An MDE Method for Improving Deep Learning Dataset Requirements Engineering using Alloy and UML

Benoît Ries[a], Nicolas Guelfi[b] and Benjamin Jahić[c]
University of Luxembourg, Esch-sur-Alzette, Luxembourg

Keywords: Model-driven Engineering, Software Engineering, Requirements Engineering, EMF, Sirius, Alloy.

Abstract: Since the emergence of deep learning (DL) a decade ago, only few software engineering development methods have been defined for systems based on this machine learning approach. Moreover, rare are the DL approaches addressing specifically requirements engineering. In this paper, we define a model-driven engineering (MDE) method based on traditional requirements engineering to improve datasets requirements engineering. Our MDE method is composed of a process supported by tools to aid customers and analysts in eliciting, specifying and validating dataset structural requirements for DL-based systems. Our model driven engineering approach uses the UML semi-formal modeling language for the analysis of datasets structural requirements, and the Alloy formal language for the requirements model execution based on our informal translational semantics. The model executions results are then presented to the customer for improving the dataset validation activity. Our approach aims at validating DL-based dataset structural requirements by modeling and instantiating their datatypes. We illustrate our approach with a case study on the requirements engineering of the structure of a dataset for classification of five-segments digits images.

1 INTRODUCTION

Deep Learning (DL) has emerged in the last decade from artificial intelligence, dating from the Dartmouth conference in 1956, combined with the recent emergence of Graphical Processing Units (GPUs). These GPUs, providing a boost in computational power, initiated the popularity of artificial intelligence in everyday life applications, such as vocal personal assistants, entertainment.

Deep learning techniques require large datasets either for their training phase, in the case of supervised learning neural networks, or for their learning phase in the case of unsupervised learning networks. It is common for real applications to have datasets as large as tens of thousands of data. In one of our previous study (Jahic et al., 2019; Jahic et al., 2020), by analysing the datasets and learning outcomes of the training of neural networks, we discovered that many issues were related to the poor specification of the datasets' structure.

Datasets are critical input artefacts necessary to construct DL-based systems. As such they should be precisely specified. Let's take the following example to illustrate our claim: a DL-based system aiming to classify hand-written digits will have difficulty to classify sevens *without* a bar if the training/learning dataset comprises solely sevens *with* bars. With our method proposed in this paper, we address this type of issues by exploring the dataset *requirements*, before the actual learning phase starts. Errors in datasets are then detected *earlier* in the software engineering development lifecycle, reducing consequently the cost of dealing with these errors.

Software engineering appeared in 1968 (Naur and Randell, 1969) as a solution to the so-called "software crisis". In the last 50 years, software applications have benefitted from research and development effort in the area of software engineering. Unfortunately DL-based systems are rarely defined systematically following a software engineering methodology. DL approaches could take advantage of traditional software engineering methodologies. Typical DL dataset requirements engineering is limited to an activity whose output is informal and not based on standard modeling languages neither grounded on formal semantics. Datasets are also used in the context of software test engineering. In this context, the se-

[a] https://orcid.org/0000-0002-8680-2797
[b] https://orcid.org/0000-0003-0785-3148
[c] https://orcid.org/0000-0002-1120-1196

Ries, B., Guelfi, N. and Jahić, B.
An MDE Method for Improving Deep Learning Dataset Requirements Engineering using Alloy and UML.
DOI: 10.5220/0010216600410052
In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2021)*, pages 41-52
ISBN: 978-989-758-487-9

lection of relevant test datasets using model-driven software engineering has shown to be an interesting approach (Ries, 2009). It is only recently that interest is growing in the research community to cross-fertilize the aforementioned two areas (Hill et al., 2016; Khomh et al., 2018; Vogelsang and Borg, 2019; Burgueño et al., 2019).

In this paper, we provide a model-driven software engineering (MDSE) method using semi-formal and formal modeling. The dataset requirements engineering team is composed of the analysts, formal experts and the customer. On the one hand, semi-formal modeling allows one to describe concepts of the dataset under development and to communicate them among the heterogeneous stakeholders at the desired level of abstraction. We use semi-formal modeling to describe the concepts required for the structure of the datasets, i.e., the datatypes of interest.On the other hand, formal modeling allows us to construct a precise description of the requirements on the structure of the datasets. Formal interpretation tools like the Alloy Analyzer and Kodkod provide instances of data *specification* that satisfy the modeled requirements.

In this paper, we contribute to the introduction of software engineering rigor in the requirements specification of datasets for DL-based systems. Section 2 presents our iterative process for dataset requirements engineering with formal model execution. Section 3 illustrates our approach with a case study on the engineering of the requirements for a five-segments digits dataset with UML-compliant modeling and Alloy models execution. Section 4 discusses some important aspects. Section 5 positions our paper w.r.t. some current related works. Finally, we conclude in Section 6 and present some future works in Section 7.

2 AN ITERATIVE EXECUTABLE DATASET REQUIREMENTS MDE METHOD

2.1 Using Executable Models to Improve the Engineering of Datasets Requirements

In this paper, our iterative method focuses on the engineering of datasets structural requirements. Concretely, our focus is on the elicitation and modeling of data types for the dataset under development. With this method the DL scientist can follow a clear process for dataset requirements engineering, supported by tools, while benefitting from advances in established software engineering methods.

Following traditional software engineering, our method takes place in a *typical requirements engineering phase*. This phase is usually (Sommerville, 2016) composed of the following activities: elicitation, specification, validation and evolution. Requirements engineering is performed by an analyst together with the customer of the system to be produced. Overall, requirements engineering is a crucial phase. It is well-known that the workload put in producing quality requirements reduces the amount of workload put in the succeeding phases, i.e., design, production, testing, deployment. Moreover, improving the requirements engineering phase eases the earlier discovery of errors in the dataset under development. Consequently, we make the hypothesis that improving the engineering of the dataset requirements phase will improve the dataset design and production phases.

Our approach is based on executing dataset requirements to improve their specifications. The solution that we explore is to specify the datasets requirements with an executable model, that we name the *dataset requirements concept model* (DRCM). DRCM is given an operational semantics as a translation to a formal language. Thanks to MDE techniques, we generate a skeleton specification of the formal DRCM (FDRCM). The formal analysts then finalise the FDRCM. With this formal semantics, we are able to interpret the dataset requirements concept model. We name these formal interpretations: model executions. A model execution is a set of datatype specification instances. The resulting model executions are formatted in a way that should suit the customer and the analyst such that they are able to decide on the validation of the DRCM.

The objective of our method is to provide help to the requirements analysts for the validation of the datatypes that will structure the dataset. Typically, in our context, these analysts are data scientists who are responsible for the dataset engineering.

