UML Consistency Rules: a Case Study with Open-Source UML Models

Damiano Torre
damiano.torre@uni.lu
University of Luxembourg
Luxembourg

Yvan Labiche
yvan.labiche@carleton.ca
Carleton University
Ottawa, Canada

Marcela Genero
marcela.genero@uclm.es
University of Castilla-La Mancha
Ciudad Real, Spain

Maged Elaasar
melaasar@gmail.com
Carleton University
Ottawa, Canada

Claudio Menghi
claudio.menghi@uni.lu
University of Luxembourg
Luxembourg

ABSTRACT

UML models are standard artifacts used by software engineers for designing software. As software is designed, different UML diagram types (e.g., class diagrams and sequence diagrams) are produced by software designers. Since the various UML diagram types describe different aspects of a software system, they are not independent but strongly depend on each other, hence they must be consistent. Inconsistencies cause faults in the final software systems. It is, therefore, paramount that they get detected, analyzed, and fixed. Consistency rules are a useful tool proposed in the literature to detect inconsistencies. They categorize constraints that help in identifying inconsistencies when violated. This case study aims at collecting and analyzing UML models with OCL consistency rules proposed in the literature and at promoting the development of a reference benchmark that can be reused by the (FM-)research community. We collected 33 UML consistency rules and 206 different UML diagrams contained in 34 open-source UML models presented in the literature. We propose an FM-based encoding of the consistency rules in OCL. This encoding allows analyzing whether the consistency rules are satisfied or violated within the 34 UML models. To assess the proposed benchmark, we analyzed how the UML models, consistency rules, diagram types contained in the benchmark help in assessing the consistency of UML models, and the consistency of diagrams across the different software development phases. Our results show that the considered UML models and consistency rules allowed identifying 2731 inconsistencies and that those inconsistencies refer to different software development phases. We concluded that the considered UML models and consistency rules could be considered as an initial benchmark that can be further extended by the research community.

KEYWORDS

Case Study, UML consistency rules, Benchmark, OCL, Open-source models, Model Checking

ACM Reference Format:
Damiano Torre, Yvan Labiche, Marcela Genero, Maged Elaasar, and Claudio Menghi. 2020. UML Consistency Rules: a Case Study with Open-Source UML Models. In *8th International Conference on Formal Methods in Software Engineering (FormaliSE '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3372020.3391554

1 INTRODUCTION

Model-Driven Software Engineering (MDSE) [17] promotes the usage of models starting from requirement analysis to design, implementation, and finally to software deployment [25]. MDSE has lately received considerable attention in academia and industry [10], which has resulted in models gaining even more importance in software development. The Unified Modeling Language (UML) [18] is the de facto standard modeling language for object-oriented design [21]. It is also frequently the chosen modeling language when implementing MDSE. UML provides 14 diagram types [18] that can be used to describe a system from different perspectives (e.g., structure, behavior) and/or abstraction levels (e.g., requirements elicitation, system design). This helps to deal with the complexity of a system specification and distributing the responsibilities between different stakeholders, among other benefits.

Since the various UML diagram types describe different aspects of a software system, they are not independent but strongly depend on each other, hence they must be consistent [28]. *Consistency rules* are useful and powerful tools proposed in the literature that allow detecting inconsistencies across UML diagrams. Using consistency rules on models is a well-spread practice used across several different domains, such as safety-critical systems [20], robotics [22] or legal compliance [31]. In an attempt to obtain an accurate picture of the current research in the area of UML consistency, we initiated a long-term investigation [26] that started with a Systematic Mapping Study (SMS) of the UML consistency rules discussed in literature [28]. This SMS allowed us to make several observations: (1) practitioners who rely on consistency rules do so for a variety of reasons (e.g., code generation, testing); (2) they rely on various sets of consistency rules, involving varied UML digram types; (3) they typically rely on very similar, sometimes identical, sets of consistency rules; and (4) 116 rules, which are not documented in the UML standard, are relevant for practitioners [30]. To further assess and compare current usages of consistency rules, we tried to classify those 116 rules (e.g., identify what the context-dependent and the general-purpose rules are), and identify the set of UML consistency rules that should always be enforced. This has been performed by interviewing experts in the field of MDSE from academia and industry [29]. Among other results [29], we identified a subset of 52 general-purpose rules out of the 116 UML consistency rules, which the MDSE experts considered should always be enforced in UML models.

Formal methods (FM) techniques are widely used in the literature to check that UML models are compliant with a given set of consistency rules [4]. A common practice is to use the Object Constraint Language (OCL) [19] to specify those rules and to use an appropriate solver to check them. After a solver is modified, or a new solver is proposed, FM experts usually have to check for the presence of bugs and assess performances. A frequent problem with this practice is related to the absence of existing benchmarks that contain a sufficiently large set of consistency rules and UML models to be verified.

This case study aims at collecting and analyzing UML models with OCL consistency rules that were proposed in the literature and at promoting the development of a reference benchmark that can be reused by the research community. To reach this goal, we (1) collected a relatively large set of 34 open-source UML models. Those UML models contained in a total set of 206 different UML diagram types; (2) considered 116 UML consistency rules proposed in the literature [30] and formalized 52 of them using the Object Constraint Language (OCL); (3) imported the UML models and the OCL rules into the Eclipse Papyrus modeling tool[1]; (4) analyzed the satisfaction of the OCL rules and recorded the obtained results; and (5) performed an in-depth analysis of the results to assess the characteristics of the proposed benchmark. In total, we identified 2731 inconsistencies over the considered 34 UML models.

To assess the characteristics of the proposed benchmark we analyzed how useful the UML models of the proposed benchmark are in assessing consistency rules (**RQ1**), how useful the consistency rules of the proposed benchmark are in assessing the consistency of UML models (**RQ2**), how useful the different UML diagram types of the proposed benchmark are in assessing consistency rules (**RQ3**), and how useful the different UML models of the proposed benchmark are in assessing the satisfaction of consistency rules UML in different software development phases (**RQ4**).

Our results show that the UML models of the proposed benchmark contain a variable number of inconsistencies that are sufficient for allowing an effective assessment of the consistency rules. The number of inconsistencies associated with the different consistency rules is variable. This indicates that the proposed benchmark contains consistency rules that help to assess the consistency of UML models to different degrees. Except for Object diagrams (OD) and Composite Structure diagrams (CSD), which contained a small number of inconsistencies, the other UML diagram types are useful for assessing consistency rules. Finally, the UML models of the proposed benchmark contain models that show examples of inconsistencies in different development phases. Those results show that the considered UML models and OCL consistency rules can be considered as an initial benchmark that can be further extended by the research community.

