Received: 27 July 2021 | Revised: 30 May 2022 | Accepted: 9 July 2022

DOI: 10.1002/spe.3133

RESEARCH ARTICLE

WILEY

Evaluating the benefits of empowering model-driven development with a machine learning classifier

Ana C. Marcén[1] | **Francisca Pérez**[1] | **Óscar Pastor**[2] | **Carlos Cetina**[1]

[1]SVIT Research Group, Universidad San Jorge, Zaragoza, Spain

[2]Research Center on Software Production Methods (PROS), Universitat Politècnica de València, Valencia, Spain

Correspondence
Ana C. Marcén, SVIT Research Group, Universidad San Jorge, Zaragoza, Spain.
Email: acmarcen@usj.es

Funding information
Generalitat Valenciana, Grant/Award Numbers: ACIF/2018/171, PROMETEO/2018/176; Ministerio de Ciencia, Innovación y Universidades, Grant/Award Number: PID2021-128695OB-I00; Ministerio de Economía y Competitividad, Grant/Award Number: TIN2016-80811-P

Abstract

Increasingly, the model driven engineering (MDE) community is paying more attention to the techniques offered by the machine learning (ML) community. This has led to the application of ML techniques to MDE related tasks in hope of increasing the current benefits of MDE. Nevertheless, there is a lack of empirical experiments that evaluate the benefits that ML brings to MDE. In this work, we evaluate the benefits of empowering model engineers of model-driven development (MDD) with an ML classifier. To do this, we tackled how to embed the ML classifier as part of the MDD. Then, this was evaluated using two different real industrial cases. Our results show that despite the ML part takes an extra effort, the use of the ML classifier pays off in terms of the quality results, the perceived usefulness, and intention to use.

KEYWORDS

data-oriented software systems, machine learning, model-driven development

1 | INTRODUCTION

Every day humans generate several petabytes of data from a variety of sources such as orbital weather satellites, ground-based sensor networks, mobile computing devices, digital cameras, and retail point-of-sale registers.[1] The availability of huge amounts of data is changing our attitude toward science, which is moving from specialized to massive experiments and from focused to very broad research questions.[2] Likewise, software development is also changing from traditional computing to data-oriented systems.[3] Data-oriented software systems focus on analyzing data to offer valuable insights, which in turn may offer a quantifiable competitive advantage to the enterprise or organization.[4] With the emergence of this data-oriented perspective, the application of ML techniques has gained interest in software development.

On the one hand, there are development methodologies to design and implement data-oriented software systems based on ML techniques. These methodologies take advantage of ML techniques to use huge amounts of data that had previously no use and makes analysis and predicting trend and patterns.[5] The two major methodologies are CRISP-DM (Cross Industry Standard Process for Data Mining) and SEMMA (Sample, Explore, Modify, Model, and Assess).[4]

AMDD, Agile Model Driven Development; CRISP-DM, Cross Industry Standard Process for Data Mining; MDD, model-driven development; MDE, model driven engineering; ML, machine learning; SEMMA, Sample, Explore, Modify, Model, and Assess; TDD, test-driven development.

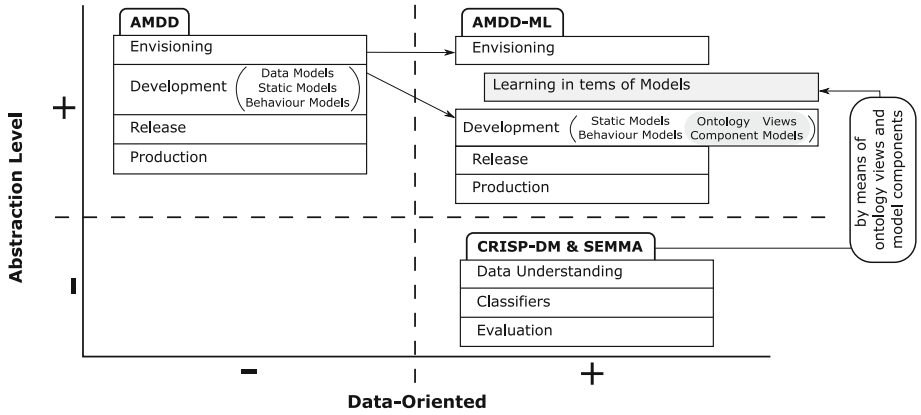

FIGURE 1 Overview of the comparison between methodologies according to human understanding and orientation to data

On the other hand, there are development methodologies to design and implement software systems based on models. These methodologies take advantage of models, which raise the abstraction level using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain.[6] Agile Model Driven Development methodology (AMDD) is an agile version of model-driven development (MDD) that takes advantage of both approaches: models and agility.

Increasingly, the model driven engineering (MDE) community is paying more attention to the techniques offered by the machine learning (ML) community.[7,8] Therefore, the software development may benefit from both the abstraction level (models) and data-oriented perspective (ML techniques). Nevertheless, there is a lack of empirical experiments that evaluate the benefits that ML brings to MDE. To cope with this lack, we evaluate the benefits of empowering model engineers of MDD with an ML classifier. To do this, the contribution of this article is twofold.

First, we tackle the ML classifier as part of the MDD. Specifically, we extend the AMDD methodology embedding the data-oriented perspective of CRISP-DM and SEMMA methodologies. Figure 1 shows an overview of how AMDD is extended to AMDD-ML. ML is embedded in AMDD-ML through the learning phase and the development phase. The learning phase is responsible for analyzing the data and creating ML classifiers for the company's purposes (e.g., an ML classifier to predict the success of its next advertising campaign). The development phase is responsible not only for the behavior and static models but also for the ontology views and component models. The ontology views and component models allow embedding the ML classifier as part of the models, providing a higher level of abstraction.

Figure 1 also shows the phases of each methodology and compares the methodologies in terms of abstraction level and oriented-data perspective. Regarding the abstraction level, as an assembly language is usually easier to understand than a binary code and a programming language is usually easier to understand than an assembly language, the high-level abstraction that is provided by models is usually easier to understand than programming languages. In addition, the models are much less bound to the underlying implementation technology, much closer to the problem domain, and allow engineers to see "the big picture."[6,9] Both AMDD and AMDD-ML are based on models, so they benefit from a higher abstraction level than SEMMA and CRISP-DM which are based on programming languages.

Regarding the data-oriented perspective, data can be modeled to be used or can be analyzed to provide valuable insights not only for enterprise competitive advantages but also for the development phase. For example, a marketing company can store information of the last advertising campaigns and show their statistical results of them. However, the company can go a step further and analyze these results creating an ML classifier that predicts the success of its next campaign. AMDD tackles the first step of this company, data modeling. In contrast, SEMMA, CRISP-DM, and AMDD-ML go a step further and benefit from analyzing and predicting patterns to exploit the data through ML techniques.

Second, we evaluate the benefits of embedding the ML in the MDD from the engineer's point of view. It was evaluated by two different real industrial cases. The first real case was CAF,* a worldwide provider of railway solutions. The second one was BSH,† the leading manufacturer of home appliances in Europe. A domain expert was responsible for identifying the requirements based on these real cases. Then, some students, who have previous knowledge of

*www.caf.net/en

†www.bsh-group.com/

working with models, were recruited to model these requirements. Specifically, the students were divided into two groups, to perform a paired design blocked by experimental objects,[10] where the experimental objects were the extended methodology (AMDD-ML) and the baseline methodology (AMDD). Finally, we evaluated the developed models concerning quality, effort, productivity, and satisfaction.

The outcome shows that there are no significant differences between the extended methodology and the baseline methodology concerning productivity. However, the extended methodology provided the best quality results, successfully passing 85.68% of the test cases. Moreover, this extended methodology also achieved the best satisfaction results for the perceived usefulness and the intention to use, overcoming the results of the baseline methodology for 8 marks and 15.34 marks, respectively.

The remainder of this article is structured as follows: Section 2 presents the AMDD-ML methodology. Section 3 details the evaluation designed to tackle the research questions, and the results of said evaluation. Section 4 discusses our methodology and the obtained results. Section 5 describes the threats to the validity of our work. Section 6 introduces the existing works that are related to the one presented through the following pages. Finally, Section 7 concludes the article.

2 | AMDD-ML METHODOLOGY

Figure 2 shows an overview of the phases of AMDD-ML and the workflow diagrams of their phases. On the one hand, AMDD-ML reuses four phases of AMDD: envisioning, development, release, and production. These four phases are briefly presented to make the article self-contained, there are more details in References 11 and 12. On the other hand, AMDD-ML contains a new phase: the learning in terms of models phase. This phase is responsible for data-oriented development through data models and ontologies. In Figure 2, the background of the learning in terms of models phase is dotted to highlight that is the new phase embedded in AMDD.

