The Journal of Systems & Software 144 (2018) 121–142

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

A systematic identification of consistency rules for UML diagrams

Damiano Torre[a,b,*], Yvan Labiche[a], Marcela Genero[b], Maged Elaasar[a]

[a] Carleton University, Ottawa, Canada
[b] University of Castilla-La Mancha, Ciudad Real, Spain

ARTICLE INFO

Keywords:
UML consistency rules
Systematic Mapping Study
Model checking and verification

ABSTRACT

UML diagrams describe different views of one software. These diagrams strongly depend on each other and must therefore be consistent with one another, since inconsistencies between diagrams may be a source of faults during software development activities that rely on these diagrams. It is therefore paramount that consistency rules be defined and that inconsistencies be detected, analyzed and fixed. The relevant literature shows that authors typically define their own consistency rules, sometimes defining the same rules and sometimes defining rules that are already in the UML standard. The reason might be that no consolidated set of rules that are relevant by authors can be found to date. The aim of our research is to provide an up to date, consolidated set of UML consistency rules and obtain a detailed overview of the current research in this area. We therefore followed a systematic procedure in order to collect from literature up to March 2017 and analyze UML consistency rules. We then consolidated a set of 119 UML consistency rules (avoiding redundant definitions or definitions already in the UML standard), which can be used as an important reference for UML-based software development activities, for teaching UML-based software development, and for further research.

1. Introduction

Model Driven Architecture (MDA) (Mukerji and Miller, 2003) is an approach for software development provided by the Object Management Group (OMG) that promotes the efficient use of models throughout software development phases, from requirements to analysis, design, implementation, and deployment (Thomas, 2004). Much attention has been paid to MDA by academia and industry in recent years (Genero et al., 2011; Lucas et al., 2009; Usman et al., 2008), which has resulted in models gaining more importance in software development, from requirement analysis until to test (Lamancha et al., 2013). The OMG specification that is most frequently used is the Unified Modeling Language (UML) (OMG, 2015), which is the de-facto standard modeling language for software modeling and documentation (Pender, 2003). It is the preferred modeling language when implementing MDA, although it is not intended to be used in every single software development project (Petre, 2013). The UML provides 14 diagram types (OMG, 2015) with which to describe a software system from different perspectives (e.g., structure and behavior), abstraction levels (e.g., analysis and design), code maintenance (Fernández-Sáez et al., 2016; Fernández-Sáez et al., 2015) (e.g.; reverse-engineering code to models), which, among other things, help deal with complexity and distribute responsibilities.

Since the various UML diagrams in a model typically describe different views of one software system under development, they depend on each other to a significant extent and must, therefore, be consistent with one another. As the UML is not a formal specification, inconsistencies may arise between different diagrams (Ibrahim et al., 2011b). When UML diagrams communicate contradicting or conflicting syntax or semantics, the diagrams are said to be inconsistent (Simmonds et al., 2004). Such inconsistencies may be a source of faults in software systems at a later date (Ibrahim et al., 2011b; Simmonds et al., 2004) and it is, therefore, of paramount importance that the consistency between diagrams be defined and that inconsistencies be routinely detected, analyzed and fixed (Spanoudakis and Zisman, 2001).

In an attempt to obtain an accurate picture of the current research in the area of UML diagram consistency, we conducted a systematic mapping study (SMS) (Torre et al., 2014), which resulted in a number of findings regarding publications in this domain. Although the aim of this SMS was to discover frequencies of publications in different categories and was not specifically to study published UML diagram consistency rules, the study provided anecdotal evidence that: (1) authors routinely define consistency rules for different UML-based software engineering activities; (2) some authors tend to define similar, often identical rules, over and over again, and (3) some authors are not aware that the UML standard specification defines consistency rules (a.k.a., well-formedness rules). We also observed that even though many researchers have either explicitly or implicitly proposed rules with which

* Corresponding author.
E-mail addresses: dctorre@sce.carleton.ca (D. Torre), yvan.labiche@carleton.ca (Y. Labiche), marcela.genero@uclm.es (M. Genero).

https://doi.org/10.1016/j.jss.2018.06.029
Received 31 October 2017; Received in revised form 1 June 2018; Accepted 7 June 2018
Available online 08 June 2018

to detect inconsistencies in a UML model, no consolidated set of UML consistency rules has been published to date. By the word "consolidated", we mean the process of combining a number of UML consistency rules that other authors feel are important into a single coherent set. We believe that, even though the rules actually required in practice may be domain, organization, or project specific, this lack of a consolidated list of UML consistency rules forces researchers to define the rules that they rely on for their own research (Ibrahim et al., 2011b), thus resulting in some rules being defined over and over again.