Organization. Section 2 describes the set of UML models involved in this work. Section 3 shows how consistency rules have been collected and formalized in OCL. Section 4 presents how the OCL rules were validated on the considered UML models. Section 5 evaluates the characteristics of the proposed benchmark. Section 6 explains the threats to the validity of this case study. Section 7 discusses our findings. Section 8 provides a summary of the related work. Section 9 concludes this work.

2 UML MODELS

UML is not tied to a specific software development method [2]. The diagrams types provided by UML are designed to effectively support an iterative and incremental process [1]. Several tools have been proposed in the literature to support users in developing UML models in practical applications. In this work, we are considering Eclipse Papyrus because it is an industrial-grade open source Model-Based Engineering tool that has been used successfully in industrial and academic projects [2]. According to Ed Seidewitz [24], Papyrus is the only 100% conformant implementation of the standard UML, and it allows developing the complete set of UML diagrams. In Table 1, we report the diagrams involved in this study. In the following subsections, we describe the five steps we carried out to identify the UML models used for this study.

Step 1. Due to the current lack of shareable, industrial UML models, we considered three open-source UML repositories to identify our UML models: the UML Repository [3] (R1), the Repository for Model-Driven Development (ReMoDD) [4] (R2) and the Lindholmen Dataset [5] [14] (R3). These three repositories aim at supporting researchers and practitioners of the Model-Driven Development community. We downloaded all the UML models from the three repositories' websites. R1, R2, and R3 contained respectively 32, 4, and 5974 models, leading to a total of 6010 UML models.

Step 2. Since the UML models available on these three repositories were developed with different UML tools, we filtered only UML models that were developed by the Eclipse Papyrus tool. This leads to a set of 276 UML Papyrus models.

Step 3. We manually analyzed the 276 models and excluded 242 of them for the following reasons:

- 193 did not contain any diagram that involved at least a total of ten of the following model elements: *classes* (for Class (CD), Object (OD), and Composite Structure (CSD) diagrams), *interactions* (for Communication (COMD), Sequence (SD) and Interaction (ID) diagrams), *actions and decisions* (for Activity diagram (AD)), *states* (for State Machine diagram (SMD)), and *use cases* (for Use Case diagram (UCD)). As

[1]https://www.eclipse.org/papyrus/download.html

[2]https://www.eclipse.org/papyrus
[3]http://models-db.com/
[4]http://www.remodd.org/
[5]http://oss.models-db.com

such, we considered those models not representative of real applications and any analysis meaningless;
- Ten models were duplicates;
- 15 were corrupt files and could not be loaded in Eclipse Papyrus;
- 14 were UML profiles, which are not the type of UML models targeted in this work;
- Seven were empty models;
- Three models only contained diagrams (i.e., Component, Package, Deployment diagrams) that are not covered by our 33 consistency rules presented in Table 2. Those rules are discussed in Section 3.

The 34 UML Papyrus models returned by Step 3 focus on different software domains (for instance, aerospace, logistics, gaming, electronics, service, and robotics).

Step 4. For each UML model we:

- assigned a unique ID;
- collected Name, online link, and number of diagrams for each UML models;
- The type of the UML diagram type involved (UDI) in each UML model project;
- The model elements involved in the UML diagrams: classes, interactions , actions and decisions, states, and use cases.

We finally identified 206 UML diagrams in the different UML models as reported in Table 1 (column Number). We did not find any of the following UML diagrams: COMD, CSD and ID. The 206 UML diagrams present a total of 1722 "model elements" including 898 classes (CD, OD, and CSD), 369 actions and decisions (AD), 341 interactions (SD), 57 use cases (UCD), and 59 states (SMD).

Step 5. As we aim at evaluating how the benchmark allows assessing consistency rules in different Software Development Phases (SDP) (see Section 5.4), we considered the models related to the five object-oriented software development phases discussed by Dathan and Ramnath [6]. We chose this SDP because it is suggested [6] to be used with the most recent stable version of UML, 2.5. The models related to these phases are detailed in the following:

- *Requirements models* (ReqM): describe the boundary and interaction between the system and users. Those models include Use Case diagrams (UCD);
- *Interaction models* (IntM): describe how objects in the system will interact. These models include Interaction (ID), Communication (COMD) and Sequence (SD) diagrams;
- *Dynamic models* (DynM): define how the interaction occurrences come together in a definition of the system and are, therefore, a part of the dynamic model of the system. These models include State Machines (SMD) and Activity (AD) diagrams;
- *Logical models* (LogM): describes the classes and objects that will make up the system. These models include Class (CD), Composite Structure (CSD), and Object (OD) diagrams;
- *Deployment model*: describes the physical architecture and the deployment of components. These models include Deployment diagrams. This phase and UML diagram type are not considered in this study because out of the scope.

Table 1: UML Diagram Types, Software Development Phases (SDP), and Number of Consistency Rules (see Table 2) that refer to each Diagram Type.

Diagram Type	Acronym	Number	SDP	N Rules
Class	CD	82	LogM	19
Use Case	UCD	12	ReqM	2
Communication	COMD	0	IntM	3
Composite Structure	CSD	0	LogM	4
State Machine	SMD	12	DynM	4
Sequence	SD	27	IntM	11
Object	OD	19	LogM	1
Activity	AD	54	DynM	3
Interaction	ID	0	IntM	1
Total		**206**		**48**

The software development phases associated with each model are also reported in Table 1 (column SDP).

3 CONSISTENCY RULES

In the next two subsections we present: (1) the UML consistency rules considered in this work (Section 3.1), and (2) how their OCL encoding was carried out (Section 3.2).

3.1 Collecting the Consistency Rules

We considered 52 of the 116 UML consistency rules recently discussed by Torre et al. [29], since according to MDSE experts from academia and the industry, those rules should always be enforced in UML models [29]. Those rules involve syntactic checks (including type checks) and semantic checks, as discussed in Torre et al. [29]. Ten out of these 52 OCL rules were obtained from other authors who presented OCL rules, in particular, rules R112, R55, and R110 were also considered by Reder and Egyed [23], rules R101, R105, R102, and R106, were also considered by Dragomir and Ober [8], and rules R84, R74, and R75 were also considered by Briand et al. [3].

