compare. Although attempts have been made to verify the correctness of the results, we cannot guarantee their absolute correctness.

Despite the above limitations, we believe that our work lays a good foundation for classifying changes associated with the evolution history of models in repositories. Our future lines of work include the lifting of several of these limitations. Section 9 provides more details on the proposed future work.

8 Related Work

Some work has also been done in classifying changes to software. Ferzund et al. [15], Herzig et al. [25], Kim et al. [32], and Xia et al. [55] introduced various techniques and algorithm to classify software changes as bug-free or buggy code. Ferzund et al. [15] presented a metric-based approach to classify changes while Herzig et al. [25] classified changesets using a graph of the structural dependencies among the changesets. Kim et al. [32] adopted a machine learning approach that uses data from the version history of a previous project, and Xia et al. [55] used genetic algorithms to identify defective changes. Pandey et al. [41] conducted an empirical study where they used various machine-learning algorithms to classify issue logs into bugs or non-buggy categories. Their results show that the Random Forest classifier offers the best performance, similar to the results reported in this paper. These research works mainly focus on the corrective maintenance activities and did not consider other types of maintenance activities.

Hassan [24] developed an algorithm to classify commit messages into their respective maintenance categories. Mockus et al. [37] also classified changes to their appropriate maintenance types via the manual classification of a change's commit message. Li et al. [34] conducted an empirical study to identify characteristics of influential changes and then developed a classifier to predict changes that are influential based on the identified characteristics. The evaluation results show that the classifier is able to achieve an F-1 score of 80%. Hindle et al. [26] developed a classifier to categorize changes in a large commit using only the commit metadata such as commit messages, commit authors and modified files. Their study reported that data related to the commit message and commit author is sufficient for classifying changes in a commit into their respective categories. Gharbi et al. [20] and Yan et al. [56] also uses active-learning and topic-modeling techniques respectively to classify changes belonging to multiple maintenance categories via its commit message. The research discussed above have targeted text-based programming languages. Furthermore, most of these research works have focused on classifying changes to their respective maintenance types via its commit message. Unfortunately, commit messages are subjective and may contain trivial or irrelevant information.

Several works on change classification have also been published in the MDE community.

Goknil et al. [21] proposed an approach for classifying changes to requirement models into the structural type of change such as addition or deletion of requirements. This classification have been used to manage the impacts of changes in software requirements. Hamlaoui et al. [13] proposed a change classification approach based on whether the elements were added, deleted, or modified. The classification process was shown to help ensure global consistency of partial models as the models evolve. Gruschko et al. [22] and Wachsmuth [52] also classified changes to metamodels based on the effect of these changes on the conformance relationship between the metamodel and its conforming models. The approaches discussed so far have focused on change classification based on the structure of the change and do not consider the maintenance categories. Furthermore, these research contributions are mainly focused on preserving the consistency of models. The research presented in this paper extends the literature by proposing a novel approach for automatically classifying changes to models into maintenance categories via the changeset metadata.

9 Conclusion and Future Work

This paper describes an approach to classify changes to model repositories into maintenance categories via a set of change metrics. The change metrics include changes to models, model elements, and model element attributes. These change metrics have been extracted from the version history of 451 Simulink models in 28 open-source repositories and 76 LabVIEW models in 10 repositories. To facilitate the extraction of the change metrics from open-source repositories, we extended the Hawk framework to support the indexing of LabVIEW and Simulink models. We mined over 300 versions of Simulink models and 60 versions of LabVIEW models.

Ten classifiers have been trained on the extracted changes to predict the type of change embedded in changesets between two version histories. Four types of changes were identified, based on the type of maintenance activity. The Random Forest classifier and the Bayes Net classifier were shown to offer the best aggregate performance among the ten classifiers.

The performance of the Random Forest classifier was evaluated by comparing its results with the labels associated with discussions that were manually extracted from the issues logs within a similar timeframe. These labels have been manually classified into an appropriate maintenance type and compared with the predictions of the Random Forest classifier. The Random Forest classifier achieved an aggregate F-1 score of 0.83.

In the future, we plan to extend this work in four major areas. Firstly, we plan to improve the performance of the classifier by developing new approaches that can also use the change history (in addition to the change operation counts used in this paper) to predict the correct type of change. We can draw insights from existing works on source code change classification such as those that uses structural dependencies among the changesets [25], refactoring changes and granular change types [19] from existing tools like RefactoringMiner [51] and ChangeDistiller [16], information extracted from commits metadata[26], and the abstract syntax tree of changed codes [35]. Secondly, we plan to expand the datasets to more model formats in order to generalize the results across different types of models. Thirdly, this paper relies primarily on file names to identify models across successive versions. In the future, we plan to utilize existing works on model clone detection [3] to support change identification and classification tasks. Finally, we plan to conduct empirical studies to understand the relationships between the different types of change and other phenomena such as bad smells and refactoring tasks.