2.2 Iterative Dataset Requirements Elicitation Process

2.2.1 Process Overview

Figure 1 shows the main elements of the business process we propose in this paper using the BPMN (Object Management Group, 2011) notation. Our process is composed of: four *activities* denoted by dark-blue horizontal rectangles; five *artefacts* denoted by light-blue vertical rectangles with a folded top-right corner; one *exclusive gateway* denoted by a dark-blue diamond with a white X, this gateway allows numerous iterations in our process until reaching the validation of the dataset requirements concept model; one *start event* denoted by a white circle; one *end event* denoted by a dark blue-filled circle.

2.2.2 Model Dataset Structural Requirements

The first activity in our process is the modeling of dataset structural requirements. Its objective is to produce a *Dataset Requirements Concept Model* (DRCM). As in a traditional requirements engineering phase, the output artifact of the requirements modeling phase describes *what* are the constraints that the system should satisfy. In our method, systems under study are datasets in DL approaches. This model should be compliant with the *Metamodel* given as input artefact, see further Section 2.3.

In our approach, we concentrate on the *structural* requirements of datasets for DL-based systems, i.e., on what should be the datatypes for the dataset under development. The DRCM model describes the requirements of the structure to be satisfied by the dataset of the DL-based system under development. The structural requirements may be described using various modeling constructs, described in the further Section 2.3.2.

During this activity, the main stakeholders are: on the one hand the customer, and on the other hand an analyst from the provider's team. This activity may be realised in a number of ways. Typically requirements elicitation is performed through discussions among the stakeholders and results in the elicitation of the requirements model.

The content of the model under creation should describe the concepts related to the required structure of the dataset under development. More concretely the data, their type and attributes as well as properties that they should satisfy, need to be elicited. This activity is further described in Section 2.3.

2.2.3 Define a Formal Semantics

The second activity of our process is performed by a formal language expert. It aims at producing an executable formal requirements specification of the structural properties of the dataset under study. This activity takes as input the DRCM produced in the previous activity. The format of the FDRCM is tool-dependent, Section 2.3, below, gives insight on how we use the Alloy language (Jackson, 2012) for this purpose.

Using formal techniques brings precision on the definition of data structure and rigorous validation using mathematically defined semantics supported by automated tools. This results in a higher quality of requirements engineering.

2.2.4 Execute the Formal Specification

This activity consists in taking as input the FDRCM specified earlier and to *execute* it. By executing, we mean that the requirements should be fed to a formal engine allowing one to query the formal model. The aim of the queries is to provide enough data specification instances for the further validation activity. It may then either comfort the analyst in the current specification, or encourage the analyst to introduce changes in the model. There are two kinds of queries that we suggest to perform:

- *Exploration Queries.* Such queries result in a set of data specification instances. For instance, if `SevenEC` is the name of the equivalence class that should contain all images representing the digit seven, then an example of exploration query is to *"find possible data specification instances of a given DRCM satisfying the SevenEC equivalence class"*.
- *Verification Queries.* Such queries check that the dataset/data structure is coherent with some given property. For instance, here is a sample verification query: *"No data specification instance should satisfy properties of more than one equivalence class"*.

This activity should be performed by formal experts together with analysts. On the one hand, the role of the analyst is to identify the queries of best interest to improve the dataset requirements, and on the other hand the role of the formal expert is to make sure that the queries are well-formalised.

2.2.5 Validate the Dataset Requirements Concept Model

Its goal is the interpretation of the FDRCM executions, performed in the former activity, in order to conclude whether these executions are satisfactory, or not. If not satisfactory, then the requirements are not validated, and the process proceeds to redoing another iteration starting with the first activity *Modeling Dataset Structural Requirements*. If the analysis of the model execution results is satisfactory, then the DRCM is validated and our process ends here. The stakeholders may then continue with the succeeding dataset engineering activities.

2.3 Syntax and Semantics of the DRCM

In this subsection, we give some details on the language to be used for the modeling of the dataset structural requirements during the activities of our process. We follow a typical modeling approach in three-

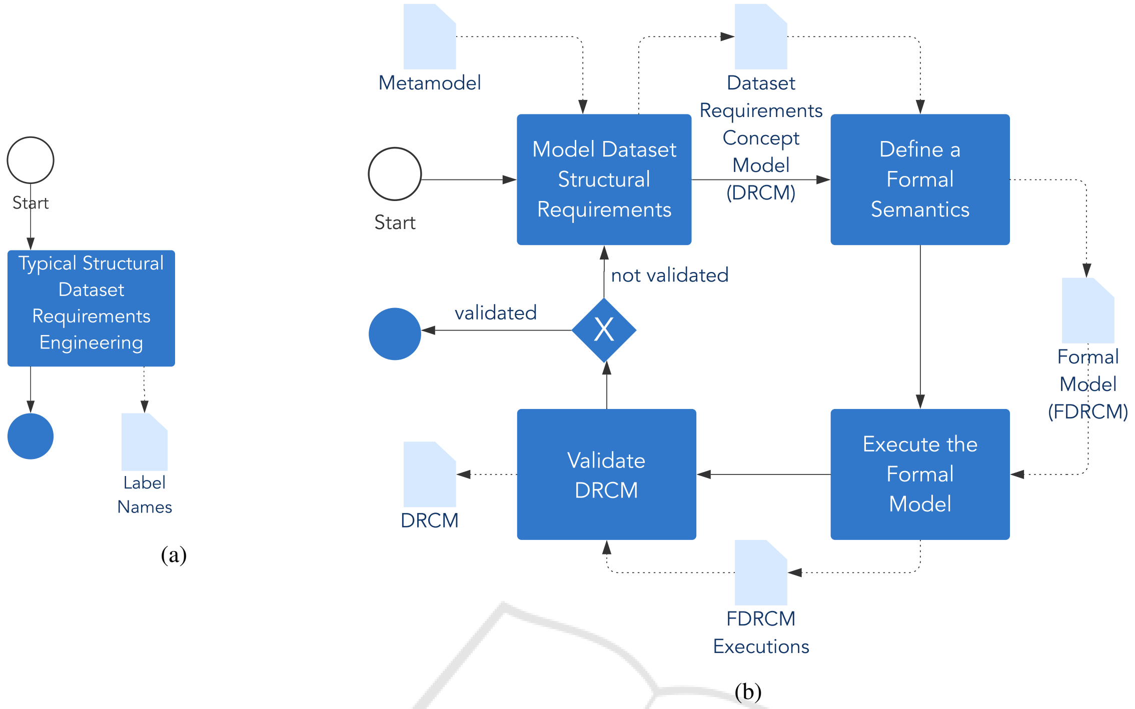

Start
Typical Structural Dataset Requirements Engineering
Label Names
(a)
Metamodel
Start
Model Dataset Structural Requirements
Dataset Requirements Concept Model (DRCM)
Define a Formal Semantics
not validated
validated
X
Formal Model (FDRCM)
DRCM
Validate DRCM
Execute the Formal Model
FDRCM Executions
(b)

Figure 1: A Typical Dataset Structural Requirements Engineering (a) in contrast with our Method (b).

levels, namely: meta-modeling, modeling and model execution. Firstly, we define a metamodel that describes the types of modeling elements usable for the modeling. Secondly, we present the scope of modeling in our approach, i.e., dataset structural requirements. And lastly we discuss the semantics of DRCM models allowing their consequent executions.