Out of these 52 rules, the 33 that are considered in this work (for the reason that will be explained in Section 3.2) are presented in Table 2. The first column contains an identifier that is associated with each rule. The second column contains a textual description of the rules. The last column indicates which of the UML diagrams presented in Table 1 is involved in each rule. Note that 18 consistency rules involve only one UML diagram, 15 consistency rules involve two different UML diagrams, and no consistency rule involves more than two diagrams. Furthermore, in Table 1, we report the number of the rules considered in this paper that constraint each of the diagram types. For example, the class diagrams (CD) are involved in 19 consistency rules.

Table 2: List of the 33 UML consistency rules translated in OCL.

ID	Rule Description	UDI
R9	A transition t″ (of U″) that is refined from t (of U) must have s″ (state sender and receiver of U″) that is refined from s (of U) ∈ Source+(t)	SMD
R13	A state machine must be deterministic, that is, in every state, only one transition should fire on a reception of an event.	SMD
R24	In a sequence diagram, if an attribute is assigned the return value of a message, then the types have to be compatible	SD
R27	The number of occurrences of a link in an object diagram, an instance of an association in a class diagram, must satisfy the multiplicity constraints specified for the association.	OD, CD
R28	A class name that appears in an activity diagram also appears in the class diagram.	AD, CD
R39	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, the different diagrams should specify the same scenarios: e.g., same sequence of messages/operations/actions, same branching or repetition conditions.	SD, AD
R40	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a flow of interaction between objects in an activity diagram should be a flow of interactions between the same objects in a sequence diagram.	SD, AD
R46	When one specifies an active class, i.e., one that has a state-based behavior described in a state machine diagram, and an instance of this active class is used in a communication diagram, the messages sent to this object and emitted by this object as specified in the communication diagram must comply to the protocol specified in the state machine diagram.	SMD, COMD
R48	When one specifies an active class, i.e., one that has a state-based behavior described in a state machine diagram, and an instance of this active class is used in a sequence diagram, the messages sent to this object and emitted by this object as specified in the sequence diagram must comply (e.g., sequence and types of signals, receivers and emitters of signals) to the protocol specified in the state machine diagram.	SMD, SD
R50	The noun of the use case's name should equal the name of one class in the class diagram.	UCD, CD
R54	Objects involved in a communication diagram should be instances of classes of the class diagram.	CD, COMD
R55	In order for objects to exchange messages in a communication diagram, the sending object must have a handle to the receiving object as specified in the class diagram. In other words, the sender must have visibility to the receiver. A specific case of this situation is when the sending objectâs class has an association (possibly inherited) to the receiving objectâs class.	CD, COMD
R74	Liskov substitution principle.	CD
R75	A class that realizes an interface must declare all the operations in the interface with the same signatures (including parameter direction, default values, concurrency, polymorphic property, query characteristic).	CD
R76	An abstract operation can only belong to an abstract class.	CD
R77	If an operation appears in a pre or post condition then it must have the property isQuery equal to true.	CD
R78	No (public) method of a class violates, as indicated by its pre and post-conditions, the class invariant of that class.	CD
R80	No precondition should violate the class invariant.	CD
R81	No post-condition should violate the class invariant.	CD
R82	A class that contains an abstract operation must be abstract.	CD
R84	A class cannot be a part of more than one composition - no composite part may be shared by two composite classes.	CD
R85	Each concrete class, i.e., it is not abstract, must implement all the abstract operations of its superclass(es).	CD
R98	Each sequence diagram corresponds to a use case, and each use case in the use case diagram is specified by an interaction diagram.	UCD, SD

Table 2: List of the 33 UML consistency rules translated in OCL.

ID	Rule Description	UDI
R101	If an assembly connector exists between two ports, one of the ports (the source) must be a required port, the other port (the destination) must be a provided port. This rule describes the opposite case of delegation, where both ports at the end of an assembly connector have conjugate interfaces (one port requires an interface; the other provides the same interface). What matters is that the two ports must either have the same interface but one of them is marked as isConjugate, while the other is not, or they should have conjugate interfaces.	CSD
R102	If a connector is typed with an association, the direction of the association must conform to the direction of the connector as derived from the direction of the ports at its ends (association navigable from class A to class B if the connector between A and B indicates that A requires services that B provides). Given that the direction of associations and connectors (however, it is calculated) could be encoded using the order of their 'memberEnd' and 'end' collection, respectively, the rule is basically saying that such direction should be the same in both cases, if a connector is typed by an association.	CSD
R105	The set of transported interfaces by a link should not be empty.	CSD
R106	If several non-typed connectors start from one port, then the sets of interfaces transported by each of these connectors have to be pair wise disjoint.	CSD
R108	The type of a lifeline (type of the connectable element of the lifeline) in a sequence diagram must not be an interface nor an abstract class.	SD, CD
R109	In case a message in a sequence diagram is referring to an operation, that operation must not be abstract.	SD, CD
R110	If a message in a sequence diagram refers to an operation, through the signature of the message, then that operation must belong, as per the class diagram, to the class that types the target lifeline of the message.	SD, CD
R112	In order for objects to exchange messages in a sequence diagram, the sending object must have a handle to the receiving object as specified in the class diagram. In other words, the sender must have visibility to the receiver. A specific case of this situation is when the sending objectâs class has an association (possibly inherited) to the receiving objectâs class.	SD, CD
R115	Each class in the class diagram must be instantiated in a sequence diagram.	SD, CD
R116	No operation can be used in a message of a sequence diagram if this breaks the visibility rules of the class diagram (public, protected, private).	SD, CD

3.2 Formalizing the Consistency Rules in OCL

We considered the 52 consistency rules and tried to encode them in OCL. One of the authors analyzed each consistency rule (in their natural language form) and tried to propose an OCL encoding. Out of the initial set of 52 consistency rules, the 33 reported in Table 2 were encoded in OCL [27]. The other 19 were not encoded in OCL for different reasons:

(1) some of these rules can not be easily expressed in OCL (e.g., those that have complex algorithms); and
(2) in some cases, the UML standard does not include the necessary information to check the rules. For example, some of the rules requiring that the name of a use case must include a verb and a noun (for instance, Validate User) were not encoded in OCL.

The 33 consistency rules that we encoded in OCL are made available as an Eclipse Papyrus OCL library in our online appendix [27].