This motivated the main research objective of this paper, which is to identify a consolidated set of consistency rules for UML diagrams. With the term "UML consistency rule", we refer to a statement that can be expressed in plain English or in a computer language (such as the Object Constraint Language (OMG, 2016)) and which checks that one or more UML diagrams do not represent any contradicting or conflicting syntax or semantics. For instance, rule 76: "an abstract operation can only belong to an abstract class" (see Table 6), specifies a rule that involves only the class diagram and it checks that all the abstract operations (if any) of the class diagram in the model belong to an abstract class of the class diagram, otherwise an inconsistency is found, which will need to be analyzed and fixed. Other UML consistency rules can involve two or more UML diagrams, such as rule 28: "a class name that appears in an activity diagram also appears in the class diagram" (see Table 6), which specifies a rule that involves activity and class diagrams. This rule checks that all the class names appearing in the activity diagram of the model also appear in the class diagram of the same model, otherwise an inconsistency is detected. Identifying a set of UML consistency rules of this nature has several benefits: it will be a reference for practitioners, educators, and researchers alike; it will provide guidance as to which rules to use in each context; and it will highlight which areas are more developed and which need further work. Moreover, the consolidated set of rules could be a good input for the UML revision task force for inclusion in a forthcoming revision of the UML standard.

In order to identify such a consolidated set of UML consistency rules, we decided to update our SMS (Torre et al., 2014) (which was published at the 18th International Conference on Evaluation and Assessment in Software Engineering) by considering literature up to March 2017, after which we collected and analyzed the rules we found in 105 primary studies. This was done by following a systematic procedure inspired by well-known guidelines for empirical research in software engineering (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015). These guidelines provide a systematic and rigorous procedure that is used to identify the quantity of existing research related to a specific research topic.

The research work presented in this paper is, therefore, an extension of our SMS (Torre et al., 2014). Of the 105 primary studies identified during the SMS (95 papers from our initial SMS (Torre et al., 2014), and 10 papers from the SMS updated to March 2017), we collected 687 consistency rules, including duplicates and rules already in the UML standard. We then analyzed these rules in order to answer a number of research questions:

RQ1) What are the existing UML consistency rules?
RQ2) Which types of consistency problems are tackled in the existing rules?
RQ3) What types of UML diagrams are involved in UML consistency rules?
RQ4) For what software engineering activities are UML consistency rules used?
RQ5) Which UML diagram elements are involved in UML consistency rules?
RQ6) What software development phases are involved with UML consistency rules?

We subsequently employed a systematic refinement process with the objective of removing duplicates and rules already in the UML standard, as explained later in this paper, and eventually compiled a consolidated set of 119 rules. We believe that this is an important long-awaited contribution. Another important contribution is a discussion, structured by means of our research questions, of the current state of the art and practice with regard to UML diagram consistency and the future directions in which further research is needed.

In summary, this manuscript differs from our previous work (Torre et al., 2014) in four ways: 1) in this paper we extend our previous work (Torre et al., 2014), which collected primary studies up to 2013, by including all the UML consistency papers published from 2013 up to March 2017 (by following the same systematic procedure); 2) in our conference paper (Torre et al., 2014) we did not present any rules and solely discussed metadata concerning the rules, while in this work we present the consolidated set of 119 UML consistency rules and provide a detailed analysis of them; 3) unlike our previous work, we now filter out the rules we collected that are already in the UML standard; 4) the focus of our conference paper (Torre et al., 2014) was primary studies (i.e., publications) and we, therefore, studied how UML diagrams are involved in published research that presents consistency rules, whereas in this work the focus of the study is the rules and we, therefore, study how UML diagrams, software engineering activities and software development methodologies are related to these rules.

The remainder of this document is structured as follows: Section 2 provides a summary of the related work, while the definition of the systematic procedure that we followed is presented in Section 3. Section 4presents the execution of the study, whose results are described in Section 5. A discussion of the threats to validity is presented in Section 6, and finally, our conclusions and outlines of future work are shown in Section 7.

2. Related work

Since this paper focuses on the systematic identification of UML consistency rules, the related work pertains to the systematic discussions of UML consistency. In Table 1, we present a summary of the five

Table 1
Summary of related work.

Reference	Search period	Summary of the main objective	RQs	Type of Study	Papers reviewed	Rules identified
(Kalibatiene et al., 2013)	2008–2011	They propose a rule-based method for consistency checking in UML models.	None	Not systematic	8	50
(Spanoudakis and Zisman, 2001)	Up to 2000	They present a survey of techniques and methods to support the management of inconsistencies in various software models.	None	Not systematic	19	None
(Usman et al., 2008)	Up to 2007	They present a survey of different UML consistency checking techniques.	None	Not systematic	17	None
(Lucas et al., 2009)	2001–2007	They aim to discover the various current model consistency conceptions, proposals, problems and solutions provided in the literature regarding UML model consistency management.	6	SMS	44	None
(Ahmad and Nadeem, 2010)	2002–2004	They evaluate the existing Description Logic-based approaches with which to check UML consistency.	None	Not systematic	3	None

pieces of work that fit this description. For each one, the table indicates the period covered by the survey, the main objective, whether the search was driven by clear research questions (RQs) and was systematic, along with the number of reviewed papers and UML consistency rules collected. We first discuss the only review (Kalibatiene et al., 2013) that presented UML consistency rules, and then the other four reviews on UML consistency which we considered important to include.