In our approach, we use the traditional standard semi-formal modeling language UML (Object Management Group, 2017) to support the modeling activity. It benefits from decades of industrial practice. In our context of improving the DL requirements engineering practices, we have chosen a small and restricted set of UML concepts to ease its transfer to non-Software Engineering practionners.

2.3.1 The DRCM Metamodel

A metamodel is a model defining the base elements of a modeling language (Object Management Group, 2017). We defined the metamodel using the Eclipse Modeling Framework, EMF (Steinberg et al.,), as an ecore metamodel. Figure 2 is a diagram representing graphically our DRCM metamodel with the EMF framework. In the following, we present shortly the main concepts of a DRCM model through its metamodel:

- *property* is a boolean characteristic of the dataset, data, or equivalence classes. It may be described by its name, signature (ie name and variable parameters), informal textual description, and/or formal expression in Alloy.
- *Invariant Property:* a property that is always true, it is a constraint on a DRCM model that must be satisfied.
- *Dataset* represents the set of data under study. It contains a set of properties and a set of invariants. A given DRCM focuses on the description of a *single* dataset.
- *Data:* is a structured element. It contains fields describing its structure. Fields are described as typed variables, with a name and a type. The type may either be a primitive type (Boolean, String, Int) or another Data defined in the same DRCM model. Data also contains a set of properties and a set of invariants.
- *Data Equivalence Class:* is a concept that represents a subset of the dataset's data and is characterised by a set of properties and invariant properties.

2.3.2 DRCM Modeling

In our approach, we use a subset of the base elements defined in the UML metamodel for class diagrams and follow the concrete graphical syntax of

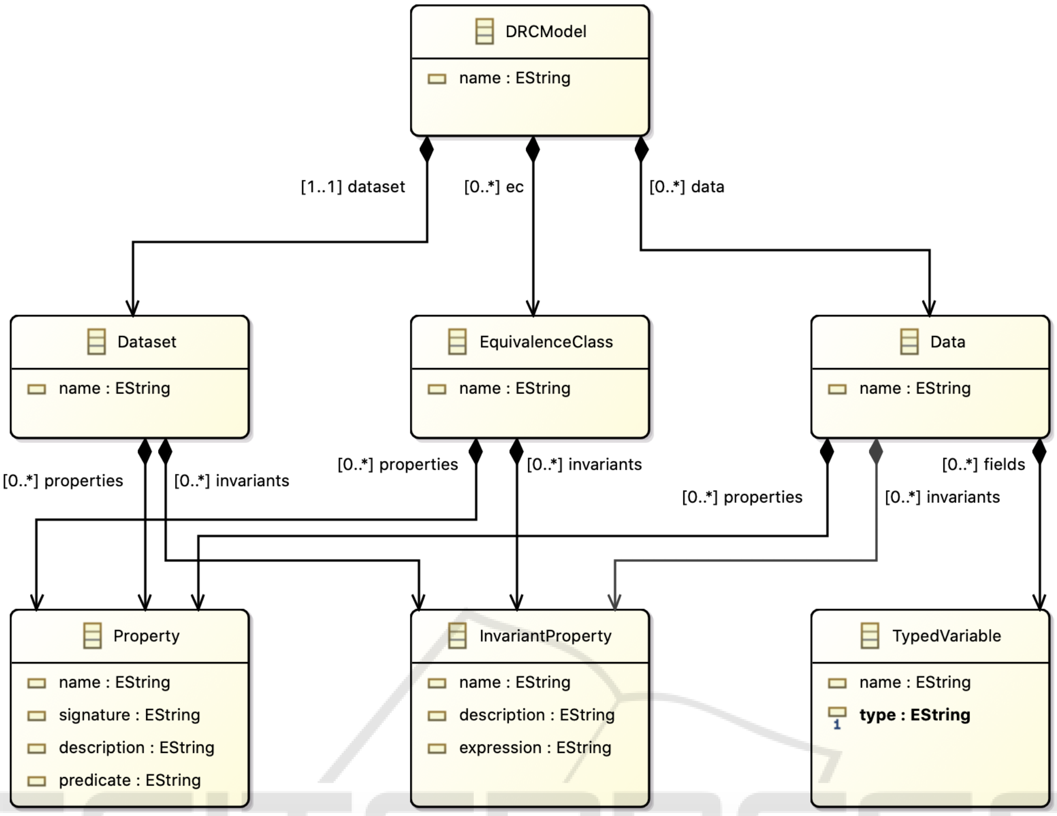


Figure 2: The DRCM Ecore Metamodel.

UML class diagrams. Based on the metamodel presented above, we describe in the following the dataset structural requirements concepts that may be modeled in a DRCM, their graphical syntax and map them to the corresponding UML elements. Each DRCM describes the requirements for a single dataset, a set of data and a set of equivalence classes.

- *property* is modeled as a *UML Operation* returning a boolean value. A property predicate is described as plain text between curly brackets `{}` after the operation signature. Note that the properties in our case study, as for instance, `isInOneEC` of data equivalence class `OneEC` in Figure 4, are described with Alloy language as a textual expression.
- *invariant property* is modeled as a *UML Operation* returning a boolean value and prefixed by a stereotype `<<inv>>`. As for properties, its expression may be described in curly brackets following the operation's signature.
- *Dataset*: is modelled as a *UML Class* with green header background. Its name is suffixed with `Dataset` and the class has a stereotype `<<dataset>>`. Properties and invariants may be described as *UML Operations*. See `FiveSegmentDigitsDataset` class in Figure 4 for an illustration.
- *Data* is modeled as a *UML Class* with blue header background. Its name is suffixed with `Data` and the class has a stereotype `<<data>>`. It possibly contains *UML Attributes* to describe its fields and boolean operations to describe its properties and invariants. (e.g. `FiveSegmentDigitData` and `Segment` classes in Figure 4).
- *Data equivalence classe* is modeled as a *UML Class* with red header background. Its name is suffixed with `EC` and the class has a stereotype `<<equivalence-class>>`. It contains boolean operations to describe its properties and invariants. (e.g. `ZeroEC` and `OneEC` classes in Figure 4).

2.3.3 Alloy Language-based Formal Specification Syntax and Semantics

In this subsection, we define the following operational semantics given as an informal translation from the DRCM concepts to Alloy language (Jackson, 2012) constructs.