2.1 | Envisioning phase

This phase identifies the scope of the project through two activities: initial requirements envisioning and initial architectural envisioning. On the one hand, the initial requirements envisioning identifies the business goals, develops a common vision, and identifies the high-level requirements for the project. On the other hand, the initial architectural envisioning performs some high-level architectural modeling early in the project to help foster agreement regarding your technical strategy within the team and with critical stakeholders.

2.2 | Learning in terms of models phase

The most fundamental challenge for data-oriented software systems is to explore large volumes of data and extract useful information or knowledge for future actions.[13] Specifically, a classifier is the artifact trained from the stored data and used

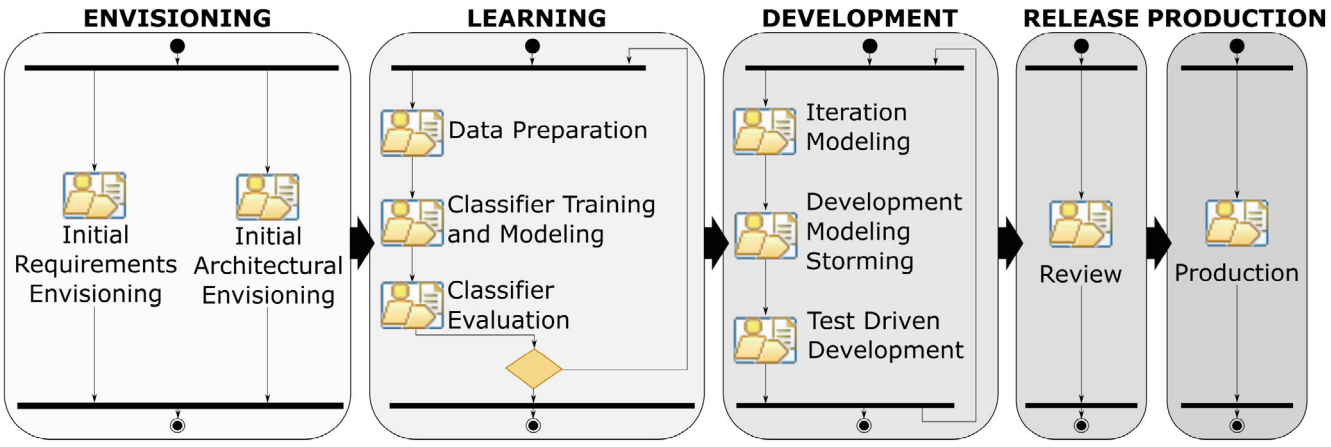

FIGURE 2 Overview of the phases and workflow diagrams of these phases

to predict, classify, or evaluate new data. Therefore, a classifier can allow modelers to know how effectively new data can be predicted, classified, or evaluated, so modelers can leverage this knowledge in their design. For example, in a railway company, if a modeler knows that a classifier can successfully determine whether a braking is correct, he may decide to communicate this information to the driver. This information will be more useful than a large list of sensor values, that the driver must analyze in real-time using his experience. However, to make this decision, the modeler needs to know that it is possible to train a successful classifier from the stored data.

For this reason, before starting the development phase, our proposed learning in terms of models phase allows to analyze the data to know if the data may be useful, what data is relevant, and how to predict, classify, or evaluate this data. To do this, the learning in terms of models phase is based on ML, so the engineers that work in this phase should have experience in ML.

ML is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically.[14] ML has a wide range of applications, including search engines, medical diagnosis, text and handwriting recognition, image screening, load forecasting, marketing, sales diagnosis, and so forth.[15] ML is based on training to learn how to extract information, and then to apply what it has learned to extract information from huge volumes of data. To do this, first, data has to be prepared to be used by an ML technique.

Second, a model is trained through an ML technique. However, the concept of model could be confused in this specific context because it has two meanings. On the one hand, in MDD, a model is mainly a representation of the essential aspects of the underlying subject.[16] On the other hand, in ML, a model is a rule-set, that is learned when comparing automatically feature vectors through an ML technique.[17] To avoid misunderstandings, in this work the concept of model in ML is replaced by the term "classifier." Therefore, second, a classifier is trained through an ML technique. Third, the classifier is validated to know its effectiveness.

The next subsections describe the step for this phase and our proposal to perform these steps in terms of models. Moreover, Figure 3 shows an example to ease the understanding of both the learning and development phases. This running example is based on one of the real cases used in our evaluation, where a railway company (CAF) needs to develop a software system for the maintenance of the different systems of a train, such as the braking system or the doors opening and closing system. To provide a simple and clear example, Figure 3 prioritizes the understanding over the completeness. Therefore, this figure only shows a part of the real artifacts as an example.

2.2.1 | Data preparation

The main target of this step consists in identifying what data are relevant for a specific purpose and encoding these data following the input format of the ML technique which is going to be used. Most of the ML techniques are designed to process feature vectors as inputs.[18] Feature vectors are known as the ordered enumeration of features that characterize the object being observed.[19] Therefore, if we plan to encode data as a feature vector, we have to identify the features from the data and select the most suitable ones.

Typically, features are selected by a domain expert or imposed taking into account experience and simple models of the system of interest.[20] However, to do this, we propose to use both ontologies and ontology views to identify the most relevant features to encode data in feature vectors. This proposal is based on the approaches in References 21 and 22, where a domain ontology is used to encode data into feature vectors and an evolutionary algorithm is used to optimize the encoding by feature selection. Specifically, each concept and relation in the ontology is considered as a feature in each feature vector. Then, the number of features is reduced through the evolutionary algorithm. The evolutionary algorithm explores the search space formed by all the features to find the combination of features that minimizes overfitting or other problems arising from a large number of features.

Similarly, in this work, an ontology contains the main concepts and relations in the domain, while the ontology views represent the selected features for a specific action (e.g., evaluate a braking). The domain expert is responsible for selecting the ontology according to the domain. If there is not an available ontology to this domain, the domain expert is responsible for defining one taking into account the domain. Following the running example, the ontology in the railway domain would contain concepts such as door, car, cabin, button, or brake, and the relations between them (see the left of Figure 3).

Therefore, the ontology contains all the concepts and relations for a domain, but only some of these concepts and relations are relevant to fulfill a specific action. For example, the brakes of the train are relevant for the braking system, but the doors are not relevant for it, but both brakes and doors are concepts in the ontology. For this reason, the ontology