The work of Kalibatiene and colleagues (Kalibatiene et al., 2013) is, to the best of our knowledge, the closest piece of work to our research since the authors searched for, collected and presented UML diagram consistency rules. They obtained 50 rules by reviewing eight articles, which they did not select by following a systematic protocol (Systematic Literature Review or SMS): no mention was made of the processes used to obtain the articles or to include/exclude these documents. After conducting our research, we confirmed that all of their 50 rules are included in our final set of 119 rules, which we obtained by following a precise, systematic and repeatable procedure.

The following four pieces of work are reviews, whose purpose is to increase the body of knowledge on UML diagram consistency, although not necessarily on UML consistency rules.

Spanoudakis and Zisman (Spanoudakis and Zisman, 2001) presented a literature review on the problem of managing inconsistencies in software models in general, but not specifically UML models. They presented a conceptual framework that views inconsistency management as a process, which incorporates activities, with which to detect overlaps and inconsistencies between software models, diagnose and handle inconsistencies, track the information generated along the way, and specify and monitor the exact means used to carry out each of these activities. They then surveyed 19 research works published prior to 2000 that address one or more of the aspects of their conceptual framework. None of these 19 papers was identified in our study because they do not present any UML consistency rules.

Usman and colleagues (Usman et al., 2008) presented a literature review of consistency checking techniques for UML models. The authors argued that formalizing UML models is preferable to verifying consistency because this helps remove ambiguities and enforce consistency. They briefly reviewed 17 articles, which represent less than a quarter of the number of articles considered in our work (105): the two pieces of research have only ten studies in common, since our search has a different objective. During this survey, the authors did not follow any Systematic Literature Review (SLR) or SMS protocols, and they simply provided an initial summary of their findings on UML consistency checking techniques (not consistency rules).

Lucas, Molina, and Toval (Lucas et al., 2009) presented an SMS on UML consistency management in which they reviewed 44 papers published between 2001 and 2007, focusing on the consistency in two or more UML diagrams. This work is different from ours in several ways. The first important difference is that they did not present any UML consistency rules during their review. The second main difference is the purpose: they focused on the management of UML (in)consistencies, i.e., they focused on the techniques used to identify and fix inconsistencies, without providing a detailed discussion of which inconsistencies had to be identified and fixed. In contrast, our work focuses on those inconsistencies that need to be identified and fixed. A direct consequence of this difference is that we considered a broader number of articles in order to consolidate the set of UML consistency rules (105 articles rather than 44), and approximately half of their studies (24) are not considered in our work.

Ahmad and Nadeem (Ahmad and Nadeem, 2010) presented a literature review restricted to Description Logic (DL)-based consistency checking approaches. They reviewed three articles, which are also studied in our research. Their main finding is that only class diagram, sequence diagram and state machine diagram inconsistencies were covered in the papers surveyed and that a few common types of inconsistencies were discussed.

Our work differs from the four reviews mentioned above (Ahmad and Nadeem, 2010; Lucas et al., 2009; Spanoudakis and Zisman, 2001; Usman et al., 2008) in three ways: a different goal, a more extensive and systematic review process, and the presentation of UML consistency rules. Our goal is to identify which consistency rules for all the UML diagrams have been proposed in literature and to create a consolidated set of UML consistency rules. This contrasts with previous reviews that had very different goals, such as focusing on a small subset of UML diagrams, inconsistency management and consistency checking techniques.

Another difference is that all but one piece (Lucas et al., 2009) of work performed an informal literature review or comparison with no defined research question, no precise repeatable search process and no defined data extraction or data analysis process. Our work, however, follows a systematic procedure inspired by a strict, well-known protocol (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015).

Finally, all the five works presented above are outdated when compared with the study reported in this manuscript.

In summary, we were unable to find answers to the question posed in our main research objective (Section 1), which confirmed the need for the systematic construction of a consolidated set of UML consistency rules.

3. Planning of the SMS

As discussed earlier, we started this work using the 105 primary studies, obtained in our SMS (Torre et al., 2014) and its update as a basis (see Appendix B for the detailed planning of this SMS). Of an initial set of 1713 research papers, 105 primary studies were selected by following a precise selection protocol driven by seven research questions (see Appendix B). The primary studies were then classified according to several criteria that were also derived from the aforementioned research questions, after which an initial set of 687 rules was collected. Since our previous work was not meant to provide a list of UML consistency rules, we did not present the rules collected; our previous work presented only anecdotal evidence that justified this new contribution (as stated in the introduction).

In this section, we present the procedure we followed, inspired by a well-accepted systematic search and analysis process (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015), to obtain our consolidated set of UML consistency rules, starting from the initial set of 687 rules collected from the 105 primary studies. From here on, UML consistency rules will be referred to simply as rules. We present the main components of the planning of our work, which are: the specification of research questions that the study aims to answer (Section 3.1); the procedure followed to select (or reject) rules to be part of our final consolidated set of rules (Section 3.2); and the procedure used to extract data from the selected rules in order to answer the research questions (Section 3.3).

3.1. Research questions

The underlying motivation for the research questions was to analyze the state-of-the-art regarding the rules. In order to do that, we considered six research questions (RQs): Table 2.