- DRCM classes are specified as Alloy signatures.

- DRCM metamodel classes are defined as abstract signatures.
- DRCM class attributes are defined as fields in the related signature.
- The semantics of operations of classes depend on whether it has an `<<inv>>` stereotype or not.
  - DRCM class operations with an `<<inv>>` stereotype are considered to be invariant properties. We define the semantics of such operations as Alloy *signature facts*.
  - DRCM class operations *without* stereotype are considered to be properties to be checked in different contexts. As such, they are given the semantics of Alloy *predicates*.

2.4 MDE Toolset for Our Approach

The toolset supporting our approach is composed of four tools: a modeling language editor; a formal language editor; a formal execution engine; and an ad-hoc software application for the creation of synthetic data based on their specification instantiations. The source code of the toolset is available publicly (Ries, 2020).

2.4.1 DRCM Modeling Editor

To support the modeling of the DRCM requirements model, we implemented a graphical modeling editor based on the Sirius (Viyović et al., 2014) framework. The metamodeling concepts are defined in a `.ecore` file based on Section 2.3.1. The concrete modeling syntax is defined compliant with UML class diagram syntax with the help of the Sirius framework.

When a DRCM is modeled, our toolset allows to generate a FDRCM based on the information modeled in the DRCM. This generation is implemented with the XTend (Bettini, 2013) language well-suited for such Model2Text model transformations.

2.4.2 FDRCM Textual Editor

Alloy Analyzer (Jackson, 2012) allows one to write formal specifications in the Alloy language and offers typical textual language editor features, such as syntax highligthing and static analysis. We recommend using the Alloy Analyzer to complete the formal specification based on the FDRCM generated by our Model2Text model transformation.

2.4.3 FDRCM Execution

Kodkod is a SAT-based constraint solver and is used as a *model finder* (Torlak and Jackson, 2007) for specifications written in the Alloy language. It allows executing Alloy models and provide instances satisfying the given specification in the context of a given execution command. The Kodkod tool, now part of the Alloy Analyzer tool, allows one to parse Alloy specification and provide possible executions, i.e., model instances. This feature is particularly interesting in our context as it allows us to perform queries on our FDRCM, see Section 2.2.4.

2.4.4 Data Specification Requirements Visualiser

The objective of this tool is to show a representation of the data based on the specification instance generated by the Alloy formal engine. For instance, if the specification describes a dataset of images, this tool would create synthetic images based on the formal data specifications instances created by Kodkod. Indeed, and unfortunately, no generic tool would exist that generates the representation based on the specification, as this is tightly coupled with the expected concrete format of the data under specification. It is thus necessary to develop an ad-hoc tool for this need. In our toolset, we have implemented a Java program that performs the FDRCM executions by calling the Kodkod SAT-based constraint solver and interpreting the model execution to create possible visual representation of the data requirements specification instances.

3 CASE STUDY: STRUCTURAL REQUIREMENTS ENGINEERING OF A FIVE-SEGMENTS DIGITS DATASET

The aim of this section is to provide a proof-of-concept of our approach by conducting a step-by-step instantiation of our process. This instantiation is performed on the five-segments digits (FSD) case study that we define in this paper. The DL-based system, for which this dataset is defined, aims at recognising images of digits from 0 to 9. The case study is available publicly (Ries, 2020).

3.1 Modeling the Five-segments Digits DRCM

In this case study, a five-segments digit is characterised by at most five segments, more precisely three horizontal segments and two vertical segments, as shown with the sample zero and nine digits in Figure 3.

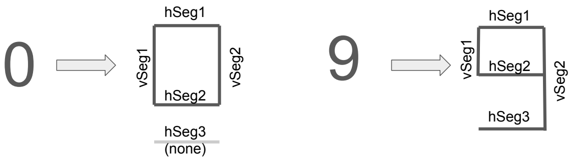


Figure 3: Samples of Five-Segments Digits (a zero and a nine).

In the first activity of our process, we elicit and model the initial structural requirements for the FSD-dataset. An extract of the resulting model of this activity is shown in Figure 4. This activity is composed of two tasks. The first task of this activity is to elicit the main elements structuring the data in this dataset. Here are some of the structural requirements that we define:

- Each data in the FSD-dataset shall be representing one digit, from 0 to 9, using at most five segments.
- The segments shall be line segments, either vertical or horizontal.
- There shall be at most three horizontal segments, named `hSeg1`, `hSeg2` and `hSeg3` and at most two vertical segments, named `vSeg1`, `vSeg2`.
- The segments shall be characterised by their size, and x-y position in an orthonormal 2D-space.
- All data should be distinct, i.e., no two data having the same segments
- All horizontal segments should be distinct, i.e., no two horizontal segments with the same x-y position and the same size.
- All vertical segments should also be distinct.
- There is no empty data, i.e., no data without segment.

The second task of this activity is the elicitation of the equivalence classes and their properties. In order to ease the specification of equivalence classes' properties, we defined a set of geometric property operations: `intersect`, `intersectT`, `isCornerTR`, etc. Let's describe one of them: `isCornerTR`.

- Property `isCornerTR[seg1,seg2:Segment]`[1] is true when the segment `seg1` ends [2] where the segment `seg2` starts.

Let's describe informally the `ZeroEC` equivalence class, modelled in Figure 4:

- `ZeroEC`: a digit is a zero-digit if it has two horizontal segments (`hSeg1`, `hSeg2`) and two vertical segments (`vSeg1`, `vSeg2`) such that the start of `vSeg1` coincides with the start of `hSeg1` (`isCornerTL[hSeg1,vSeg1]` is true), the end of `hSeg1` coincides with the start of `vSeg2` (`isCornerTR[hSeg1,vSeg2]` is true), the end of `vSeg2` coincides with the end of `hSeg2` (`isCornerBR[hSeg2,vSeg2]` is true) and the start of `hSeg2` coincides with the end of `vSeg1` (`isCornerBL [hSeg2,vSeg1]` is true).