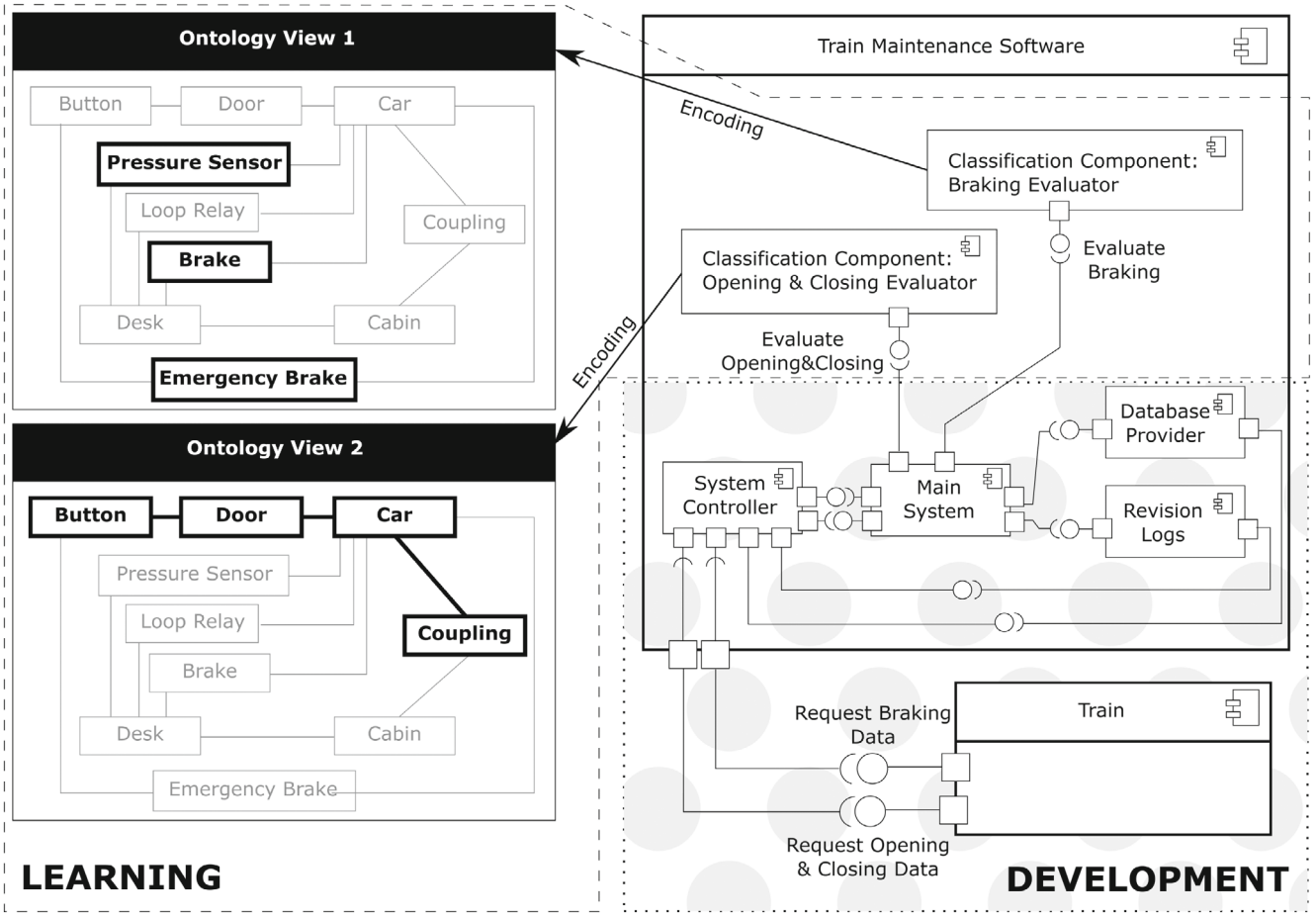

FIGURE 3 Example of views on an ontology and model components for a train maintenance software

views point out the concepts and relations of the ontology that are relevant for a specific action. In Figure 3, the ontology view 1 is an example of an ontology view to develop the braking system. This ontology view indicates that the relevant data to develop the braking system is provided by the pressure sensors, the brakes, and the emergency brakes, so the rest of the concepts and relations are not relevant for the braking system.

2.2.2 | Classifier training and modeling

The main target of this activity is to train the classifier using an ML technique. The type of ML technique will depend on the domain or problem. For example, predictive maintenance applications prevent failures in equipment that make up the production lines on the factory floor. For this kind of application, the most common ML algorithm is random forest followed by neural network based methods (i.e., artificial neuronal networks, convolution neural networks, long short-term memory network, and deep learning).[23] For a different problem like the Internet of Things security, the most common ML algorithms are neural networks followed by the support vector machine technique.[24] For a different domain like medicine, the most popular algorithm in medical informatics is support vector machine, followed by random forest.[25]

The selected ML technique is used to train a classifier, which can tackle different goals. For example, imagine that a train company stores the information about the brakings (e.g., wind, speed, brake status, deceleration, slope, pressure, emergency brake status), the result of each braking (abrupt, smooth, glitchy, or delayed), and possible solutions to improve the brakings (start braking earlier or later, activate or deactivate the automatic braking, use the emergency brakes, etc.). A classifier can be trained to determine if a braking is correct or not, to prevent an abrupt braking, to classify a braking

into abrupt, smooth, glitchy, or delayed, or to provide the driver with a ranking of solutions that help him to make a good decision. The goal of classifiers will depend on the problem and the knowledge base.

Knowledge base is the name for the labeled data used for training. Given the company data, one or more domain experts extract part of this data to train and test the classifier. When the knowledge base is composed of the data extracted for the experts, the knowledge base has to be encoded into feature vectors taking into the concepts and relations that the ontology view points out as relevant. The resulting feature vectors from the encoding compose the training dataset. Specifically, the classifier learns the rule-set through the comparison of the feature vectors which the training dataset contains. After the training process, the classifier can apply the learned rules to extract information from huge volumes of data.

For example, for the braking system, the knowledge base may be composed of a set of brakings. The relevant features for these brakings would depend on the selected concepts and relations of the ontology view. According to the ontology view 1 of Figure 3, the relevant features for these brakings are provided by the pressure sensors, the brakes, and the emergency brakes, so each braking in the knowledge base could be composed of features such as the number of active brakes, the average of the pressure, or the traction effort. In addition to these features, each braking would also contain a numerical value to indicate if the braking is correct (1) or is wrong (0). This numerical value is commonly called a label or tag. The features and the labels would be used to encode each breaking into a feature vector. Then, the classifier could learn when a braking is correct or not through the comparison among the feature vectors. For example, a simple rule that the classifier could learn thanks to the comparison could be *if one of the brakes is not active the train is braking wrong*. Therefore, the trained classifier knows how to determine if a braking is correct or wrong.

To provide the classifier in terms of models, we propose to embed the classifier into a classification component. The top right of Figure 3 shows two classification components that would contain the necessary classifiers to evaluate if a braking is correct or not and to evaluate if the doors are closed or open correctly. These classification components are related to the ontology views to specify what are the relevant features to carry out their purposes.

2.2.3 | Classifier evaluation

The target of this step consists in assessing the effectiveness of the classifier to know if it may be useful for the software system. To do this, the classifier has to test the feature vectors of the training dataset and the results given by the classifier are compared to the labels in these feature vectors. The effectiveness of the classifier is determined by the number of times, the classifier assigns to a feature vector the same value that is in the correspondent label. However, since the training dataset has to be used for both training and testing the classifier, the training dataset is divided into two segments using cross-validation, which is a statistical method of evaluating and comparing ML algorithms by dividing data into two segments: one used to train a classifier and the other used to validate the classifier.[26] Moreover, to reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results are averaged over the rounds.[27]

If the classifiers are not effective enough for a software system, it will be necessary to perform another learning iteration. In this iteration, the ontology view that points out the relevant concepts or relations will be modified to improve a classifier. For example, if the classifier for the braking system was able to classify correctly 80% of the brakings, it would mean that the percentage of fail is very high to be accepted. Because, in 20% of cases, the train could have problems to brake and the braking system would not detect the fail so this fail would not be repaired. Therefore, another iteration would be necessary to improve the effectiveness of the classifier. In the data preparation activity of that new iteration, the domain experts have to modify the ontology view to add other features or to remove some of the features in the ontology view. Adding features can provide more complete information for the classifier. In contrast, removing features can avoid redundancy information and overfitting for the classifier. In our example, the domain experts could remove the pressure sensor of the ontology view 1 (see Figure 3). Therefore, the new classifier would be trained from data provided by the brake and the brake emergency, ignoring the pressure sensor and the rest of the concepts and relations in the ontology.

In contrast, if the classifiers are effective enough for the software system, the development phase starts where the ontology views and the classification components are included as a part of the software system. Specifically, the ontology views indicate what data is relevant. The classification components know how to predict, classify, or evaluate the data. Finally, the evaluation of the classifiers reveals to what extent it is possible to extract valuable information from data through the classifiers, in other words, if the available data is useful enough for the software system.

2.3 | Development phase

The final goal of this phase is to build the software system. To do this, this phase consists of several iterations where each iteration is composed of three activities: iteration modeling, development modeling storming, and test-driven development (TDD). First, in the iteration modeling activity, the team plans (estimate and schedule) the work that will be done in the iteration. Second, the development modeling storming activity is just in time modeling. In this activity, a member of the team identifies an issue that needs to be resolved, he quickly requests the help of a few teammates, the group explores the issue through models until their satisfied that they understand it, and then they continue as before.

While the learning in terms of models phase is based on ML, the development phase is based on models. Therefore, engineers working in this phase must have expertise in modeling, instead of ML. An engineer can be an expert in both ML and modeling. However, it is not necessary to have a single engineer who has expertise in both subjects. A team with an ML expert and a modeler will have the necessary skills to apply the methodology.

Finally, the last activity performs the development following the TDD approach. According to the TDD approach, requirements are turned into test cases, and then the developers write just enough code to fulfill those test cases and refactoring. Specifically, test cases are used to evaluate the designed models. It does not matter whether a classification component or another type of element is used for modeling. The purpose of the test cases is to evaluate if a requirement has been met regardless of the type of element used for modeling.