10 Declarations

10.1 Ethical Approval

Not applicable.

10.2 Competing Interests

The authors have no known competing interests that may affect the results presented in this paper.

10.3 Authors' Contributions

Saheed Popoola wrote the main manuscript. Saheed Popoola and Xin Zhao did the manual classification of commits into appropriate maintenance types. Antonio Garcia-Dominguez provided mentorship and helped with Hawk integration. Jeff Gray provided mentorship and supervision. All authors reviewed the manuscript.

10.4 Funding

Not applicable

10.5 Availability of Data and Materials

The data used in this paper can be publicly accessed here - https://github.com/sopopoola/hawk and https://bit.ly/ChangeAnalysisSAM

References

[1] Companies using LabVIEW. https://enlyft.com/tech/products/labview.

[2] MATLAB Simulink. https://www.mathworks.com/products/simulink.html.

[3] Manar H Alalfi, James R Cordy, Thomas R Dean, Matthew Stephan, and Andrew Stevenson. Models are code too: Near-miss clone detection for simulink models. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 295–304. IEEE, 2012.

[4] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. What's a typical commit? a characterization of open source software repositories. In *16th IEEE international conference on program comprehension*, pages 182–191, 2008.

[5] Alexandre Bailly, Corentin Blanc, Élie Francis, Thierry Guillotin, Fadi Jamal, Béchara Wakim, and Pascal Roy. Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. *Computer Methods and Programs in Biomedicine*, 213:106504, 2022.

[6] Konstantinos Barmpis, Antonio García-Domínguez, Alessandra Bagnato, and Antonin Abherve. Monitoring model analytics over large repositories with Hawk and MEASURE. In *Model Management and Analytics for Large Scale Systems*, pages 87–123. 2020.

[7] Konstantinos Barmpis and Dimitris Kolovos. Hawk: Towards a scalable model indexing architecture. In *1st Workshop on Scalability in Model Driven Engineering*, pages 1–9, 2013.

[8] Christian Bartelt. Consistence preserving model merge in collaborative development processes. In *International Workshop on Comparison and Versioning of Software Models*, pages 13–18, 2008.

[9] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

[11] Shafiul Azam Chowdhury, Lina Sera Varghese, Soumik Mohian, Taylor T Johnson, and Christoph Csallner. A curated corpus of simulink models for model-based empirical studies. In *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 45–48, 2018.

[12] John G Cleary and Leonard E Trigg. K*: An instance-based learner using an entropic distance measure. In *Machine Learning*, pages 108–114. 1995.

[13] Mahmoud El Hamlaoui, Saloua Bennani, Mahmoud Nassar, Sophie Ebersold, and Bernard Coulette. A MDE approach for heterogeneous models consistency. In *13th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 180–191, 2018.

[14] Jeannie Falcon. Facilitating modeling and simulation of complex systems through interoperable software. In *Keynote address at ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, 2017.

[15] Javed Ferzund, Syed Nadeem Ahsan, and Franz Wotawa. Software change classification using hunk metrics. In *IEEE International Conference on Software Maintenance*, pages 471–474, 2009.

[16] Beat Fluri, Michael Wursch, Martin PInzger, and Harald Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on software*

engineering, 33(11):725–743, 2007.

[17] Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H Witten, and Len Trigg. Weka-a machine learning workbench for data mining. In *Data Mining and Knowledge Discovery Handbook*, pages 1269–1277. 2009.

[18] Antonio Garcia-Dominguez, Nelly Bencomo, Juan Marcelo Parra-Ullauri, and Luis Hernán García-Paucar. Querying and annotating model histories with time-aware patterns. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 194–204, 2019.

[19] Lobna Ghadhab, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Montassar Ben Messaoud. Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. *Information and Software Technology*, 135:106566, 2021.

[20] Sirine Gharbi, Mohamed Wiem Mkaouer, Ilyes Jenhani, and Montassar Ben Messaoud. On the classification of software change messages using multi-label active learning. In *34th ACM/SIGAPP Symposium on Applied Computing*, pages 1760–1767, 2019.

[21] Arda Goknil, Ivan Kurtev, Klaas Van Den Berg, and Wietze Spijkerman. Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, 56(8):950–972, 2014.

[22] Boris Gruschko, Dimitrios Kolovos, and Richard Paige. Towards synchronizing models with evolving metamodels. In *International Workshop on Model-Driven Software Evolution*, page 3, 2007.

[23] Thomas Hartmann, Francois Fouquet, Matthieu Jimenez, Romain Rouvoy, and Yves Le Traon. Analyzing complex data in motion at scale with temporal graphs. In *29th International Conference on Software Engineering and Knowledge Engineering*, pages 596–601, 2017.

[24] Ahmed E Hassan. Automated classification of change messages in open source projects. In *23rd ACM symposium on Applied computing*, pages 837–841, 2008.

[25] Kim Herzig, Sascha Just, Andreas Rau, and Andreas Zeller. Classifying code changes and predicting defects using changegenealogies. *Saarland University*, 2013.