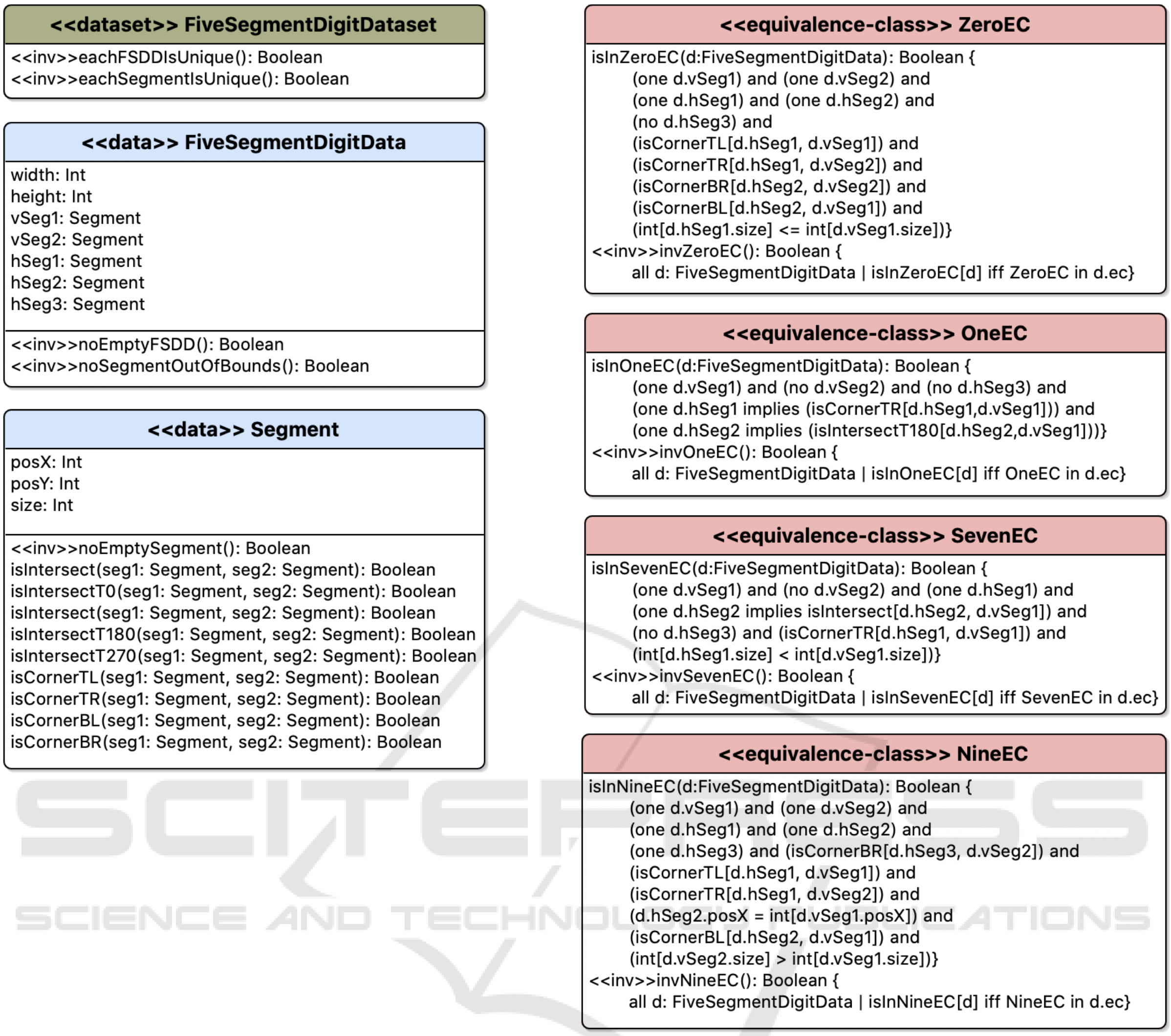
3.2 Specifying the Formal Requirements for the Five-segments Digits' DRCM

We, taking the role of a *formal expert*, specify in Alloy each requirement elicited formerly in the DRCM based on the FDRCM generated by our toolset. After completion of the specification, the Alloy FDRCM is as follows:

- Firstly, the metamodel concepts:
  - 3 abstract signatures for each of the 3 metamodel concepts: `Data`, `EquivalenceClass` and `Dataset`.
  - 3 relations between: `Data` and `EquivalenceClass` (named `ec`).
  - each metamodel properties, invariant or not, are defined at the level of the model.
- Then, the Alloy constructs for the specification of the model concepts specific to the FSD case study:
  - 1 signature `FiveSegmentDigitDataset` extending the `Data` signature.
  - 10 signatures extending the `EquivalenceClass` signature: `ZeroEc`, `OneEC`, `TwoEC`, `ThreeEC`, `FourEC`, `FiveEC`, `SixEC`, `SevenEC`, `EightEC`, `NineEC`.
  - each of the 10 signatures specify one predicate characterising the equivalence class of a digit.
  - 9 predicates for expressing the geometrical boolean operations.
- Finally, the Alloy invariants as 4 facts:
  - Specifying the metamodel's `eachDataIsUnique` invariant implied writing two invariants at the model-level: `eachFSDDIsUnique` and `eachSegIsUnique`.
  - Similarly for the metamodel's invariant `noEmptyData`, which resulted in the specification of `noEmptyFSDD` and `noEmptySegment`.

[1] seg1 is assumed to be an horizontal segment and seg2 a vertical segment.

[2] As a convention, the start, resp. the end, of an horizontal segment is the point with the lowest x-value, resp. the highest x-value. Similarly, the start, resp. the end, of a vertical segment is the point with the highest y-value, resp. lowest y-value.



```
<<dataset>> FiveSegmentDigitDataset
<<inv>>eachFSDDIsUnique(): Boolean
<<inv>>eachSegmentIsUnique(): Boolean

<<data>> FiveSegmentDigitData
width: Int
height: Int
vSeg1: Segment
vSeg2: Segment
hSeg1: Segment
hSeg2: Segment
hSeg3: Segment
<<inv>>noEmptyFSDD(): Boolean
<<inv>>noSegmentOutOfBounds(): Boolean

<<data>> Segment
posX: Int
posY: Int
size: Int
<<inv>>noEmptySegment(): Boolean
isIntersect(seg1: Segment, seg2: Segment): Boolean
isIntersectT0(seg1: Segment, seg2: Segment): Boolean
isIntersect(seg1: Segment, seg2: Segment): Boolean
isIntersectT180(seg1: Segment, seg2: Segment): Boolean
isIntersectT270(seg1: Segment, seg2: Segment): Boolean
isCornerTL(seg1: Segment, seg2: Segment): Boolean
isCornerTR(seg1: Segment, seg2: Segment): Boolean
isCornerBL(seg1: Segment, seg2: Segment): Boolean
isCornerBR(seg1: Segment, seg2: Segment): Boolean

<<equivalence-class>> ZeroEC
isInZeroEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (one d.vSeg2) and
      (one d.hSeg1) and (one d.hSeg2) and
      (no d.hSeg3) and
      (isCornerTL[d.hSeg1, d.vSeg1]) and
      (isCornerTR[d.hSeg1, d.vSeg2]) and
      (isCornerBR[d.hSeg2, d.vSeg2]) and
      (isCornerBL[d.hSeg2, d.vSeg1]) and
      (int[d.hSeg1.size] <= int[d.vSeg1.size])}
<<inv>>invZeroEC(): Boolean {
      all d: FiveSegmentDigitData | isInZeroEC[d] iff ZeroEC in d.ec}

<<equivalence-class>> OneEC
isInOneEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (no d.vSeg2) and (no d.hSeg3) and
      (one d.hSeg1 implies (isCornerTR[d.hSeg1,d.vSeg1])) and
      (one d.hSeg2 implies (isIntersectT180[d.hSeg2,d.vSeg1]))}
<<inv>>invOneEC(): Boolean {
      all d: FiveSegmentDigitData | isInOneEC[d] iff OneEC in d.ec}

<<equivalence-class>> SevenEC
isInSevenEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (no d.vSeg2) and (one d.hSeg1) and
      (one d.hSeg2 implies isIntersect[d.hSeg2, d.vSeg1]) and
      (no d.hSeg3) and (isCornerTR[d.hSeg1, d.vSeg1]) and
      (int[d.hSeg1.size] < int[d.vSeg1.size])}
<<inv>>invSevenEC(): Boolean {
      all d: FiveSegmentDigitData | isInSevenEC[d] iff SevenEC in d.ec}

<<equivalence-class>> NineEC
isInNineEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (one d.vSeg2) and
      (one d.hSeg1) and (one d.hSeg2) and
      (one d.hSeg3) and (isCornerBR[d.hSeg3, d.vSeg2]) and
      (isCornerTL[d.hSeg1, d.vSeg1]) and
      (isCornerTR[d.hSeg1, d.vSeg2]) and
      (d.hSeg2.posX = int[d.vSeg1.posX]) and
      (isCornerBL[d.hSeg2, d.vSeg1]) and
      (int[d.vSeg2.size] > int[d.vSeg1.size])}
<<inv>>invNineEC(): Boolean {
      all d: FiveSegmentDigitData | isInNineEC[d] iff NineEC in d.ec}
```