The right part of Figure 3 shows how the artifacts of the learning phase are integrated into the development phase. Since the final goal of this phase is to build the software system, the developers who are responsible for modeling in this phase have at their disposal the ontology views and the classification components created and evaluated in the previous phase. Specifically, the ontology view indicates what features are relevant for a specific goal. For example, the ontology view 1 in Figure 3 shows that the brakes, the pressure sensors, and the emergency brakes are the relevant features to detect if a train is braking correctly or not. On the other hand, the classification component embeds the classifier of the learning phase, to be used in the models designed in the development phase. For example, in Figure 3, the classification component called braking evaluator embeds the classifier that can automatically evaluate a braking and decide if this braking is correct or not. Figure 3 also shows how that classification component (braking evaluator) is used within the component model for the train maintenance software. This component model is designed in the development phase adding the classification components and other components that are not provided by the learning phase. Therefore, the developers would specify the rest of the components for the train maintenance system. Those components allow to store the data, control system inputs and outputs, or generate the logs (see the bottom right of Figure 3).

Since the learning phase was included before the development phase, it is obvious that models will be affected by the results of the learning phase. Figure 4 shows an example that compares a sequence diagram modeled using AMDD methodology and a sequence diagram modeled using AMDD-ML methodology. Both sequence diagrams model the same requirement: "From current data (e.g., wind, speed, brake status, deceleration, slope, pressure, emergency brake status), inform the driver of the braking status to verify that the train is correctly braking."

In Figure 4, when modelers design the sequence diagram through AMDD, they have no information about how to exploit the data. Therefore, the sequence diagram is completely based on the information described in the requirement and their personal experience. In the sequence diagram designed, the driver receives all the data related to each braking. Then, he will have to think about what decisions to make (e.g., use the emergency brakes or request brake maintenance) based on his experience.

In contrast, when modelers design the sequence diagram through AMDD-ML, they knew that the most relevant features to evaluate the braking status are the brakes, pressure sensor, and emergency brakes and that it is possible to successfully train a classifier to determine if a braking is correct or not. Therefore, the sequence diagram is based on the information described in the requirement and the information obtained from the learning phase (i.e., the ontology view 1 and the classification component: braking evaluator in Figure 3). In the sequence diagram designed, the driver receives the data filtered and a first evaluation of the braking. This first evaluation is provided by the classification component, which is highlighted in black color. When the driver receives the answer, he will have to decide what to do based not only on his experience but also on the first evaluation of the braking which can simplify his decisions.

Since the classifiers will have an error rate, the modelers can consider some alternatives, such as performing a second evaluation when the braking is not correct or providing the driver with all the features when specific conditions are fulfilled. Furthermore, as needed for system development, modelers will also design other types of models, such as state machine diagrams or activity diagrams. Therefore, Figure 4 provides a simple example to be able to show the difference

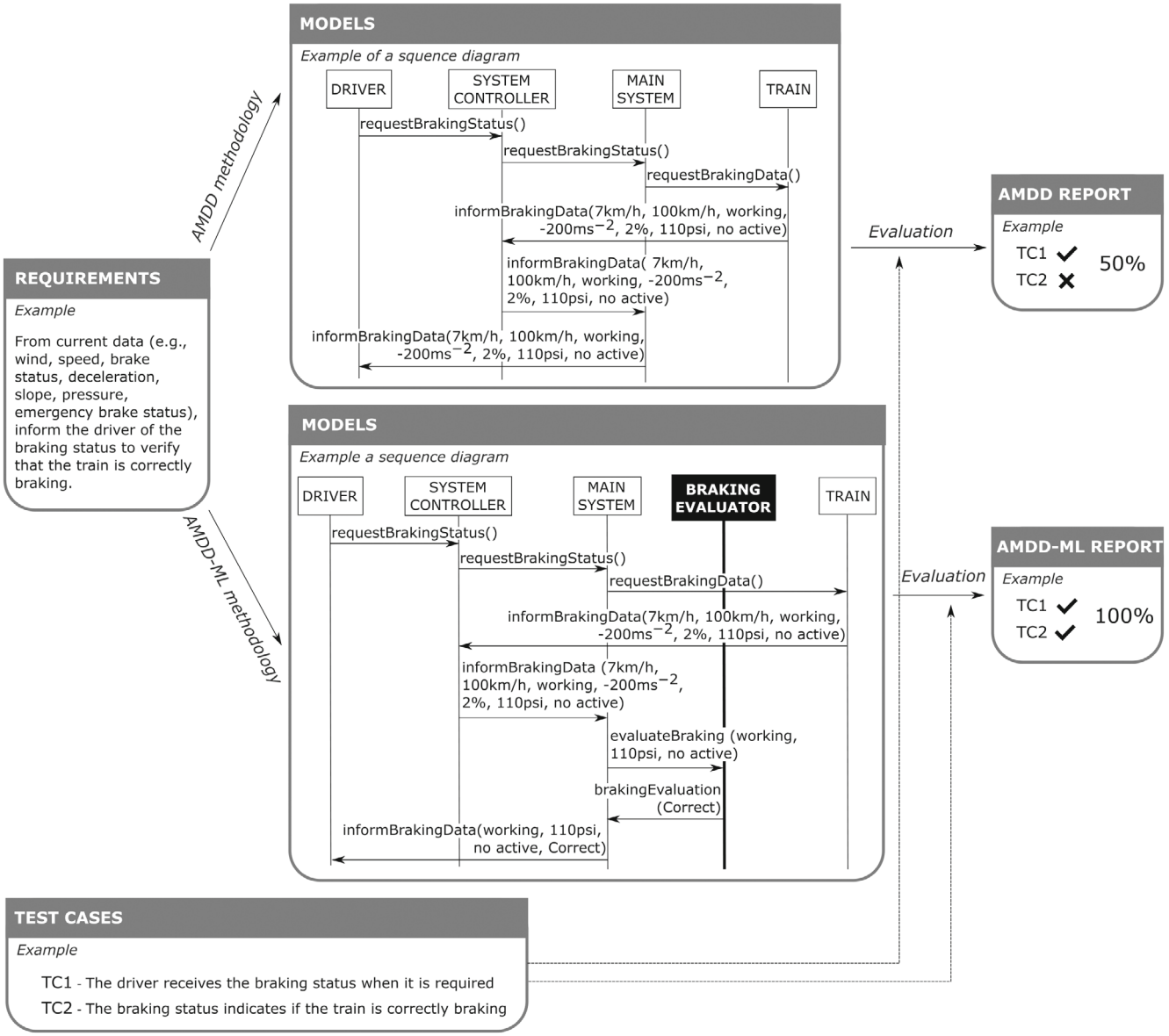

FIGURE 4 Example of a requirement, models designed through AMDD and AMDD-ML, test cases, and reports concerning quality

between models designed through AMDD or AMDD-ML. The real models would be larger, more complex, and of different types as they address the development of a system considering alternative solutions and a larger number of requirements.

2.4 | Release phase

In this phase, a group of qualified people, often both technical staff and project stakeholders, evaluate a model or document. The purpose of this evaluation is to determine if the models not only fulfill the demands of the user community, but also are of sufficient quality to be easy to develop, maintain, and enhance.

2.5 | Production phase

In this phase, the developed software system is put into production making it available to the end-user. Specifically, the product is checked against the objectives that were defined in the envisioning phase. If the objectives are met, the development of the software system is complete and the development cycle is finished.

3 | EVALUATION OF OUR AMDD-ML METHODOLOGY

This section presents the evaluation of our approach: the research questions that our work tackles, the experimental procedure, a description of the real cases where we applied the evaluation, the experimental setup, and the obtained results.

3.1 | Research questions

From the described problem, four research questions arise:

RQ1: What is the difference in terms of quality between the models that result from the development phase in AMDD and the ones that result from the development phase in AMDD-ML?
RQ2: What is the difference in terms of effort between the development of models through AMDD and the development of models through AMDD-ML?
RQ3: What is the difference in terms of productivity between the development of models through AMDD and the development of models through AMDD-ML?
RQ4: What is the difference in terms of satisfaction between the engineers' impressions after developing the models through AMDD and the ones that they have after developing the models through AMDD-ML?

3.2 | Experimental procedure

Therefore, the experiment studies one factor: development method, where a factor is a variable whose effect on the response variable we want to understand.[10] The control in this experiment is the AMDD method (baseline) while the treatment is the AMDD-ML method.