[26] Abram Hindle, Daniel M German, Michael W Godfrey, and Richard C Holt. Automatic classication of large changes into maintenance categories. In *17th IEEE International Conference on Program Comprehension*, pages 30–39, 2009.

[27] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[28] A Horváth, István Ráth, and Rodrigo Rizzi Starr. Massif-the love child of Matlab Simulink and Eclipse. *EclipseCon NA*, 2015.

[29] ISO/IEC. International standard-iso/iec 14764:2006; software engineering- software lifecycle processes and maintenance. *International Standard Organization*, pages 1–46, 2006.

[30] Gary W Johnson, Richard Jennings, and Richard Jennigns. *LabVIEW graphical programming*, volume 580. McGraw-Hill New York, 2006.

[31] Arvinder Kaur and Deepti Chopra. Gccgit change classifier for extraction and classification of changes in software systems. In *Intelligent Communication and Computational Technologies*, pages 259–267. 2018.

[32] Sunghun Kim, E James Whitehead, and Yi Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196, 2008.

[33] Pat Langley, Wayne Iba, Kevin Thompson, et al. An analysis of bayesian classifiers. In *10th National Conference on Artificial Intelligence*, volume 90, pages 223–228, 1992.

[34] Daoyuan Li, Li Li, Dongsun Kim, Tegawendé F Bissyandé, David Lo, and Yves Le Traon. Watch out for this commit! a study of influential software changes. *Journal of Software: Evolution and Process*, 31(12):e2181, 2019.

[35] Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. Atom: Commit message generation based on abstract syntax tree and hybrid ranking. *IEEE Transactions on Software Engineering*, 48(5):1800–1817, 2020.

[36] Yiguang Liu, Zhisheng You, and Liping Cao. A novel and quick SVM-based multi-class

classifier. *Pattern Recognition*, 39(11):2258–2264, 2006.

[37] Audris Mockus and Lawrence G Votta. Identifying reasons for software changes using historic databases. In *International Conference on Software Maintenance*, pages 120–130, 2000.

[38] Audris Mockus and David M Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[39] Alessandro Murgia, Giulio Concas, Roberto Tonelli, Marco Ortu, Serge Demeyer, and Michele Marchesi. On the influence of maintenance activity types on the issue resolution time. In *10th international conference on predictive models in software engineering*, pages 12–21, 2014.

[40] Richard F Paige, Dimitrios S Kolovos, Louis M Rose, Nicholas Drivalos, and Fiona AC Polack. The design of a conceptual framework and technical infrastructure for model management language engineering. In *14th IEEE International Conference on Engineering of Complex Computer Systems*, pages 162–171, 2009.

[41] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, 13(4):279–297, 2017.

[42] Judea Pearl. Bayesian networks. *The Handbook of Brain Theory and Neural Networks*, pages 149–153, 1998.

[43] Ralph Peters and Andy Zaidman. Evaluating the lifespan of code smells using software repository mining. In *16th European Conference on Software Maintenance and Reengineering*, pages 411–416, 2012.

[44] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. advances in kernel methods-support vector learning. *MIT Press*, pages 185–208, 1999.

[45] Saheed Popoola and Jeff Gray. A labview metamodel for automated analysis. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1127–1132. IEEE, 2019.

[46] Saheed Popoola, Xin Zhao, Jeff Gray, and Antonio Garcia-Dominguez. Classifying changes to models via changeset metrics. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 276–285, 2022.

[47] J Ross Quinlan. *C4. 5: Programs for Machine Learning*. 2014.

[48] Per Rovegård, Lefteris Angelis, and Claes Wohlin. An empirical study on views of importance of change impact analysis issues. *IEEE Transactions on Software Engineering*, 34(4):516–530, 2008.

[49] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. Slnet: a redistributable corpus of 3rd-party simulink models. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 237–241, 2022.

[50] Antoine Toulmé and I Inc. Presentation of EMF Compare utility. In *Eclipse Modeling Symposium*, volume 1, pages 1–8, 2006.

[51] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. Refactoringminer 2.0. *IEEE Transactions on Software Engineering*, 48(3):930–950, 2020.

[52] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In *European Conference on Object-Oriented Programming*, pages 600–624, 2007.

[53] Pin Wang, En Fan, and Peng Wang. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recognition Letters*, 141:61–67, 2021.

[54] Zhihua Wen and Vassilios Tzerpos. An effectiveness measure for software clustering algorithms. In *12th IEEE International Workshop on Program Comprehension*, pages 194–203, 2004.

[55] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. Collective personalized change classification with multiobjective search. *IEEE Transactions on Reliability*, 65(4):1810–1829, 2016.

[56] Meng Yan, Ying Fu, Xiaohong Zhang, Dan Yang, Ling Xu, and Jeffrey D Kymer. Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project. *Journal of Systems and Software*, 113:296–308, 2016.