To formalize some of these 33 consistency rules in OCL, we had to interpret the semantics of some of the words used in the plain English description, and simplify the rule to be able to encode it in OCL. For example, rule R115 requiring that each class in the class diagram must be instantiated in a sequence diagram was encoded in OCL as follows:

context UML :: $InteractionFragment$

inv Rule115 : $(self- > exists(self.name = UML :: Class.name))$

Furthermore, some of the rules presented in Table 2 need to be considered together when the OCL encoding is proposed. For instance, rule R54, that requires objects involved in a communication diagram to be instances of classes of a class diagram, and rule R115, that requires each class in the class diagram to be instantiated in a sequence diagram, are considered together when the OCL encoding is proposed.

This step finally provided an OCL encoding for each of the 33 consistency rules.

4 CHECKING THE CONSISTENCY RULES

The selected models and the OCL encoding of the considered consistency rules were checked using Eclipse Papyrus. Specifically, for each model and consistency rule, we performed the four following steps summarized in Figure 1:

Step A. We imported the model into the Eclipse Papyrus tool (version 2.0).[6]

[6]The installation of the following additional plug-in was required: OCL Classic 2 SDK, Ecore/UML Parsers, Evaluator, and Edit (version 5.2).

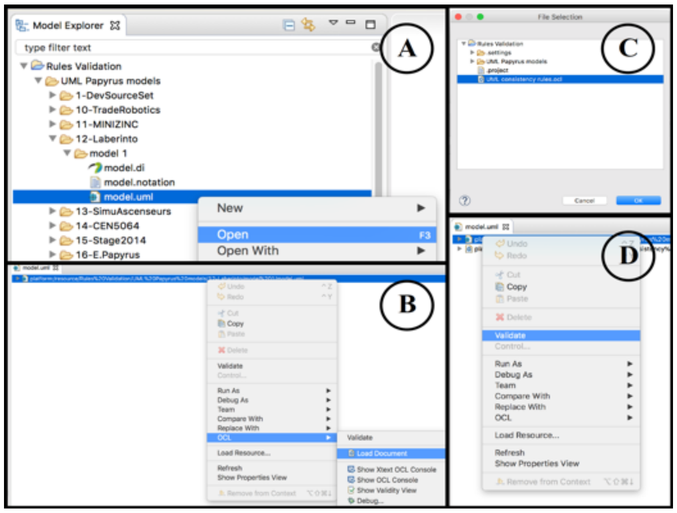

Figure 1: Eclipse Papyrus steps to check OCL rules.

Steps B-C. We loaded the consistency rule. This step added the OCL rules in our library to the set of UML well-formedness rules already implemented in Eclipse (based on the UML specification).

Step D. We validated the considered model with respect to the OCL rules loaded in the workspace. In order to run the OCL consistency rules over the UML models, we used the OCL solver implemented in the Eclipse OCL Classic 2 SDK plugin.

These steps allowed us to assess the proposed benchmark.

5 ASSESSMENT OF THE BENCHMARK

To evaluate the proposed benchmark, we considered the following research questions:

RQ1: *How useful are the UML models of the proposed benchmark in assessing consistency rules?*

We evaluate how the models of the proposed benchmark are useful in assessing consistency rules. Ideally, we would like to have a benchmark that includes models that satisfy some of the consistency rules presented in Section 3 and violate others.

RQ2: *How useful are the consistency rules of the proposed benchmark in assessing the consistency of UML models?*

We evaluate how the proposed benchmark allows assessing each consistency rules. Ideally, we would like to have a benchmark that, for each consistency rule presented in Section 3, contains some models that satisfy the rule and others that violate it

RQ3: *How useful are the different UML diagram types of the proposed benchmark in assessing consistency rules?*

We evaluate how the different UML diagram types of the proposed benchmark asses the consistency rules. Ideally, we would like to have a benchmark that, for each diagram type, includes diagrams that satisfy some of the consistency rules presented in Section 3 and violate others.

RQ4: *How useful are the different UML models of the proposed benchmark in assessing the satisfaction of consistency rules UML in different software development phases?*

We evaluate how the UML models of the proposed benchmark asses consistency rules during different software development phases. Ideally, we would like to have a benchmark that, for each software development phase, includes models that satisfy some of the consistency rules and violate others.

In order to answer these research questions we considered the 33 OCL rules identified in Section 3 and the 34 UML models containing 206 UML diagrams identified in Section 2.

5.1 RQ1 — Assessment of the UML Models

To check how useful the UML models of the proposed benchmark are in assessing consistency rules, we evaluated how many inconsistencies are contained in each of the UML models of the proposed benchmark. To reach this goal we (1) considered each of the 34 UML models; (2) for each of these models we evaluated the 33 consistency rules; and (3) we recorded the number of inconsistencies that were found.

Results. Table 3 presents the number of inconsistencies we found in each of the 34 UML models. The first row of each section presents a model identifier assigned to each UML model. The second row presents the number of inconsistencies found in each model. For example, the UML model 29.1 shows 21 inconsistencies. The results presented in Table 3 show that:

– 15 of the 34 UML models do not show any inconsistency. For three of these models (marked with an * in Table 3), the absence of inconsistencies is not due to the correctness of the models. Indeed, those models only involve a single AD (in two models) and a single UCD (in one model), which are not covered by any of our 33 OCL rules. We do not have rules that only involve (or check the correctness of them) either one or the other of these two diagrams. We do have rules that involve both diagrams (see rules R39, R40, R50, and R98 in Table 3), but none of them focuses only on UCD or AD, or checks that the two diagrams are consistent with one another. They rather relate some elements of AD or UCD with other diagrams. The remaining 12 UML models are consistent according to our 33 OCL rules. For instance, UML model 42 passed all the consistency checks. Those 12 models include 20 UML diagram types: ten CDs, six ODs, and four SDs.
– 19 of the 34 UML models contain at least one inconsistency. UML model M14, which is the biggest model of the set of models we considered,[7] triggered 1924 different inconsistencies. With this model, 13 out of 33 OCL rules (39.4%) revealed at least one inconsistency.

Answering RQ1: Our results show that the UML models of the proposed benchmark contain a number of inconsistencies that are sufficient to effectively asses the consistency rules. 15 models do not reveal any inconsistencies of the considered consistency rules, and 19 models contain a variable number of inconsistencies which depends on the models.