Figure 5 shows an overview of the experiment that was followed to evaluate our AMDD-ML methodology and the baseline. The requirements were extracted from the documentation provided by our industrial partner. Among the requirements are functional and non-functional requirements, but all of them are defined using natural language. These requirements were modeled by computer science students following a methodology: AMDD-ML or AMDD. Since the evaluation is focused only on the development phase, the students had to design the model for the requirements (baseline), and then, design the models for different requirements using the learned artifacts that were given by a domain expert (AMDD-ML). Specifically, the learned artifacts given to students were the classification components. Since the students only have to design the models, we did not provide them with the ontology views.

For each student, we obtained four outputs: the models that were designed for the requirements of one of the real cases using one of the methodologies, the satisfaction questionnaire for this design, the models that were designed for the requirements of another real case using another methodology, and the satisfaction questionnaire for this last design. At the end of the experiment, the models and questionnaires are used to compare our AMDD-ML methodology to the baseline concerning effort, productivity, satisfaction, and quality, to tackle the research questions.

1. *Quality* is defined by ISO 9126-1[28] as a set of several characteristics: functionality, reliability, usability, efficiency, maintainability, portability. Since the functionality is more focused on satisfying end-user expectations, we decided to measure the quality taking into account a sub-characteristic of the functionality: accuracy. Specifically, the ISO 9126-1 defines accuracy as the capability of the software product to provide the right or agreed results or effects. For this reason, accuracy was based on the correctness of the models constructed.

   Specifically, each requirement had at least one associated test case defined by domain experts. Test cases focused on the correct design of different parts of a model. Figure 4 shows an example of how models are evaluated using the test cases. In this example, a requirement is associated with two test cases. The first one verifies if the driver receives the braking status when it is required. The second one verifies if the braking status indicates that a braking is correct or not.

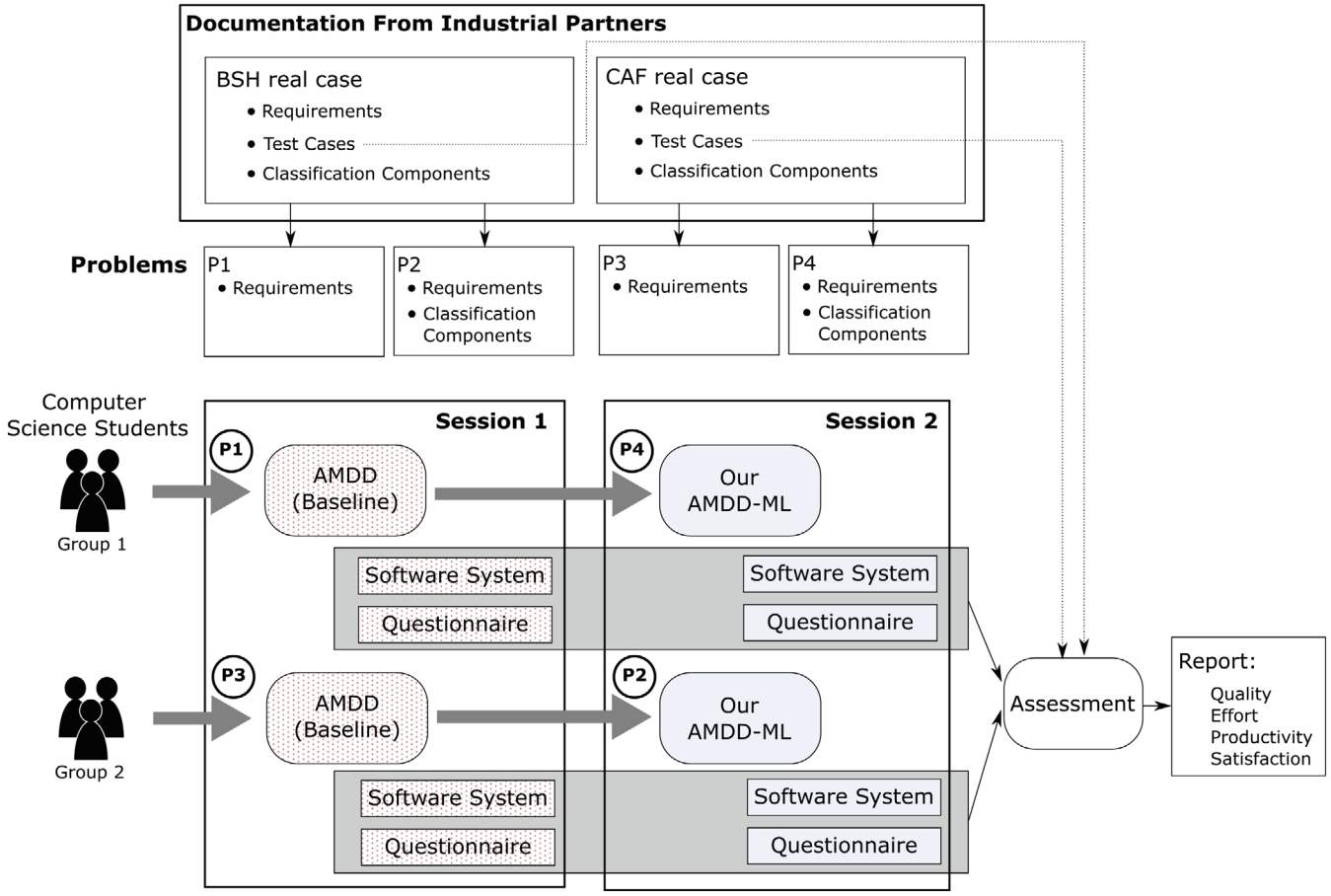

FIGURE 5 Experimental procedure

The accuracy was measured as the percentage of test cases that are successfully passed. Following the example in Figure 4, the two test cases are used to evaluate both the sequence diagram designed through AMDD and the sequence diagram designed through AMDD-ML. The two sequence diagrams satisfy the first test case, but the second one is only satisfied by the sequence diagram designed through AMDD-ML. Therefore, based on these two test cases, the quality of the sequence diagram designed through AMDD would be 50%, and the quality of the sequence diagram designed through AMDD-ML would be 100%. As in the example in Figure 4, the accuracy was calculated based on how many of the 9 test cases for the CAF case and 11 test cases for the BSH case passed successfully.

2. *Effort* is the ratio of time to develop a system per developer.[29] Therefore, the effort was measured as the time taken by each student to model the requirements for each real case.
3. *Productivity* is the ratio of quality work to effort.[29] Therefore, productivity was measured as the accuracy to effort ratio (accuracy/effort).
4. *Satisfaction* is the positive attitude toward the use of the development method.[29] Therefore, satisfaction was measured through a scale questionnaire whose questions are designed to evaluate the perceived usefulness, perceived ease of use, and intention to use. Each of these metrics was measured through several questions of a scale questionnaire where the higher mark in the scale indicates the best satisfaction.

3.3 | Real industrial cases

1. *CAF case*: The first real case where we applied our methodology was CAF, a worldwide provider of railway solutions. Their trains can be found all over the world and in different forms (regular trains, subway, light rail, monorail, etc.). A train unit is furnished with multiple pieces of equipment through its vehicles and cabins. Some of these pieces (e.g., sensors) control the braking of the trains, and the braking data are sent automatically and

continuously to the maintenance system of the company. From the data braking, this system has to evaluate if the trains are braking correctly and if the brakes need maintenance. To prevent possible failures in the brakes, the system analyzes the braking data of a train, and then, the system decides if the train needs maintenance or not.

The requirements that were selected from the CAF case involved the modeling of the maintenance system. To do this, the modeling software systems had to take advantage of the braking data. These braking data are provided by the trains that have been developed for decades, and the software system had to be able to evaluate the state of the brakes and to prevent possible failures in the brakes taking into account the provided braking data.

2. *BSH case*: The second real case where we applied our extended methodology was BSH, the leading manufacturer of home appliances in Europe. Their induction division has been producing Induction Hobs under the brands of Bosch and Siemens for the last 15 years. Their premium Induction Hobs provide information about their state remotely. The obtained information from the Induction Hobs is used both for statistical purposes and for customer services, such as preventive maintenance.

   The requirements that were selected from the BSH case involved the modeling of preventive maintenance. Therefore, the designed models had to take advantage of the information that is provided by the Induction Hobs to prevent what Inductions Hobs have to be repaired before breaking.

3.4 | Experimental setup

This study was evaluated through computer science students from Universidad San Jorge (USJ, Spain) who have previous knowledge of working with models. Participation in this experiment was voluntary and the participants were rewarded with extra points. We inform them about the experiment, but the students did not know that the methodology was proposed by the researchers. The researchers do not teach classes to the students. Nevertheless, one of the researchers and the teacher of the subject were available in the sessions of the experiment. In total, 31 students were recruited. These students were divided into two groups, to perform a paired design blocked by experimental objects[10] shown in Figure 5.