Figure 4: Case Study: Extract of the Dataset Requirements Concept Model Resulting from the First Iteration of Our Process.

3.3 Executing the Formal Specifications

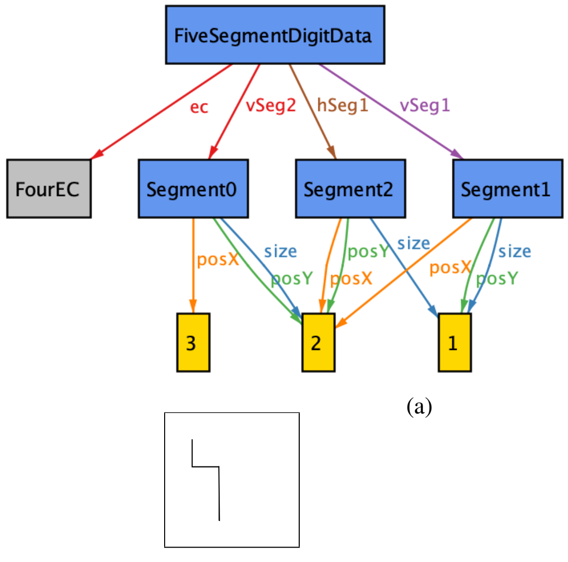
In this activity, we define two Alloy commands to run two different executions of the DRCM model.

The objective of the first execution is to explore the possible data specification instances. We specify an Alloy command to instantiate all digits of size 5x5. Figure 5a shows one of the instance generated from the execution of this command. In this figure, the model execution corresponds to a specification of a digit with 3 segments: one horizontal segment starting in position (2,2) of size 1, one vertical segment starting in position (2,1) of size 1 and a second vertical segment starting in position (3,2) of size 2. This instance satisfies the property of the `FourEC` equivalence class.

Thanks to the Alloy Analyzer tool API and the Java program that we develop to support this approach, the generated Alloy specification is interpreted and a file is created to represent the five-segment digit specification instance as an image, see Figure 5b.

In this execution, we generate, more than 400 different digit specifications. Figure 6 shows a visual interpretation of an extract of these generated digits.

We also perform a second model execution with the objective to check that there is no data satisfying the properties of more than one equivalence class. It returns a non-empty set of digits that satisfy two equivalence classes: `OneEC` and `SevenEC`. Figure 7 shows the interpreted results of this second Alloy model execution.



(a)

(b)

Figure 5: An Alloy model execution (a) and its visual representation (b).

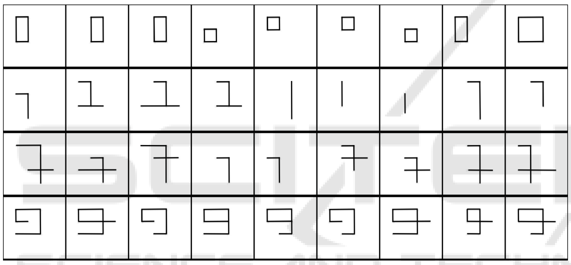


Figure 6: Extract of the interpreted digits instantiated by the exploration query.

3.4 Validating the DRCM

In this last step of our process, we analyse the FDRCM execution in order to evaluate if the DRCM is validated, or not. Here are the three points concluding the analysis:

1. We are satisfied by the model executions concerning the characterisation of the zero-digit data structure, thus we validate the DRCM related to the zero-digit equivalence class.

2. The verification execution is not satisfactory. Indeed, in this case study, we expect that the relation between digit instances and equivalence class is unambiguous by relating a digit data with *exactly* one equivalence class. Thus this execution exhibits an issue in our DRCM, namely the fact that several specification instances satisfy the properties of both one-digit's equivalence class (`OneEC`) and seven-digit's equivalence class (`SevenEC`).

3. We wish to improve the DRCM such that nine-digits without the bottom horizontal segment should also be part of the dataset requirements.

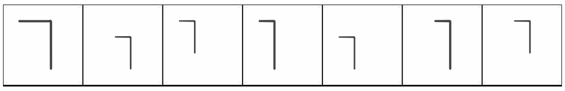


Figure 7: Digits corresponding to specifications in more than one equivalence class.

In the end, we conclude that the DRCM of this first iteration is *not validated* due to the two identified issues in points 2. and 3. above. Thus there is a need to perform a second iteration of the process.