[7]It contains 48 UML diagrams, i.e., 12 SDs, nine CDs, six SMDs, three UCDs, six ADs, and 12 ODs.

Table 3: Number of inconsistencies (Inc) detected for each UML model (MID).

MID	M1.1	M1.1	M2	M6	M8.2	M9.2	M9.3	M9.4	M9.5	M9.6	M9.7	M12	M 13.1	M13.2	M14	M15*	M18
Inc.	46	4	75	12	0	0	2	0	0	0	2	30	68	44	1924	0	10
MID	M22	M24*	M26.1	M26.2	M28.1	M28.2	M29.1	M29.2	M30	M31	M35*	M36	M37	M38	M39	M40	M42
Inc.	2	0	4	5	0	0	21	24	156	36	0	264	0	0	0	0	0

Table 4: Number of Inconsistency (Inc) for every consistency rule (Rule).

Rule	R9	R13	R24	R27	R28	R39	R40	R46*	R48	R50	R54*	R55*	R74	R75	R76	R77	R78
Inc.	0	62	0	0	63	0	298	0	103	62	0	0	0	0	58	0	0
Rule	R80	R81	R82	R84	R85	R98	R101	R102	R105	R106	R108	R109	R110	R112	R115	R116	
Inc.	0	0	58	0	0	596	1	0	1	0	0	182	139	55	899	154	

5.2 RQ2 — Assessment of the Consistency Rules

To evaluate how useful the consistency rules of the proposed benchmark are in assessing the consistency of UML models, we measured how many inconsistencies of each type of consistency rule are contained in the UML models of the proposed benchmark. To reach this goal we (1) considered each of the 33 consistency rules; (2) for each of these consistency rules we considered the 34 UML models; and (3) recorded the number of inconsistencies that were found.

Results. Table 4 presents the number of inconsistencies we found for each of the 33 OCL rules. The identifier of the consistency rule is reported in the “Rule" labeled row, followed by the number of inconsistencies found in the 34 UML models. For example, we found a total of 58 inconsistencies across the 34 UML models for rule R82. The results presented in Table 4 show that:

- 18 out of 33 OCL rules did not find any inconsistency in the 34 UML models. Among these, three OCL rules do not find any inconsistencies in the UML models, because they specifically focus on the COMDs (R46, R54 and R55 in Table 2) and the set of UML models does not include any COMDs. Those rules are marked with the symbol * in Table 4. Among the other 15 rules, 12 involved only one UML diagram (eight CDs, two CSDs, one SDs, and one SMDs). To compare existing solvers for consistency rules checking among each other, a benchmark should either include rules that identify inconsistencies and others that do not. For this reason, we included those rules in our benchmark even if they did not find any inconsistency. Furthermore, these rules might be useful to find inconsistencies in other instances of UML models.
- 15 out of 33 OCL rules found an inconsistency in at least one of the 34 UML models. 13 of those 15 OCL rules found 86.7% of the inconsistencies. The OCL rules R98 and R115 detected the highest number of inconsistencies across the UML models, respectively 596 and 899 inconsistencies. Most of these inconsistencies were found in the UML model M14 that contains the largest number of diagrams (9 CDs, 12 SDs, and 3 UCDs), which are the diagrams considered by those two rules.

Table 5: Number of Inconsistencies (Inc) for every UML diagram type (UDT).

UDT	CD	OD	SD	SMD	UCD	AD	ID	CSD
Inc.	1670	0	1830	165	658	361	596	2

In total, the 33 OCL rules found 2731 inconsistencies on the considered 34 UML models.

Answering RQ2: Our results show that the consistency rules of the proposed benchmark are able to detect 2731 inconsistencies. 18 out of 33 OCL rules did not find any inconsistency in the 34 UML models, while 15 out of 33 OCL rules found at least one inconsistency. Furthermore, the number of inconsistencies associated with the different consistency rules is variable. This indicates that the proposed benchmark contains consistency rules that help to assess the consistency of UML models up to different degrees.

5.3 RQ3 — Assessment of the Different UML Diagram Types

To evaluate how useful are the different UML diagram types of the proposed benchmark in assessing consistency rules, we measured for each diagram type, how many inconsistencies are contained in the UML models of the considered type in the proposed benchmark. To reach this goal, we considered each of the eight UML diagram types involved in this study, and then evaluated all the 33 consistency rules on them. We recorded the number of inconsistencies that were found.

For each inconsistency identified, we computed (1) the cumulative number of models that have the right diagram types so that the OCL rules that involve those diagrams can have a chance to discover an inconsistency along (2) the cumulative number of models that contains an inconsistency as specified by the OCL rules.

Results. Table 5 describes the number of times each UML diagram type was found involved in at least one inconsistency according to any of the 33 OCL rules. For example, the OCL rules found

Table 6: Number of Inconsistencies (Inc) for every software development phase (SDP).

SDP	ReqM	IntM	DynM	LogM
Inc.	658	2426	526	1672

1670 inconsistencies that involved class diagrams (CD). Note that the sum of the inconsistencies' frequencies for each UML diagram is 5282, which is higher than 2731 (the total number of inconsistencies identified) because the rules that involved two UML diagram types were counted twice. For instance, rule R98, which describes some consistency between the UCD and CD, identified a total of 596 inconsistencies, and we, therefore, counted 596 inconsistencies for both UCD and SD in Table 5. Table 5 shows that CD and SD are the UML diagrams having the highest number of inconsistencies. This is not surprising since (1) these are the most frequently used UML diagram types [7]; and (2) the majority of our OCL rules are related to CDs and SDs, respectively with 19 (57.6%) and 11 (33.3%) rules out of 33 OCL rules, and the 31 UML models contained 82 CDs (with 898 classes) and 27 SDs (with 341 interactions) out of 206 total UML diagrams. The 54 ADs (with a total of 369 actions and decisions), which is the second UML diagram most involved in the 34 UML models after the CDs, triggered only 361 inconsistencies. The three rules that involved ADs were triggered 7 out of 10 possible times.
Answering RQ3: Our results show that the UML diagram types of the proposed benchmark contain a variable number of inconsistencies. With the exception of Object diagrams and Composite Structure diagrams, which respectively contain no inconsistency and a really low number of inconsistencies, the other UML diagram types are useful for assessing consistency rules.