Both groups used the baseline methodology in the first session and AMDD-ML in the second session. The first group designs the models for the maintenance system of CAF. In contrast, the second group designs the models for the preventive system of BSH. In the second session, both groups used the AMDD-ML methodology. In this case, the first group designs the models for the preventive system of BSH, and the second group designs the models for the maintenance system of CAF. Since the problem used in the first session is different from the problem in the second session, we avoided the learning effect from one session to the other. The materials of the experiment (i.e., the problem descriptions, the satisfaction questionnaire, and the results) are available in the repository at https://goo.gl/YRrXSp.

3.5 | Results

To answer the research questions, this subsection presents the results of quality, effort, productivity, and satisfaction obtained through the evaluation. Figure 6 shows the comparison of the baseline results against the AMDD-ML results. Moreover, the collected data was analyzed for the factor being observed: development method (AMDD and AMDD-ML). Specifically, since the number of participants is less than 50, we use the Shapiro–Wilk test the normality of the collected data.[30] Then, Mann–Whitney *U* test[31] was used for the variables whose data follows a normal distribution (i.e., accuracy) and *T*-Student[32] test for the variables whose data does not follow a normal distribution (i.e., effort, productivity, perceived usefulness, perceived ease of use, and intention to use).

In response to RQ1, the top left box-and-whisker plot of Figure 6 compares the quality of the baseline to the quality of AMDD-ML. The quality was measured through the accuracy as the percentage of test cases passed. Specifically, the higher percentage of test cases passed indicates the best accuracy. AMDD achieved a 70.54% accuracy mean and AMDD-ML achieved a 85.69% accuracy mean. All the subjects that used AMDD-ML to develop the requirements of any real case achieves an accuracy mean up to 85%, except for one subject with 54%. Moreover, in the statistical analysis, Mann–Whitney *U* test provided a *p*-value equal to 0.043 for accuracy. Therefore, since the *p*-value is less than 0.05, we can conclude that there are significant differences between the two methodologies regarding accuracy.

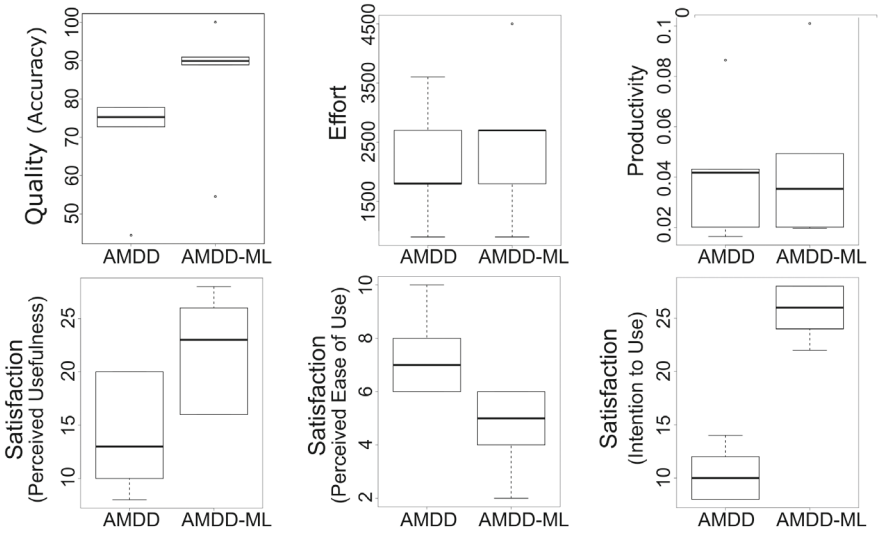

FIGURE 6 Box-and-whisker plots for quality, effort, productivity, and satisfaction

In response to RQ2, the top center box-and-whisker plot of Figure 6 compares the effort of the baseline to the effort of AMDD-ML. The effort was measured as the time in seconds that subjects take to solve the requirements for each real case. Specifically, the less time spent in the development indicates the best effort. AMDD achieved a mean effort of 2100 s and AMDD-ML achieved a mean effort of 2500 s. Moreover, in the statistical analysis, *T*-Student test provided a *p*-value equal to 0.48 for effort. Therefore, since the *p*-value is greater than 0.05, we can conclude that there are no significant differences between the two methodologies regarding the effort.

In response to RQ3, the top right box-and-whisker plot of Figure 6 compares the productivity of the baseline to the productivity of AMDD-ML. Productivity was measured as the accuracy to effort ratio. Specifically, the higher ratio indicates the best productivity. AMDD achieved a 0.042 mean productivity and AMDD-ML achieved a 0.043 mean productivity. Moreover, in the statistical analysis, *T*-Student test provided a *p*-value equal to 0.91 for productivity. Therefore, since the *p*-value is greater than 0.05, we can conclude that there are no significant differences between the two methodologies regarding productivity.

In response to RQ4, the bottom box-and-whisker plots of Figure 6 compare the satisfaction of the baseline to the satisfaction of AMDD-ML. Satisfaction was measured concerning perceived usefulness, perceived ease of use, and intention to use. Each of these metrics was measured through several questions of a scale questionnaire (3 for perceived usefulness, 1 for ease of use, 3 for intention to use). Specifically, the higher mark on the scale indicates the best satisfaction. For perceived usefulness mean, AMDD achieved 14 marks out of 30 and AMDD-ML achieved 22 marks out of 30. For perceived ease of use mean, AMDD achieved 7.33 marks out of 10 and AMDD-ML achieves 4.67 marks out of 10. For the intention to use mean, AMDD achieved 10.33 marks out of 30 and AMDD-ML achieved 25.67 marks out of 30. Moreover, in the statistical analysis, *T*-Student test provided a *p*-value equal to 0.02 for perceived usefulness and 0.02 for perceived ease to use, and a *p*-value less than 0.0001 for the intention to use. Therefore, since the *p*-value is less than 0.05 for all the satisfaction variables, we can conclude that there are significant differences between the two methodologies regarding the perceived usefulness, the perceived ease to use, and the intention to use.

4 | DISCUSSION

The results reveal that AMDD-ML provides better quality than the baseline. Moreover, thanks to the perceived usefulness, we find out that the development is easier for the subjects if the classifier is available before the development of the requirements. If we have into account only test cases that involve data, we will obtain a 8.33% accuracy mean for the baseline and a 91.67% accuracy mean for AMDD-ML. Therefore, for our evaluation, AMDD-ML is beneficial to simplify the data processing in the development phase.

In contrast, the results reveal a higher effort with AMDD-ML than with the baseline. Although the difference between the effort with AMDD-ML and the effort with the baseline is small, this difference causes the productivity to be the same for both methodologies. However, this difference may be caused by a lack of knowledge because the students have worked with models but they have never worked with ML and classifiers. This fact also causes that the perceived ease of use to the baseline is greater than the perceived ease of use to AMDD-ML. For the students, the baseline is easier because they often use it for their projects. In contrast, they did not know ML and classifiers, so they had to do an additional understanding effort.

Even though the effort for AMDD-ML is greater than the effort for the baseline, the satisfaction results reveal a high intention to use it in the future. The students consider that the effort is worth it, and they consider that AMDD-ML is relevant for both their future projects and the industrial sector.

Based on the obtained results, we consider that it is worth following the research of the proposed methodology. Therefore, as future work, we expect to improve our methodology considering different aspects, such as the iteration between the envisioning, learning, and development phases or the development of the ontology. Moreover, taking into account the promising results of our methodology, as future work, we plan to improve and evaluate our methodology through a deeper experiment. This experiment will be based not only on the development phase but also on the learning phase and will consider bottom-up methods in addition to top-down methods in data mining.

5 | THREATS TO VALIDITY

In this section, we use the classification of threats of validity of References 33 and 34 to acknowledge the limitations of our approach.

Reliability: This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers.

- Fishing: This threat appears when researchers search for a specific result. We avoided this threat by using all the collected data, none of which was removed for any reason whatsoever.
- Reliability of measures: This threat appears when there is no guarantee that the outcomes will be the same if a phenomenon is measured twice. We avoided this threat by measuring accuracy, effort, and productivity with objective metrics. Unfortunately, since satisfaction is subjective, this measurement suffered from this threat.
- Data collection: This threat appears when data collection is not done in the same way throughout the different sessions. To minimize this threat, the same procedure was used for data collection in the two sessions.
- Completion data: This threat appears when there are some missing data after the data collection process. To minimize this threat, two observers (the teacher and one of the researchers) tested the data coherence when the subjects finished each exercise.