Three of those six research questions (specifically, RQs 1, 2, and 3) had already been used in our previous work (Torre et al., 2014) (see Section 1 of Appendix B). The main differences between the three RQs in this paper and those in our previous work (Torre et al., 2014) lies in how and for what they are used:

RQ1: In our previous work, we did not present any rules (we merely provided a link to a webpage with a list of rules) and solely discussed metadata concerning the rules, while in this work we present the consolidated set of rules and provide a detailed analysis of them.

Table 2
Research questions.

Researchquestions	Main Motivation
RQ1) What are the existing UML consistency rules?	To find the UML consistency rules used in literature in order to assess the state of the field.
RQ2) Which types of consistency problems are tackled in the existing rules?	To find the types of consistency problems tackled in the UML consistency rules according to different well-accepted dimensions in the field of model consistency: 1) horizontal, vertical and evolution consistency; 2) syntactic and semantic consistency; 3) observation and invocation consistency, and to assess the state of the field as regards that dimension of UML diagram consistency (see Section 4 in Appendix B for definitions and examples)
RQ3) What types of UML diagrams are involved in UML consistency rules?	To discover the UML diagrams that research and practice has focused on, to reveal which UML diagrams are considered more important than others, and to identify opportunities for further research.
RQ4) For what software engineering activities are UML consistency rules used?	To find the different software engineering activities that require UML diagrams to be consistent according to specific sets of consistency rules, in order to understand what they should focus on.
RQ5) Which UML diagram elements are involved in UML consistency rules?	To find the UML diagram elements that are most frequently tackled in the UML consistency rules.
RQ6) What software development phases are involved with UML consistency rules?	To find the software development phases that are most frequently addressed in the UML consistency rules.

RQ2: Unlike our previous work, we now filter out the rules we collected that are already in the UML standard.
RQ3: In our previous work, the focus of the study was on the papers and we, therefore, studied how UML diagrams are involved in papers that present consistency rules, whereas in this work, the focus of the study is on the rules and we, therefore, study how UML diagrams and Software Engineering(SE) activities are related to them.

The other three research questions (RQs, 4, 5, and 6) are new and specific to this paper.

3.2. Inclusion and exclusion criteria

In this section, we discuss the inclusion and exclusion criteria used to refine the set of rules. Since we used the 687 rules that resulted from the update of our SMS (Torre et al., 2014) as a starting point, and those rules had already been obtained through the use of inclusion and exclusion criteria, we applied only the following additional exclusion criteria to the rules:

- Inaccurate rules:
  - ○ Rules that we considered wrong. For instance, one rule specified that each message in a sequence diagram should trigger a transition in a state machine diagram. We believe that this is incorrect, since some of the classifiers receiving a message in a sequence diagram may not have a state-based behavior, and even if they have, not all messages need to trigger state changes;
  - ○ Rules not focusing on UML diagram consistency. For instance, rules which discussed consistency between UML diagrams and other non-UML data, such as requirements or source code, were discarded;
  - ○ Rules that were considered obsolete because they focus on UML characteristics that are no longer supported in the latest UML standard (version 2.5);
- Redundant rules, i.e., all those rules that can be implied by other rules;
- Rules that are already included in previous UML standard versions.

3.3. Data extraction strategy

We answered the research questions (Table 2) by extracting data from the collected rules and recording that data on the Excel spreadsheet (our data extraction form) presented in Table 3.
For each rule, we collected the following data:

- Reference and year of the primary study paper from which the rule was collected (see Appendix A of our online technical report (Torre et al., 2015a)) (RQ1).

Table 3
Data extraction form.

Extraction form – UML consistency rules		
Paper	Reference	Year of publication
Rule	Definition	
	Type	Dimension
	UML diagram(s) involved	SE activity involved
	SDM#1 phase(s) involved	
	SDM#2 phase(s) involved	
	SDM#2 phase(s) involved	
Reason for exclusion (if applicable)		

- Reason for excluding the rule, as per the exclusion criteria (see Section 3.2) (RQ1).
- Precise definition of the rule (see Section 5 and Appendix A) (RQ1 and RQ5). When the rule, as originally specified by authors, was not sufficiently clear or precise, we redefined it, whilst maintaining its original intent as we understood it, thus hopefully facilitating comparisons and the consolidation of rules. We also recorded the papers that introduced each rule, which can, for instance, be used to identify the rules that are the most important to users; (RQ1).
- Which of the 14 UML diagrams are involved in the rule (RQ3): 1) Class Diagram (*CD*); 2) Communication Diagram (*COMD*) or Collaboration Diagram (*COD*); 3) Use Case Diagram (*UCD*); 4) State Machine Diagram (*SMD*) or Protocol State Machine Diagram (*PSMD*); 5) Sequence Diagram (*SD*); 6) Component Diagram (*CTD*); 7) Object Diagram (*OD*); 8) Profile Diagram (*PRD*); 9) Activity Diagram (*AD*); 10) Composite Structure Diagram (*CSD*); 11) Interaction Overview Diagram (*IOD*); 12) Package Diagram (*PD*); 13) Timing Diagram (*TD*); 14) Deployment Diagram (*DD*).