5.4 RQ4 — Assessment of the Different Software Development Phases

To evaluate how useful are the different UML models of the proposed benchmark in assessing the satisfaction of consistency rules in different software development phases, we considered each of the four development phases presented in Section 2. We considered all the UML diagram types that are generated during each software development phase, and we evaluated all the 33 consistency rules for each of these diagrams. We recorded the number of inconsistencies that were found.

Results. Table 6 presents the number of inconsistencies we found in each of the four software development phases. The first row of Table 6 presents the code assigned to each development phase. The second row presents the number of inconsistencies found in each phase. For example, the Interaction Model phase (IntM in Table 6) shows 2426 inconsistencies, which is obtained from summing up the number of inconsistencies triggered by SD (1830) and ID (596). As mentioned for RQ3, the sum of the inconsistencies frequencies for each software engineering phase is 5282, which is higher than 2731 because the rules that involved two UML diagrams were counted twice. The results in Table 6, show that Interaction models (IntM) have the highest number of inconsistencies (2426 out of 2731 ≈ 88.8%), followed by Logical models (LogM — 1672 out of 2731 ≈ 61.2%), Requirements model (ReqM — 658 inconsistencies out of 2731 ≈ 24.1%), and Dynamic models (DynM — 526 out of 2731 ≈ 19.3%).
Answering RQ4: Our results show that the UML models of the proposed benchmark contain models that show examples of inconsistencies in different software development phases. As such, they are useful for assessing the satisfaction of consistency rules in different software development phases.

6 THREATS TO VALIDITY

We discuss the threats to validity that affect our evaluation of assessing the benchmark capability of assessing consistency rules (RQ1), UML models (RQ2), UML diagram types (RQ3), and software development phases (RQ4).

Construct Validity. It concerns the degree to which a test measures what it claims, or purports, to be measuring. In our study, this is related to the ability to encode the 33 consistency rules presented in Table 2 as OCL rules. The encoding of the UML consistency rules in OCL was performed by the first author and tested on various simple UML diagrams to evaluate that they check the right properties of the UML diagrams, i.e., classes, attributes, connectors, relationships, names, etc.

Internal Validity. It concerns the extent to which a piece of evidence supports a claim about cause and effect. The collection process of inconsistencies rules and the UML models is a threat to internal validity. However, the considered models come from a variety of domains, and the inconsistency rules were previously evaluated with academic researchers and industrial developers [29].

External Validity. It concerns the extent to which it is possible to generalize the findings. We can not guarantee that our findings are generalizable to other UML models. The number of inconsistencies that are present in each model depends on the experience of the developer, the intention of the model, the rigor of the project, etc. So, we can not guarantee that other UML models and inconsistency rules will provide results that are compliant with the one mentioned in RQ1, RQ2, RQ3, and RQ4.

Reliability Validity. It concerns the extent to which the results can be reproduced when the research is repeated under the same conditions. The results of RQ1, RQ2, RQ3, and RQ4 presented in this paper are related to the 33 models and 31 UML models considered in this study. There is no guarantee these results will generalize to other models. However, the open-source UML models that were collected were developed in different software domains (i.e., aerospace, logistics, gaming, electronics, service, robotics, etc.). As such, they are representative of different domains.

7 DISCUSSION

In this section, we discuss the results of this case study.

7.1 Combining RQ1, RQ2, and RQ3.

In this section, we try to critically analyze and combine the results of RQ1, RQ2 and RQ3. To reach this goal, Table 7 aggregates the results presented in Tables 3 and 4. The column "TD" reports the different diagram types. We decided to aggregate CSD with CD and ID with SD as they contain similar UML elements. The column "Rules" reports the total number of OCL rules that involve each

Table 7: Analysis of the results according to RQ1-RQ3.

TD	Rules	Matches	Inc	TI	IM	TTI
CD/CSD	23	247	53	1672	21%	31
AD	3	10	7	361	70%	51
SD/ID	12	51	24	2426	47%	101
OD	1	1	0	0	0%	-
SMD	4	8	5	165	62%	33
UCD	2	4	4	658	100%	164

diagram type: for instance, 12 rules involve SDs, three rules involve ADs. The column "Matches" reports the number of times an OCL rule was analyzed on that specific diagram type. For instance, the value of CD and SD is incremented every time the rule R109 is analyzed, as it involves both CD and SD (see Table 2). The column "Inc" contains the number of times at least one inconsistency was detected by the considered OCL rules. For instance, the value of both CD and SD is incremented when rule R115 is analyzed on M30, as 91 inconsistencies were detected, and M3 involves CD and SD. The column "TI" (Total Inconsistencies) contains the total number of inconsistencies discovered by the OCL rules for each type of diagram.

Table 7 shows that while the models of the proposed benchmark contain a considerable number of inconsistencies, these are not equally distributed across different models. As follows, we computed the number of times inconsistencies were detected for any given rule concerning the number of matches:

$$IM = \frac{Inc}{Matches} * 100$$

We report the results in in Table 7 (Column IM). IM gives an indication about which models are more suitable for being used in the analysis of solvers. For some UML diagram types (e.g., CD/CSD 21%) is lower than for other types of models (e.g., SD/ID 47%). If the goal is to analyze the capabilities of two solvers in detecting inconsistencies, it is better to consider models with a high number of inconsistencies and compare the inconsistencies detected by the two solvers. Furthermore, the average number of inconsistencies detected for every model for the rules that found at least one inconsistency as

$$TTI = \frac{TI}{Inc}$$

and report it in Table 7 (Column TTI). TTI gives an indication about which rules are more suitable for begin used in the analysis of solvers. For some UML diagram types (e.g., CD/CSD 31), the number of inconsistencies founded for each rule is lower than for other types of models (e.g., SD/ID 101). If the goal is to analyze the capabilities of two solvers in detecting inconsistencies, it is better to consider models that allow detecting for each consistency rule a high number of inconsistencies and then compare the inconsistencies detected by the solvers.

Table 8: Summary of related work.

Authors	Year	Case Study	#OCL Rules	Models
Reder and Egyed [23]	2013	Yes	20	29
Gogolla et al. [12]	2015	Yes	10	18
Czarnecki and Pietroszek [5]	2006	Yes	1	1
Liu et al. [16]	2002	No	1	none
Sapna and Mohanty [9]	2007	No	1	none

7.2 UML Consistency Rules Most Triggered

The results presented in Section 5 show that most of the inconsistencies identified by the OCL rules involve rules that focus on two UML diagrams, i.e., CDs and SDs. Seven of the 15 OCL rules that found inconsistencies (rules R40, R48, R98, R109, R110, R115, and R116), found more than 100 inconsistencies each (see Table 4). All of these rules involve two diagrams, and one of those two diagrams is always the sequence diagram. This indicates that our benchmark is representative of real development. Indeed, in practical cases, for UML designers, it is usually easier to keep consistency within a single UML diagram rather than keeping consistency between different UML diagram types.