Internal validity: This aspect of validity is of concern when causal relations are examined. There is a risk that the factor being investigated may be affected by other neglected factors.

- Instrumentation: This threat appears when the artifacts used for the experiment affect the experiment. To mitigate this threat, the artifacts provided to the students (classifications components) were designed by an ML expert, who was not involved in the research.
- Selection: This threat appears when the outcomes of the experiment may depend on the type of subjects. The experiment was affected by this threat since all of the subjects were recruited from the computer science course.
- Learning of objects: This threat appears when the subjects learn something during the experiment that may influence later tasks. We avoided this threat by using two different problems in our design, so subjects do not get the chance to learn objects.
- Subject motivation: This threat means that the subjects are not motivated to participate in the experiment. To avoid this threat, the participation was voluntary and the subjects were rewarded with extra points.

- Understanding This threat appears when the subjects do not understand how to proceed to the experiment. To minimize this threat, the vocabulary of the requirements was reviewed by the computer science professor. The vocabulary that is used by our industrial partners may be too complex for people who are not domain experts, so some specific words of the domain were translated into more understandable words (e.g., PLC was translated as railway system).
- Diffusion or imitation of treatments: This threat appears when a control group learns about the treatment from the group in the experiment study or the control group tries to imitate the behavior of the group in the study. To minimize this threat, both groups have different problems and they perform the experiment at the same time so the subjects can have no time to exchange information.

Construct validity: This aspect of validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind and what is investigated based on the research questions.

- Mono-operation bias: This threat appears when treatments depend on a single tool only. To avoid this threat, the subject can use the modeling tool that they prefer.
- Mono-method bias: This threat is due to using a single type of measure or observation. Quality, effort, productivity, and satisfaction measurements suffer from this threat. To minimize its effect regarding effort and productivity, we mechanized the measurement as much as possible using automatic timing. Moreover, to minimize this threat regarding satisfaction, the satisfaction questionnaire includes redundant questions expressed in different ways.
- Iteration of different treatments: This threat appears when there are several treatments applied at the same time. To avoid this threat, the two treatments (AMDD and AMDD-ML) were addressed by the subjects in two different sessions.
- Interaction of testing and treatment: This threat means that the application of treatments, may make the subjects more sensitive or receptive to the treatment. To avoid this threat, the researchers were responsible for measuring the treatment.
- Hypothesis guessing: This threat means that the subject may guess the hypotheses and work to fulfill them. To mitigate this threat, the subjects were not informed about research questions or the goal of the experiment.
- Evaluation apprehension: This threat appears when the subjects are afraid of being evaluated. To avoid this threat, the subjects were volunteers and they were informed that this experiment could not negatively affect their subject points.
- Experimenter expectancies: This threat means that researchers can bias the results of a study both consciously and unconsciously based on what they expect from the experiment. To mitigate this threat, the requirements and responses (test cases) were extracted from the documentation of our industrial partners. This documentation was provided by domain experts who were not involved in this article.

External validity: This aspect of validity is concerned with to what extent it is possible to generalize the finding, and to what extent the findings are of relevance for other cases.

- Interaction of selection and treatment: This threat appears when a subject population is not representative of the population we want to generalize. Unfortunately, since the subjects are students instead of development engineers, this work suffers from this threat. However, using students as subjects instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts.[35-37] Nevertheless, it would be necessary to replicate the experiment with different subject roles to mitigate this threat.
- Interaction of history and treatment: This threat appears when the experiment is conducted at a special time or day which affects the results. To mitigate this threat, both sessions of the experiment were conducted one after the other.
- Influence of the domain: This threat appears when the outcomes depend on a specific domain. To minimize this threat, instead of a specific domain, the experiment was based on two different real cases: a railway domain and a home appliances domain. Furthermore, the proposed approach extends the AMDD methodology, which is domain-independent, embedding the data-oriented perspective of CRISP and SEMMA, which are also domain-independent. Considering that a methodology is applied in different domains, the proposed methodology is also designed independently of the domain. Nevertheless, the experiment and its results should be replicated in more domains before assuring their generalization.

6 | RELATED WORK

In the literature, there is a wide range of works that compare software development methodologies. Panach et al.[38] present an experiment where MDD and traditional development are compared. Dybå and Dingsøyr[39] present a study about agile software development, where they analyze the benefits and the limitations of the agile methods taking into account the strength of evidence. Awad[40] presents a comparison between some traditional and agile methodologies with regard to their strengths and weaknesses, and this work also provides the challenges associated with implementing agile processes in the software industry. Asadi and Ramsin[41] provide a review of several prominent MDA-based methodologies, and they present a criteria-based evaluation that highlights their strengths and weaknesses.

Although the previous works compare methodologies where models are used with methodologies where models are not used, none of these works take into account the development of data-oriented software systems. In contrast, the data-oriented software systems are fundamental in our work. On the other hand, we have reviewed the measurements that are used in these works to compare methodologies to select the measurements for our experiment.

Some works rely on models to improve some aspects of the data-oriented software systems, such as quality, architecture, or evolution. DICE[42] is a quality-aware methodology for data-intensive cloud applications. DICE has into account properties that are largely ignored by existing models and QA techniques, such as volumes, velocities, and data location. Guerriero et al.[43] propose a preliminary step in the supporting model-driven design of big data applications. Specifically, they provide an architecture and the foundational means as a starting point to further elaborate tool-support for model-driven big data design. JUNIPER[44] is an EU-funded project which assists data-oriented developers to create architecture-aware software where MDD is employed to ease development, portability, and deployment instead of expressing an entire data-oriented system at the source-code level. Ceri et al.[2] propose mega-modeling as a new holistic data and model management system for the acquisition, composition, integration, management, querying, and mining of data and models, capable of mastering the co-evolution of the data and models and of supporting the creation of what-if analyses, predictive analytics, and scenario explorations.

None of the previous works address the goal of combining the advantages of using models and the advantages of using ML techniques to develop data-oriented software systems. In fact, none of the previous works compare in terms of accuracy, time, accuracy to time ratio, perceived usefulness, perceived ease of use, and intention to use, with a conventional development using models as our work does.

Furthermore, some recent works focus on methodologies for software development. These works research how development methodologies are used or how to improve these methodologies. Hron and Obwegeser[45] perform a systematic review of the literature to analyze why and how the Scrum methodology has been modified in different instances. In Reference 46, the systematic literature review identifies the techniques employed in cloud computing environments that are useful for agile development. Molina-Ríos and Pedreira-Souto[47] compare web development methodologies including some agile methodologies. The comparison is based on the dynamic features presented during the lifecycle to identify their use, relevance, and characteristics. In Reference 48, a novel approach is proposed to make use of a collaborative filtering strategy to recommend valuable entities related to the metamodel under construction.

Although these works propose several modifications to development methodologies, none of these modifications focus on benefiting from a huge volume of data through an agile methodology. In contrast, that is the main goal of the methodology proposed in this work.

Our previous work was focused on bridging ML and models. In Reference 21, we present an approach to find out the optimal or near optimal encoding for model fragments through an evolutionary algorithm based on an ontology. In contrast, this work proposes to extend the AMDD methodology with the learning in terms of models phase, which analyzes data through ontology views and classification components at model level. Moreover, the evaluation assesses the improvement given by our extension to the development of oriented-data software systems in two industrial domains.

7 | CONCLUSION

With the emergence of huge amounts of data, some methodologies have been defined to guide the development of data-oriented software systems. We propose the AMDD-ML methodology, which empowers to develop data-oriented software systems through models and learning. This methodology takes advantage of the high level of abstraction that is

provided by MDD and the data-oriented development that is provided by data mining methodologies. We evaluate our AMDD-ML methodology in terms of quality, effort, productivity, and satisfaction. To do so, we compared our methodology to a baseline in two industrial domains.

Despite the extra effort, the use of ontology views and classifier components at model level pays off. AMDD-ML is beneficial for software development, and this fact is shown through the quality results, the perceived usefulness, and intention to use.

The promising results of this work lead to interesting research questions for the future, such as the following: *Can we achieve similar results in other industrial domains?*; *What are the benefits from engineers' points of view?*; *What are the benefits from a company point of view?*; *How scalable is the proposed approach to larger scenarios?*. Therefore, to answer some of these research questions and to test the external validity of the results of this work, the next step is to bring the methodology to the industry through a tool support. It will facilitate the application of the methodology in other domains, the exhaustive study of the entire methodology and its performance, and the analysis of the advantages and disadvantages of the methodology in different industrial contexts.