  UML terminology has changed over the years and the following pairs of UML diagrams were, therefore, considered and counted (in the statistical reports presented in this research) as only one diagram:

  - ○ COD and COMD (diagram number 2 in the above list) were counted as one diagram since COMD is the UML 2.x equivalent of the COD in UML 1.x (Pender, 2003).
  - ○ PSMD is a kind of SMD (diagram number 4 in the list above) (OMG, 2011). Although we initially decided to collect the information for these two diagrams separately since they have different purposes during software development, both diagrams

were eventually reported as only one diagram.

- UML consistency dimensions and types of the rule (see Appendix B) (RQ2).
- The software engineering activity that required UML diagrams to be in some way consistent with one another, as specified in the rule (RQ5). The following eight definitions describe activities in software engineering in which UML diagram consistency was required, as collected in the primary studies of our SMS (Torre et al., 2014). Since this is a list of activities from the primary studies we found, this list is not meant to exhaustively represent all the software engineering activities that require diagrams to be consistent:
  - ○ The verification of consistency (*Verification*) is the process of detecting inconsistencies between (or in a singular) UML diagram (s) during software development. (This definition is in line with the IEEE definition of verification (IEEE, 1992)).
  - ○ Consistency management (*Management*) includes the detection of inconsistencies in a software model, the diagnosis of the significance of inconsistencies and the handling of detected inconsistencies (Spanoudakis and Zisman, 2001).
  - ○ Model Refinement and Transformation (*Ref. & Tra.*) are defined as follows (Paige et al., 2005):
  - ■ Transformation (also called mapping), which takes a model as input and generates another model as output, e.g., transforming a Platform Independent Model (PIM) into a Platform Specific Model (PSM), or a PSM into executable code. These transformations generally add more detail to the output model.
  - ■ Refinement, which is a semantics-preserving transformation applied to a model, and which changes the model in place, e.g., refining a Platform Independent Model (PIM) into a more detailed PIM;
  - ○ Safety and Security consistency (*Saf. & Sec.*): From the point of view of automated analysis, safety consistency implies that there are no conflicting requirements and unintentional (internal) nondeterminism in a UML diagram (Pap et al., 2005). Security consistency is, meanwhile, defined as the consistency of bounded values in non-abstract elements of a UML diagram (Pilskalns et al., 2006).
  - ○ Impact Analysis (*Impact analysis*) is defined as the process of identifying the potential consequences (side-effects) of a change to the model, and estimating what else needs to be modified/tested (Briand et al., 2003).
  - ○ Model formalization (*Formalization*) describes the formal process used to develop a UML model (Song et al., 2008), (Yang, 2009).
  - ○ Model understanding (*Understanding*) is the process employed to understand that a UML model is much more than a set of annotated boxes and lines, as it reflects the meaning of a piece of software, that is, the UML has semantics. The semantics of the UML is expressed through its metamodel and its so-called well-formedness rules and context/domain dependent consistency rules. These rules describe in plain language, and often using the Object Constraint Language (OCL), what constraints UML model elements have to satisfy (Labiche, 2008).
- The three software development methodologies (SDM)(RQ6):
  - ○ SDM#1: according to (Arlow and Neustadt, 2005; Hunt, 2003) the four Unified Process phases and associated UML diagrams are:
    - ■ Requirements: This phase focuses on the activities which allow the functional and non-functional requirements of the system to be identified. The primary product of this phase is the use case model. In this phase, the activities are focused on finding actors and UCD, prioritizing use cases, detailing use cases, prototyping user interface, structuring the use case model, etc.
    - ■ Analysis: The aim of this phase is to restructure the requirements identified in the requirements discipline in terms of the software to be built rather than in the user's less precise terms. In this phase SD, COD, OD, CD, PD, and AD are used.
    - ■ Design: This produces the detailed design which will be implemented in the next phase. This phase focuses on tracing use cases, refining and designing classes. CD and SMD are used.
    - ■ Implementation: This represents the coding of the design in an appropriate programming language, and the compilation, packaging, deployment and documenting of the software. The DD and CTD are used in this phase.

    The following diagrams are not considered in SDM#1: IOD, CSD and TD.

  - ○ SDM#2: Dathan and Ramnath (Dathan and Ramnath, 2015) present five object-oriented software development phases along with each of the UML diagrams involved:
    - ■ Requirements model: Describes the boundary and interaction between the system and users by employing UCDs.
    - ■ Interaction model: Describes how objects in the system will interact (i.e., by using COMD, SD and TD). The Interaction model is useful when we want to flesh out the details of the all the required behaviors.
    - ■ Dynamic model: This defines how the interaction occurrences come together in a definition of the system and are, therefore, a part of the dynamic model of the system. An SMD describes the states or conditions that a class assumes over time. An AD describes the workflows that the system will implement. An IOD can be also used in this phase.
    - ■ Logical model: Describes the classes and objects that will make up the system, i.e., it describes what software entities have to be created by using CD, CTD, CSD, PD, and OD.
    - ■ Deployment model: Describes the physical architecture and the deployment of components (i.e., the DD) on that hardware architecture.
  - ○ SDM#3: Bruegge and Dutoit (Bruegge and Dutoit., 2009) present the following three object-oriented software development phases along with each of the UML diagrams involved:
    - ■ The functional model: represented in UML with UCD. This describes the functionality of the system from the user's point of view.
    - ■ The object model: represented in UML with CD, OD, and DD. This describes the structure of the system in terms of objects, attributes, associations, and operations.
    - ■ The dynamic model: represented in UML with SD, COMD, SMD and AD. This describes the internal behavior of the system. SD and COMD describe the behavior as a sequence of messages exchanged among a set of objects, whereas SMD describes behavior in terms of states and the possible transitions between states. AD describes behavior in terms of control and data flows.