7.3 Limitations of the Benchmark

We validated a limited number of inconsistencies and UML models. Those models might not be representative of various domains. In addition, the sampling of UML models only considered Eclipse Papyrus models. Our collection strategy could have incurred a possible selection bias (for example, a high probability of similar UML models). Finally, in the open-source UML models, we might not be representative of industrial models.

We nevertheless believe that, given the extent of the models we used, and the extent of the inconsistencies we revealed, the results of this study are still meaningful and valuable. This is a preliminary step that we believe that can stimulate the development of a benchmark of UML models and OCL consistency rules that can be used as a reference benchmark by the research community. As such, our benchmark is open to extensions, i.e., new UML models and OCL encoding of consistency rules can be proposed by the community and added to the benchmark.

8 RELATED WORK

In this section, we present related work that (1) consider OCL rules as an instrument to check consistency between UML diagrams, and (2) Benchmarks that consider OCL rules.

8.1 Checking UML Consistency with OCL Rules

Those works are reported in Table 8, where we report the authors and reference of the study, the year of the publication, whether the paper involves any case study, the number of OCL rules discussed in the paper, and finally the number of models involved.

Reder and Egyed [23] presented an approach for identifying how design rules cause a given inconsistency and what model elements are involved. They demonstrated their approach on 29 UML models and 20 OCL rules. Three of these 20 rules were considered in our work as well (see section 3.1.1 for details); The rest (17) of the rules were collected in our previous work [30] as UML consistency rules, but the MDSE experts of our survey [29] did not consider they should be enforced in every UML model. For this reason, those 17 rules were not included in this work.

Gogolla et al. [12] report on a middle-sized case study, in which the consistency of a UML class model is checked. The class model restrictions are expressed by UML multiplicity constraints and explicit, non-trivial OCL rules. Their approach (i.e., satisfiability analysis) automatically searches for a valid system state that is an instance of the class model. Failure to find a valid state that satisfies the OCL rules is an indication of inconsistency.

Czarnecki and Pietroszek [5] use OCL to define well-formedness rules for the verification of feature-based model templates that are analyzed by an Unrestricted satisfiability (SAT) solver. Their approach allows OCL constraints to be written against the meta-model of the target notation for the template instances. Then these constraints can be evaluated against a template using a template interpretation for OCL. They presented a tool prototype that was tested on a business model for an e-commerce platform.

Liu et al. [16] describe a rule-based solution to detect software design inconsistency problem. They characterized classes of inconsistency that occur in software design. They defined a production system language and rules specific to software design modeled in UML. Using this approach, they detect inconsistencies, notify the users, recommend resolutions, and automatically fix the inconsistency during the design process.

Sapna and Mohanty [9] deal with structural inconsistencies between use case, activity, collaboration, state machines, and class diagrams by using OCL rules converted to SQL triggers applicable across tables that store the UML diagrams.

This paper complements these works in several ways. It proposes a broader set of consistency rules identified and validated by MDSE experts and evaluates these rules on a more extensive set of models. Moreover, the objective of this case study is to collect and analyze UML models and consistency rules proposed in the literature and at promote the development of a reference benchmark that can be reused by the research community.

8.2 Benchmarks Using OCL Rules

Benchmarks that consider OCL rules have been proposed in the literature [11, 13, 15]. Differently from these benchmarks, in this work, the OCL rules were obtained from consistency rules that were (1) systematically collected from the literature, and (2) evaluated by MDSE experts. Furthermore, the UML models were collected by considering existing open-source UML repositories. As such, we believe that the proposed benchmark is likely to be more representative of real case scenarios. An in-depth comparison and integration of our benchmark with existing benchmarks proposed in the literature will be performed as future work.

9 CONCLUSION

This work presents the results of a case study that involved a set of 33 consistency rules and 34 UML models. We proposed an OCL formalization of 33 consistency rules and validated them on the 34 UML models. We were able to identify 2731 inconsistencies among the 206 UML diagrams included in the 34 UML models. The formalization of the consistency rules and the UML model are released in the form of a benchmark that can be reused by the research community, e.g., to validate new FM-based solvers.

We have evaluated the proposed benchmark by assessing how the considered UML models, consistency rules, and model types contained in the benchmark help in assessing the consistency of UML models (RQ1, RQ2, and RQ3). Furthermore, we checked if the benchmark allows assessing inconsistencies that occur in different software development phases (RQ4). Our results show that

(1) the UML models contained in the benchmark are sufficiently diverse to represent conditions that occur during real development as they contain a variable number of inconsistencies;
(2) the consistencies rules contained in the benchmark are sufficiently diverse to represent conditions that occur during real development as they can detect a variable number of inconsistencies;
(3) all the considered UML diagram types, excluding object diagrams and composite structure diagrams, contain at least one inconsistency;
(4) the diagrams contained in the UML models of the proposed benchmark allow identifying a considerable number of inconsistencies for each software development phase.

We believe that the present work will contribute to the creation of a common reference benchmark that can allow assessing consistency rules on UML models. The benefits of the presence of such a benchmark will be multiple. First, it will provide a library of template OCL formulae associated with existing consistency rules proposed in the literature. Those formulae can be reused within the research community. Second, it provides a reference benchmark that developers can reuse for evaluating existing/new OCL solvers. As such, our benchmark is open to extensions. As new UML models and OCL encoding of consistency rules are used/proposed by the research community, they can be added to the proposed benchmark.

As future work, we plan to consolidate the list of OCL rules we have specified in this paper. We intend to encode in OCL the reminding 63 UML consistency rules that we collected in Torre et al. [30] and evaluate and compare existing OCL solvers. We will also try to collect and analyze other UML models that are not designed using Eclipse Papyrus.

ACKNOWLEDGMENTS

Discovery grant of the Natural Sciences and Engineering Research Council of Canada, the GEMA project ("Consejería de Educación, Cultura y Deporte de la Dirección General de Universidades, Investigación e Innovación de la JCCM", SBPLY/17/180501/000293), the ECLIPSE project ("Ministerio de Ciencia, Innovación y Universidades, y FEDER", RTI2018-094283-B-C31), and the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277).