3.5 Second Iteration of the Process

3.5.1 Updating the DRCM and the FDRCM

In the second iteration, we update the DRCM and the FDRCM. We introduce modifications on the property descriptions of:

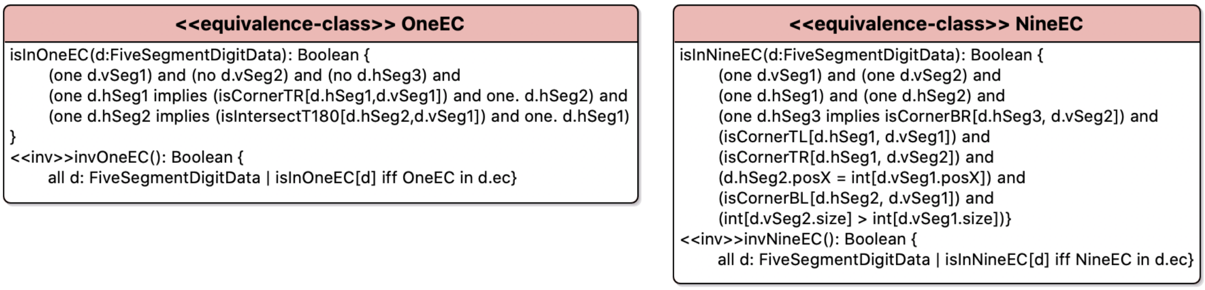
- the `NineEC` in which we state that the third horizontal segment is optional with a logical `implies` statement.
- the `OneEC`. There should be now either zero horizontal segments or two horizontal segments, but never a single horizontal segment such that the ones are not mistaken for sevens.

3.5.2 Executing the Updated FDRCM and Validation of the Updated DRCM

We define a first Alloy command and run it to explore the new nine-digits specification. Figure 9 shows an extract of some of the new nine digits resulting from the change in the DRCM's model property `isInNineEC` of the `NineEC` class. A second command is defined and we execute it to confirm that there is no digit satisfying the properties of two equivalence classes at the same time. In particular, the one-digits and the seven-digits are now partitioned. Thus in the end of this second iteration, we validate the DRCM and end the process.

Consequently, we may then go on with the further activity in the dataset development process, namely dataset construction, which is out of the scope in this paper. Among possible activities, this could typically involve: data labelling, data preparation, data synthesis, etc.

As a conclusion, we improved the quality of the dataset by using our method and tool. Firstly by providing a model of the dataset structural requirements. Secondly our method allowed us to detect the issue with ambiguous ones and sevens, as well as allowing us to explore a different possible kind of nines. Our



```
<<equivalence-class>> OneEC
isInOneEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (no d.vSeg2) and (no d.hSeg3) and
      (one d.hSeg1 implies (isCornerTR[d.hSeg1,d.vSeg1]) and one. d.hSeg2) and
      (one d.hSeg2 implies (isIntersectT180[d.hSeg2,d.vSeg1]) and one. d.hSeg1)
}
<<inv>>invOneEC(): Boolean {
      all d: FiveSegmentDigitData | isInOneEC[d] iff OneEC in d.ec}
```

```
<<equivalence-class>> NineEC
isInNineEC(d:FiveSegmentDigitData): Boolean {
      (one d.vSeg1) and (one d.vSeg2) and
      (one d.hSeg1) and (one d.hSeg2) and
      (one d.hSeg3 implies isCornerBR[d.hSeg3, d.vSeg2]) and
      (isCornerTL[d.hSeg1, d.vSeg1]) and
      (isCornerTR[d.hSeg1, d.vSeg2]) and
      (d.hSeg2.posX = int[d.vSeg1.posX]) and
      (isCornerBL[d.hSeg2, d.vSeg1]) and
      (int[d.vSeg2.size] > int[d.vSeg1.size])}
<<inv>>invNineEC(): Boolean {
      all d: FiveSegmentDigitData | isInNineEC[d] iff NineEC in d.ec}
```

Figure 8: DRCM Changes in Classes OneEC and NineEC.

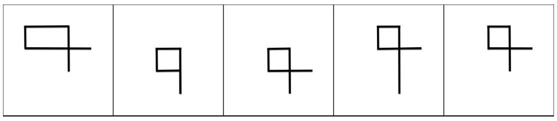


Figure 9: Explored Model Executions of Nines Without 3rd Horizontal Segment.

method helped detect these issues earlier in the DL-based system development process.

4 DISCUSSION

In this section, we discuss shortly about some important aspects of this paper. The first aspect is the scalability of our method. In this paper, we present our model-driven engineering method for requirements of datasets and we validate it experimentally by applying it to the five-segments digits case study. Our case study is of rather low combinatory complexity. Due to this low complexity, we could specify its requirements models: DRCM and FDRCM at a low-level of abstraction, with precise details on the data structure, i.e., horizontal/vertical segments and their size/position. We could also execute all model executions due to the low number of combinations, i.e., around 400 possible data instances. Let's note that it is rarely possible to be able to perform all possible model executions. So first of all, it is most important that there should be enough model executions to either comfort the analyst and the customers that their requirements model is valid; or to exhibit some requirements to be improved thanks to our so-called exploration or validation queries. Secondly, requirements models allow to describe conceptual domains at a *chosen level of abstraction*. Thus, as for any modeling methods, it must be decided what is the right level of abstraction adapted to the project under development. Indeed, when using our method for a larger-scale case study, the level of abstraction should consequently be higher in order to be tractable. This is especially true for formal methods whose interest is not in being applicable for a full real system but in its contribution to increase the quality of the system under development.

The second aspect that we would like to discuss is the toolset limitation. The presented toolset is a prototype implementation, intended to make a proof-of-concept in terms of tool-support for our method. Some parts of the toolset must indeed be improved before being transfered to an industrial context, or used at a larger scale. For instance, some limitations include, the fact that the different textual editors are the default ones provided by EMF, thus there is no syntax checking, nor syntax highlighting on the textual expressions written in Alloy, nor rich text editing on the properties and invariants descriptions.

5 RELATED WORK

Only few works are available on requirements engineering for DL-based systems. The first one by Hill et al. (Hill et al., 2016) presents a process comprising nine stages for a machine learning workflow, but without going into the details of the first stage, *model requirements*. The second one by Amershi et al. (Amershi et al., 2019) present a general SE process for ML-based systems. Their process includes a requirements engineering phase that considers the system in the large without giving particular concerns about the requirements of the dataset.

Industry also contributes to this area of ML-based systems development. As for instance, Google describes a ML Workflow (Google, 2020), but the first activity is *Source and Prepare your data* and the requirements engineering is basically skipped. Microsoft defines the TDSP (TDSP, 2020; Mathew et al., 2018) process, standing for *Team Data Science Process*. This process is composed of a lifecycle starting with a Problem Definition phase, with an activity *Data characteristics questions*. Unfortunately, these two related works lack support for semi-formal and formal modeling of dataset structural requirements.

In the MDE community, common interest in MDE and AI is also quite recent, a first workshop on AI and MDE was held in 2019 (Burgueño et al., 2019).

Regarding model execution, a large number of related works are available, published in the last two decades in the literature, around the formalisation and execution of UML models (Object Management Group, 2018; Shah et al., 2009; Mellor et al., 2002), but none of these approaches tackle specifically the modeling of datasets requirements for DL-based systems.