AUTHOR CONTRIBUTIONS

Conceptualization and defining the problem statements: A. C. Marcén, O. Pastor, and C. Cetina. Design of proposed methodology: A. C. Marcén. Design of the evaluation and testing: A. C. Marcén and F. Pérez. Experimental results analysis and discussion: A. C. Marcén and C. Cetina. Drafting of the manuscript: A. C. Marcén. Review of the manuscript for important intellectual content: A. C. Marcén and F. Pérez. Study supervision: O. Pastor and C. Cetina.

ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project VARIATIVA under Grant PID2021-128695OB-I00, and in part by the Gobierno de Aragón (Spain) (Research Group S05_20D).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available in the repository at https://goo.gl/YRrXSp.

ORCID

Ana C. Marcén https://orcid.org/0000-0002-5054-4618
Francisca Pérez https://orcid.org/0000-0001-6371-915X
Óscar Pastor https://orcid.org/0000-0002-1320-8471
Carlos Cetina https://orcid.org/0000-0001-8542-5515

REFERENCES

1. Embley DW, Liddle SW. Big data—Conceptual modeling to the rescue; 2013:1-8; Springer.
2. Ceri S, Della Valle E, Pedreschi D, Trasarti R. Mega-modeling for big data analytics. *Concept Model*. 2012; 75:1-15.
3. Ponce M, Spence E, Gruner D, van Zon R. Scientific computing, high-performance computing and data science in higher education. *arXiv preprint arXiv:1604.05676*; 2016.
4. Zicari RV, Rosselli M, Ivanov T, et al. Setting up a big data project: challenges, opportunities, technologies and optimization; 2016:17-47.
5. Shafique U, Qaiser H. A comparative study of data mining process models (KDD, CRISP-DM and SEMMA). *Int J Innovat Sci Res*. 2014;12(1):217-222.
6. Selic B. The pragmatics of model-driven development. *IEEE Softw*. 2003;20(5):19-25.
7. Hartmann T, Moawad A, Fouquet F, Le Traon Y. The next evolution of MDE: a seamless integration of machine learning into domain modeling. *Softw Syst Model*. 2019;18(2):1285-1304.
8. Kienzle J, Mussbacher G, Combemale B, et al. Toward model-driven sustainability evaluation. *Commun ACM*. 2020;63(3):80-91.
9. Burden H, Heldal R, Whittle J. Comparing and contrasting model-driven engineering at three large companies; 2014; ACM.
10. Juristo N, Moreno AM. *Basics of Software Engineering Experimentation*. Springer Science & Business Media; 2013.
11. Ambler SW. *The Object Primer: Agile Model-Driven Development with UML 2.0*. Cambridge University Press; 2004.
12. Ambler SW. *Agile Model Driven Development (AMDD): The Key to Scaling Agile Software Development*. Ambysoft Inc. http://www.agilemodeling.com/essays/amdd.htm; 2003. [Online. Accessed 31-December-2019].
13. Leskovec J, Rajaraman A, Ullman JD. *Mining of Massive Datasets*. 2nd ed. Cambridge University Press; 2014.
14. Lary DJ, Remer L, MacNeill D, Roscoe B, Paradise S. Machine learning and bias correction of MODIS aerosol optical depth. *IEEE Geosci Remote Sens Lett*. 2009;6(4):694-698.

15. Nguyen TT, Armitage G. A survey of techniques for internet traffic classification using machine learning. *IEEE Commun Surv Tutor*. 2008;10(4):56-76.
16. Beydeda S, Book M, Gruhn V. *Model-driven Software Development*. Vol 15. Springer; 2005.
17. Shabtai A, Moskovitch R, Elovici Y, Glezer C. Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. *Inf Secur Techn Rep*. 2009;14(1):16-29.
18. Bianchini M, Maggini M, Jain LC. *Handbook on Neural Information Processing*. Springer; 2013.
19. Chandrashekar G, Sahin F. A survey on feature selection methods. *Comput Electr Eng*. 2014;40(1):16-28.
20. Anderson B, McGrew D. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity; 2017:1723-1732; ACM.
21. Marcén AC, Pérez F, Cetina C. *Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation*. Springer International Publishing; 2017:491-505.
22. Marcén AC, Pérez F, Pastor Ó, Cetina C. Enhancing software model encoding for feature location approaches based on machine learning techniques. *Softw Syst Model*. 2021; 21:1-35.
23. Carvalho TP, Soares FA, Vita R, Francisco RP, Basto JP, Alcalá SG. A systematic literature review of machine learning methods applied to predictive maintenance. *Comput Ind Eng*. 2019;137:106024.
24. Ahmad R, Alsmadi I. Machine learning approaches to IoT security: a systematic literature review. *IoT*. 2021;14:100365.
25. Hasan N, Bao Y. Understanding current states of machine learning approaches in medical informatics: a systematic literature review. *Health Technol*. 2021;11(3):471-482.
26. Refaeilzadeh P, Tang L, Liu H. Cross-validation; 2009:532-538; Springer.
27. Song Q, Jia Z, Shepperd M, Ying S, Liu J. A general software defect-proneness prediction framework. *IEEE Trans Softw Eng*. 2011;37(3):356-370.
28. Standardization FIO, Commission IE. *Software Engineering–Product Quality: Quality Model*. Vol 1. ISO/IEC; 2001.
29. ISO I. *Systems and Software Engineering–Vocabulary*. IEEE Computer Society; 2010.
30. Razali NM, Wah YB. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-darling tests. *J Stat Model Anal*. 2011;2(1):21-33.
31. McKnight PE, Najab J. Mann-Whitney U Test. *Corsini Encycl Psychol*. 2010;1-1. doi:10.1002/9780470479216.corpsy0524
32. Smith G. Learning statistics by doing statistics. *J Stat Educ*. 1998;6(3):1-12.
33. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empir Softw Eng*. 2009;14(2):131-164.
34. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering*. Springer Science & Business Media; 2012.
35. Echeverria J, Pérez F, Cetina C, Pastor O. Comprehensibility of variability in model fragments for product configuration; 2016:476-490; Springer.
36. Reinhartz-Berger I, Figl K, Haugen Ø. Comprehending feature models expressed in CVL; 2014:501-517; Springer.
37. Kitchenham BA, Pfleeger SL, Pickard L, et al. Preliminary guidelines for empirical research in software engineering. *IEEE Trans Softw Eng*. 2002;28(8):721-734.
38. Panach JI, España S, Dieste O, Pastor O, Juristo N. In search of evidence for model-driven development claims: an experiment on quality, effort, productivity and satisfaction. *Inf Softw Technol*. 2015;62:164-186.
39. Dybå T, Dingsøyr T. Empirical studies of agile software development: a systematic review. *Inf Softw Technol*. 2008;50(9):833-859.
40. Awad MA. *Comparison Between Agile and Traditional Software Development Methodologies*. University of Western Australia; 2005.
41. Asadi M, Ramsin R. MDA-based methodologies: an analytical survey; 2008:419-431; Springer.
42. Casale G, Ardagna D, Artac M, et al. Dice: Quality-driven development of data-intensive cloud applications; 2015:78-83; IEEE.
43. Guerriero M, Tajfar S, Tamburri DA, Di Nitto E. Towards a model-driven design tool for big data architectures; 2016:37-43; ACM.
44. Gray I, Chan Y, Audsley NC, Wellings A. Architecture-awareness for real-time big data systems; 2014:151; ACM.
45. Hron M, Obwegeser N. Why and how is Scrum being adapted in practice: a systematic review. *J Syst Softw*. 2022;183:111110.
46. Younas M, Jawawi DN, Ghani I, Fries T, Kazmi R. Agile development in the cloud computing environment: a systematic review. *Inf Softw Technol*. 2018;103:142-158.
47. Molina-Ríos J, Pedreira-Souto N. Comparison of development methodologies in web applications. *Inf Softw Technol*. 2020;119:106238.
48. Di Rocco J, Di Ruscio D, Di Sipio C, Nguyen PT, Pierantonio A. MemoRec: a recommender system for assisting modelers in specifying metamodels. *Softw Syst Model*. 2022;1-21.

How to cite this article: Marcén AC, Pérez F, Pastor Ó, Cetina C. Evaluating the benefits of empowering model-driven development with a machine learning classifier. *Softw Pract Exper*. 2022;52(11):2439-2455. doi: 10.1002/spe.3133