REFERENCES

[1] Jim Arlow and Ila Neustadt. 2005. *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition).* Addison-Wesley Professional.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice, Second Edition.* Morgan & Claypool Publishers.

[3] Lionel C. Briand, Yvan Labiche, and Leeshawn O'Sullivan. 2003. Impact Analysis and Change Management of UML Models. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands.* 256–265.

[4] Jordi Cabot, Robert Clarisó, and Daniel Riera. 2009. Verifying UML/OCL Operation Contracts. In *Integrated Formal Methods, 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16-19, 2009. Proceedings.* 40–55. https://doi.org/10.1007/978-3-642-00255-7_4

[5] Krzysztof Czarnecki and Krzysztof Pietroszek. 2006. Verifying feature-based model templates against well-formedness OCL constraints. In *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, October 22-26, 2006, Proceedings.* 211–220.

[6] Brahma Dathan and Sarnath Ramnath. 2015. *Object-Oriented Analysis, Design and Implementation - An Integrated Approach, Second Edition.* Springer.

[7] Brian Dobing and Jeffrey Parsons. 2006. How UML is used. *Commun. ACM* 49, 5 (2006), 109–113.

[8] Iulia Dragomir and Iulian Ober. 2010. Well-formedness and typing rules for UML Composite Structures. *CoRR* abs/1010.6155 (2010). arXiv:1010.6155 http://arxiv.org/abs/1010.6155

[9] Sapna P. G. and Hrushikesha Mohanty. 2007. Ensuring Consistency in Relational Repository of UML Models. In *10th International Conference on Information Technology, ICIT 2007, Roukela, India, 17-20 December 2007.* 217–222.

[10] Marcela Genero, Ana M. Fernández-Sáez, H. James Nelson, Geert Poels, and Mario Piattini. 2011. Research Review: A Systematic Literature Review on the Quality of UML Models. *J. Database Manag.* 22, 3 (2011), 46–70.

[11] Martin Gogolla, Fabian Büttner, and Jordi Cabot. 2013. Initiating a Benchmark for UML and OCL Analysis Tools. In *Tests and Proofs TAP.* Springer, 115–132.

[12] Martin Gogolla, Lars Hamann, Frank Hilken, and Matthias Sedlmeier. 2015. Checking UML and OCL Model Consistency: An Experience Report on a Middle-Sized Case Study. In *Tests and Proofs - 9th International Conference, TAP 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 22-24, 2015. Proceedings.* 129–136.

[13] Martin Gogolla, Mirco Kuhlmann, and Fabian Büttner. 2008. A Benchmark for OCL Engine Accuracy, Determinateness, and Efficiency. In *Model Driven Engineering Languages and Systems (MoDELS).* Springer.

[14] Regina Hebig, Truong Ho-Quang, Michel R. V. Chaudron, Gregorio Robles, and Miguel Angel Fernández. 2016. The quest for open source projects that use UML: mining GitHub. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016.* 173–183. http://dl.acm.org/citation.cfm?id=2976778

[15] Mirco Kuhlmann, Lars Hamann, Martin Gogolla, and Fabian Büttner. 2012. A benchmark for OCL engine accuracy, determinateness, and efficiency. *Software and Systems Modeling* 11, 2 (2012), 165–182. https://doi.org/10.1007/s10270-010-0174-8

[16] WenQian Liu, Steve Easterbrook, and John Mylopoulos. 2002. Rule-Based Detection of Inconsistency in UML Models. In *Workshop on Consistency Problems in UML-Based Software Development.*

[17] Jishnu Mukerji and Joaquin Miller. 2003. MDA Guide V1.0.1, Overview and guide to OMG's architecture. *Object Management Group* (2003). http://www.omg.org/mda/

[18] OMG. 2015. OMG Unified Modeling LanguageTM - Superstructure Version 2.5. *Object Management Group* (2015). http://www.omg.org

[19] OMG. 2016. Object Management Group - Object Constraint Language (OCL). *Object Management Group* (2016). http://www.omg.org/spec/OCL

[20] Rajwinder Kaur Panesar-Walawege, Mehrdad Sabetzadeh, and Lionel C. Briand. 2013. Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. *Information and Software Technology* 55, 5 (2013), 836–864.

[21] Tom Pender. 2003. *UML Bible.* John Wiley & Sons, Inc.

[22] Hongyang Qu and Sandor M. Veres. 2016. Verification of logical consistency in robotic reasoning. *Robotics and Autonomous Systems* 83 (2016), 44–56. https://doi.org/10.1016/j.robot.2016.06.005

[23] Alexander Reder and Alexander Egyed. 2013. Determining the Cause of a Design Model Inconsistency. *IEEE Trans. Software Eng.* 39, 11 (2013), 1531–1548.

[24] Ed Seidewitz. 2019. At the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, Germany, September 15-20, 2019. https://twitter.com/MFamelis/status/1174663155798224897?s=20

[25] Dave A. Thomas. 2004. MDA: Revenge of the Modelers or UML Utopia? *IEEE Software* 21, 3 (2004), 15–17.

[26] Damiano Torre. 2014. On collecting and validating UML consistency rules: a research proposal. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014.* 57:1–57:4.

[27] Damiano Torre. 2018. Benchmark Material. https://github.com/yvanlabiche/UML-model-consistency

[28] Damiano Torre, Yvan Labiche, and Marcela Genero. 2014. UML consistency rules: a systematic mapping study. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014.* 6:1–6:10.

[29] Damiano Torre, Yvan Labiche, Marcela Genero, and Maged Elaasar. 2018. How consistency is handled in Model Driven Software Engineering: a survey of experts in academia and industry. In *Technical Report, Carleton University.*

[30] Damiano Torre, Yvan Labiche, Marcela Genero, and Maged Elaasar. 2018. A systematic identification of consistency rules for UML diagrams. *Journal of Systems and Software* 144 (2018), 121–142.

[31] Damiano Torre, Ghanem Soltana, Mehrdad Sabetzadeh, Lionel C. Briand, Yuri Auffinger, and Peter Goes. 2019. Using Models to Enable Compliance Checking Against the GDPR: An Experience Report. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, Germany, September 15-20, 2019.* 1–11.