The following diagrams are not considered in SDM#3: IOD, CTD, PRD, CSD, TD, and PD.

None of the three methodologies above involve PRD, which is used to adapt the UML metamodel to different platforms and domains.

4. Execution of the SMS

In this section, we report on the results obtained, in terms of the number of rules collected, after applying the protocol discussed in Section 3. Please recall that we started from the primary studies collected in our SMS (Torre et al., 2014), which began in September 2012 and was completed in October 2013. From an initial set of 1603 papers, 95 primary studies papers were eventually identified. The execution of the rule selection protocol of our SMS (Torre et al., 2014) began in May 2014 and was completed in October 2015.

However, by this point, a considerable amount of time had passed, and we decided to update this SMS (Torre et al., 2014). A second phase was, therefore, planned and executed. This phase began in January

Table 4
Summary of the answers to the RQs.

Research Question 1: What are the existing UML consistency rules? (see Appendix A)											
Research Question 2: Which types of consistency problems are tackled in the existing rules?											
Horizontal	99	Invocation	9	Vertical	7	Evolution	3				
Observation	1	Syntactic	97	Semantic	22	–	–				
Research Question 3: What types of UML diagrams are involved in UML consistency rules?											
1	SMD	15	**10**	UCD and SMD	0	**19**	COMD	1	**28**	CD	35
2	CD, SMD and AD	1	**11**	CD and SMD	7	**20**	UCD and CD	3	**29**	ID and CD	0
3	SD and UCD	5	**12**	PSMD and CD	0	**21**	OD and AD	0	**30**	ID and UCD	2
4	AD	0	**13**	SD and AD	7	**22**	COMD and AD	1	**31**	PSMD and SMD	0
5	SD	6	**14**	UCD and OD	0	**23**	COMD and CD	3	**32**	CSD	8
6	OD and COMD	0	**15**	SMD and COMD	1	**24**	COMD and UCD	0	**33**	SD and CD	9
7	OD and CD	1	**16**	COMD and CD	1	**25**	UCD and AD	4	**34**	SD and COMD	0
8	AD and CD	4	**17**	SMD and SD	1	**26**	UCD	4	**35**	CD, SMD and COMD	0
9	SMD and AD	0	**18**	PSMD	0	**27**	SMD and OD	0	**36**	UCD, SD and CD	0
Research Question 4: For what software engineering activities are UML consistency rules used?											
Verification	253	Ref/ &Tra.	54	Understanding	21	Saf. & Sec.	7				
Impact Analysis	57	Management	72	Formalization	14	–	–				
Research Question 5: Which UML diagram elements are involved in UML consistency rules?											
class/es	161	action/s	18	invariant/s	9	capsule/es	4				
state/s	86	collaboration	17	interface/s	9	class name	3				
operation/s	56	chart/s	17	interaction	9	super class	2				
association/s	45	machine/s	17	sender	8	sub sequence	2				
use case	41	pre-condition	16	flow	8	sub classes	2				
activity/ies	39	Strings	15	composite	8	scenario	2				
message/s	38	Link	13	property	7	query	2				
sequence/s	38	event/s	12	connector/s	7	post-state	2				
object/s	36	guard/s	12	call	7	package	2				
instance/s	30	Visibility	11	node	6	swimlanes	1				
name/s	26	aggregation	11	generalization/s	6	stimulus	1				
transition/s	26	post-condition/s	10	constraint/s	6	stimuli	1				
type/s	23	method/s	10	port	5	name pace	1				
attribute/s	19	Receiver	10	parameter	5	edge	1				
behavior	18	multiplicity	9	lifeline	4	actor	1				
Research Question 6: What software development phases are involved with UML consistency rules?											
SDM#1	Requirement	38	Design	398							
	Analysis	424	Implementation	0							
SDM#2	Requirements model	38	Logical model	294							
	Interaction model	233	Deployment model	0							
	Dynamic model	170	–	–							
SDM#3	Functional model	38	Dynamic model	398							
	Object model	424	–	–							

2017 and was completed in April 2017 and included all the papers regarding UML consistency rules that were published from 2013 to March 2017. We extracted 865 additional papers, which were analyzed and classified using the same process as described in Section 3 of Appendix B. Finally, 10 new primary studies were added to the initial 95. The execution of the rule selection protocol for the 10 new primary studies began in June 2017 and was completed in September 2017.

The two phases (the initial SMS (Torre et al., 2014) and its update) eventually resulted in 105 papers as primary studies to be analyzed.