Lastly, a number of approaches, mainly in the AI community, e.g. DeepXplore (Pei et al., 2017), Scenic (Fremont et al., 2019), DeepTest (Tian et al., 2018), CARLA (Dosovitskiy et al., 2017), have been published on designing and generating datasets for DL-based systems. Whereas in our paper, we provide an iterative method involving the customer and SE/DS analysts focused on the production of a *requirements* specification model, i.e., our method is not about designing, nor producing datasets.

6 CONCLUSION

In this paper, we contributed to the introduction of software engineering rigor in the engineering of datasets for DL-based systems. We presented our model-driven engineering method composed of a process defined with BPMN and supported by a toolset. Our Dataset Requirements Concept Model (DRCM) is modeled in compliance with UML syntax. We presented a DRCM graphical editor implemented EMF and Sirius that allows creating diagrams to ease the requirements elicitation discussions. The DRCM model is executable with the Alloy Analyzer tool, thanks to our informal definition of the DRCM operational semantics as a translation from UML to Alloy. Our Xtend implementation allows us to generate a FDRCM skeleton code in Alloy. Lastly, we presented an instantiation of our process in a case study related to the requirements engineering of a five-segments digits dataset and discuss some current issues.

7 FUTURE WORK

In this paper, we present a method for improving DL dataset requirements engineering, which has raised a number of interesting future works.

In order to increase the MDE dimension of our approach, in our method, the definition of the translational semantics of DRCM models will firstly be improved using an Alloy metamodel. Together with the definition of Model-2-Model transformations from DRCM concepts to Alloy concepts. Secondly, we plan to define a textual domain-specific language (DSL) to allow specifying the DRCM textually. The textual and graphical DRCM will be synchronised such that changes in diagrams update the textual counterpart and vice-versa.

Some other future works are related to the rigorous validation of our method. For this, we plan to conduct additional experimentations, with well-known examples like MNIST (Yann et al., 2018). These experimentations will serve for comparative evaluation of our method against existing methods on common case studies. Moreover, we will apply our method on a larger-case study, in the area of earth's ecosystems health monitoring. This latter case study will investigate scalability, practicality, and possibly adaptation, of our method for potential transfer to larger examples.

Lastly, some final future works are related to the software engineering aspects of our approach. We will improve our approach in two ways. Firstly, we will broaden the work on the dataset requirements engineering phase by including other types of functional and non-functional requirements. Moreover this phase should be placed in the more general context of DL-based software requirements. How do the dataset requirements involve (or are involved in) the DL-based software requirements ? Secondly, we will concentrate on the definiton of the succeeding dataset engineering lifecycle phases, e.g., dataset design, dataset production, dataset assessment, dataset deployment.

REFERENCES

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., and Zimmermann, T. (2019). Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, Canada.

Bettini, L. (2013). *Implementing Domain-Specific Languages with Xtext and Xtend: Learn How to Implement a DSL with Xtext and Xtend Using Easy-to-Understand Examples and Best Practices*. Community Experience Distilled. Packt Publ, Birmingham.

Burgueño, L., Burdusel, A., Gérard, S., and Wimmer, M. (2019). Preface to MDE Intelligence: 1st Workshop on Artificial Intelligence and Model-Driven Engineering. In *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and

Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. *arXiv:1711.03938 [cs]*.

Fremont, D. J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A. L., and Seshia, S. A. (2019). Scenic: A Language for Scenario Specification and Scene Generation. *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2019*, pages 63–78.

Google (Last visited on Sept. 2020). Machine learning workflow. https://cloud.google.com/ai-platform/docs/ml-solutions-overview.

Hill, C., Bellamy, R., Erickson, T., and Burnett, M. (2016). Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 162–170, Cambridge. IEEE.

Jackson, D. (2012). *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.

Jahic, B., Guelfi, N., and Ries, B. (2019). Software Engineering for Dataset Augmentation using Generative Adversarial Networks. In *Proceedings of the 10th IEEE International Conference on Software Engineering and Service Science (ICSESS 2019)*, Beijing, China. IEEE.

Jahic, B., Guelfi, N., and Ries, B. (2020). Specifying Key-Properties to Improve the Recognition Skills of Neural Networks. In *Proc. of the 2020 European Symposium on Software Engineering*, Roma, Italy. ACM.

Khomh, F., Adams, B., Cheng, J., Fokaefs, M., and Antoniol, G. (2018). Software Engineering for Machine-Learning Applications: The Road Ahead. *IEEE Software*, 35(5):81–84.

Mathew, S., Danielle, D., and Tok, W. H. (2018). *Deep Learning with Azure - Building and Deploying Artificial Intelligence Solutions on the Microsoft AI Platform*. Apress.

Mellor, S. J., Balcer, M., and Jacoboson, I. (2002). *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., USA.

Naur, P. and Randell, B. (1969). Software Engineering Report of a conference sponsored by the NATO Science Committee Garmisch Germany 7th-11th October 1968.

Object Management Group (2011). Business Process Model and Notation (BPMN) v2.0. OMG Standard Full Specification formal/2011-01-03.

Object Management Group (2017). Unified Modeling Language: Superstructure (UML), v. 2.5.1. OMG Standard Full Specification formal/17-12-05.

Object Management Group (2018). Semantics of a Foundational Subset for Executable UML Models (fUML), v. 1.4. OMG Standard Full Specification formal/18-12-01.

Pei, K., Cao, Y., Yang, J., and Jana, S. (2017). DeepXplore: Automated Whitebox Testing of Deep Learning Systems. *Proceedings of the 26th Symposium on Operating Systems Principles*.

Ries, B. (2009). *SESAME - A Model-Driven Process for the Test Selection of Small-Size Safety-Related Embedded Software*. PhD thesis, University of Luxembourg, Luxembourg.

Ries, B. (2020). DRCM Editor and FSDD Case Study. https://doi.org/10.5281/zenodo.4020938.

Shah, S. M. A., Anastasakis, K., and Bordbar, B. (2009). From UML to Alloy and back again. In *Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation - MoDeVVa '09*, pages 1–10, Denver, Colorado. ACM Press.

Sommerville, I. (2016). *Software Engineering*. Pearson, tenth edition edition.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. Eclipse Modeling Framework Second Edition. page 14.

TDSP (Last visited on Sept. 2020). The Team Data Science Process. https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/.

Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars.

Torlak, E. and Jackson, D. (2007). Kodkod: A Relational Model Finder. In Grumberg, O. and Huth, M., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424, pages 632–647. Springer Berlin Heidelberg, Berlin, Heidelberg.

Viyović, V., Maksimović, M., and Perisić, B. (2014). Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*.

Vogelsang, A. and Borg, M. (2019). Requirements Engineering for Machine Learning: Perspectives from Data Scientists. *arXiv:1908.04674 [cs]*.

Yann, L. C., Corinna, C., and Christopher, B. (2018). THE MNIST DATABASE of handwritten digits.