The primary studies selected span the period from 2000 to 2016 (no primary study was found in the first three months of 2017). The list of primary studies can be found elsewhere (Appendix B of our online technical report (Torre et al., 2015a)).

We then started to study the 105 primary study papers, which allowed us to identify 687 rules. In order to document the execution activity of the protocol (Section 3) in sufficient detail, we describe the following multi-tasked process:

- First task (T1): 687 rules were specified from all the 105 primary study papers. Each of the 105 papers was analyzed and the rules were manually collected according to the inclusion and exclusion criteria specified in Section 3 of Appendix A. This phase took approximately 3 months and was performed by the first author and supervised by the second.
- Second task (T2): we obtained a reduced set of rules by deleting all those that were, according to the exclusion criteria, inaccurate. We identified 209 inaccurate rules and obtained a set of 478 accurate rules. The first and second authors were responsible for performing this phase, which took approximately 2 months. For a rule to be included, the first and second authors had to be in agreement. In those cases, in which the two authors did not agree on the inclusion or exclusion of the rule, the fourth author was contacted to discuss the appropriate action.
- Third task (T3): we further reduced the set of rules by deleting all those that were, again according to the exclusion criteria, redundant. We identified 280 redundant rules and obtained a set of 198 accurate and non-redundant rules. This phase took approximately 1 month and was conducted by the first and second authors in the same manner in which T2 was performed.
- Fourth task (T4): we further reduced the set of rules by deleting those already presented in previous UML standards, again according to the exclusion criteria. We identified 79 rules that were already in the standard (i.e., well-formedness rules) and obtained a set of 119 rules that are accurate, non-redundant and not already in the standard. In order to exclude these 79 rules from the 198 (39.9%) non-redundant accurate rules, we compared them with the applicable UML specification version at the time the rules were published (we either found a reference to the applicable version in the publication or inferred it from the time of publication). Finally, we also checked whether each rule was contained in the most recent version of the specification (UML 2.5). The 79 rules were excluded because they had already been presented in one of the previous UML standards

(UML 1.3, 1.5, 2.0, 2.4.1, and 2.5). This phase took approximately 1 month and was conducted by the first and second authors.

In summary, the 687 rules were identified from the 105 primary study papers by the first author under the supervision of the second. The first and second authors then applied the exclusion criteria described in Section 3.2, and contacted the fourth author in those cases in which the two authors did not agree on the inclusion or exclusion of the rule. Finally, 119 unique rules were identified. The entire systematic procedure applied to this study was performed by the first author under the supervision of the second and third authors.

The set of 478 accurate rules (T2), and the final set of 119 rules (T4), were then classified according to the data extraction strategy discussed in Section 3.2.

5. Results

The data recorded on our data extraction form (an excel file) allowed us to answer the six research questions discussed in Section 3.1. A quantitative summary of the results for research questions RQ2 to RQ6 is presented in Table 4. More details are provided in the following sub-sections, in which we shall refer to Table 4 for raw data.

Two different sets of rules were used to answer our six research questions:

1) the final set of 119 consistency rules (T4 in Section4) to answer RQ1, RQ2 and RQ3;
2) the set of 478 accurate rules (T2 in Section 4) for RQ4, RQ5 and RQ6. We chose to include the 280 redundant rules (T3 in Section 4) and the 79 rules already presented in the UML standard (T4 in Section 4) to answer these three research questions, because we wish to present the whole picture of what researchers have focused on.

With regard to Table 4 (first raw), the reader is requested to simply refer to Appendix A to obtain the answers for research question RQ1, since the purpose of this question is to report on the consistency rules we have coalesced. The section of Table 4 that contains data for research question RQ3 shows all the combinations of UML diagrams for which we have found consistency rules in the primary studies, and the number of consistency rules for each combination after conducting the protocol discussed in Section 3 (i.e., after applying inclusion/exclusion criteria). That is to say, we have found 14 rules involving only the state machine diagram (SMD), 5 rules involving the simultaneous use of sequence and use case diagrams (SD and UCD), and 1 rule involving the simultaneous use of class, state machine and activity diagrams (CD, SMD and AD). The primary studies contained rules for 36 different diagram combinations, i.e., a small subset of all the possible combinations of UML diagrams. Note also that the table shows combinations with no rule (e.g., UCD and SMD), signifying that we found rules in the primary studies that involve these diagrams, but that the rules were eventually discarded because of our inclusion/exclusion criteria. The first column shows the unique ID# we gave to each of the 36 UML diagram combinations; these unique IDs will later be used in figures to simplify the data analysis. The second column describes the 36 UML diagram combinations, which refer to: 1) rules involving only one diagram, such as AD, SD etc.; 2) rules involving pairs of diagrams, such as SD and UCD, and 3) rules involving 3-tuples of diagrams, such as CD, SMD and AD. Not all of the 14 UML diagrams appear in Table 4, since a diagram (or a pair or a 3-tuples of diagrams) is not shown if we were unable to find a rule involving that (those) diagram(s) in the primary studies.

5.1. What are the existing UML consistency rules? (RQ1)

The principal observation we can make about RQ1 is that the researchers who have developed UML consistency rules have typically defined a number of similar rules over and over again. Specifically, we collected a list of 687 rules from the 105 primary studies. After removing rules that were inaccurate, redundant, and already presented in the UML standards, we obtained a final set of 119 rules (see Table 6 in Appendix A). In other words, only 17.32% (119 out of 687) of the rules initially collected are included in our final set of rules because of our inclusion/exclusion criteria (Section 3.2). The remaining rules were eliminated because they were inaccurate: 30.42%, 209 out of 687, resulting in 478 accurate rules. Specifically, 96 (13.97%) were not consistency rules (e.g., rules describing good modeling practices); the language in which 28 (4.50%) were explained was too ambiguous; 14 (2.25%) were obsolete because they referred to UML elements that are no longer used in the latest UML standard, and 71 (10.33%) were simply wrong (i.e., they contradicted the UML specification). Other rules were mostly eliminated because of redundancies (58.58%, 280 out of 478). Finally, 79 out of 198 (39.9%) non-redundant rules were excluded because they are already part of the UML standard.

The rules with the largest number of references in primary studies are rule 112 (with 53 references), and rules 110, 48, and 115 (with 38, 30, and 18 reference each, respectively). In a nutshell, rule 112 focuses on the required visibility of some class diagram elements in order to exchange messages in a sequence diagram; rule 110 concerns the consistency of messages in a sequence diagram and the class diagram; rule 48 specifies the consistency of behavior specification in the lifelines of a sequence diagram and a state machine of the corresponding classifiers, and rule 115 describes the consistency of class names in class diagrams and sequence diagrams. The complete list of references for the 119 rules presented in Table 6 (see Appendix A) is presented elsewhere owing to space limitations (Appendix A of our online technical report (Torre et al., 2015a)).

5.2. Which types of consistency problems are tackled in the existing rules? (RQ2)

The results obtained for RQ2 show (Table 4) that the vast majority of rules are Horizontal (83.19%, 99 out of 119 rules) and Syntactic (81.51%, 97 out of 119 rules). Moreover, we found 22 (18.48%) Semantic rules. Researchers strikingly described many more syntactic than semantic consistency rules. We conjecture that the main reason for this is that syntactic rules are easier to specify than semantic rules and that the UML standard more formally specifies syntax than semantics, which hinders the specification of semantic rules. It may also be the case that semantic rules are specific to how the UML notation is actually used, which is organization, project, or team specific, and are, therefore, seldom described in published manuscripts.

Researchers have also paid much less attention to: 1) Invocation rules (9 rules, 7.56%); 2) Vertical rules (7 rules, 5.88%); Evolution rules (3 rules, 2.52%) and Observation rules (only one rule, 0.84%). We believe that the mappings between vertical levels, the evolution of a UML model, and the invocation and observation consistency are concepts which are much less easy to understand than the mere specification of a UML model at the horizontal consistency level, and this explains why, despite finding papers discussing these consistency dimensions, these papers did not present many consistency rules.

5.3. What types of UML diagrams are involved in UML consistency rules? (RQ3)

Columns (a) of Table 5 show the mapping of the excluded rules onto the 36 combinations of the UML diagrams presented in Table 4: the combinations are numbered vertically in the table (first column: combo ID). The last column of the group of columns (a) is a sum. Columns (b) report on the rules we kept, and the diagram(s) they involve, with a total in the last (far right) column for each diagram(s) combination. For instance, in Columns (b) the total of combo 2 is 1 because we have only

Table 5
Summary of the excluded and accepted rules.

Combo ID	Total	(a) Details of excluded rules							(b) Rules we kept and grouped by UML diagrams							
		Ambiguous	Standard	Obsolete	No consistency	Incorrect	Redundant	Total	ID	UCD	CD	OD	AD	CSD	SMD	Total
1	69	10	24	0	8	4	8	**54**	–	–	–	–	–	–	15	15
2	1	0	0	0	0	0	0	**0**	–	–	1	–	1	–	1	1
3	10	0	0	0	0	4	1	**5**	5	5	–	–	–	–	–	5
4	7	0	4	0	1	2	0	**7**	–	–	–	–	–	–	–	0
5	51	3	7	0	9	1	25	**45**	6	–	–	–	–	–	–	6
6	3	0	0	0	0	3	0	**3**	–	–	–	–	–	–	–	0
7	3	0	1	0	0	0	1	**2**	–	–	1	1	–	–	–	1
8	15	0	0	0	3	0	8	**11**	–	–	4	–	4	–	–	4
9	4	0	0	0	0	2	2	**4**	–	–	–	–	–	–	–	0
10	3	0	0	0	0	3	0	**3**	–	–	–	–	–	–	–	0
11	67	2	3	1	8	7	39	**60**	–	–	7	–	–	–	7	7
12	1	0	1	0	0	0	0	**1**	–	–	–	–	–	–	–	0
13	14	0	0	0	0	4	3	**7**	7	–	–	–	7	–	–	7
14	1	0	0	0	0	1	0	**1**	–	–	–	–	–	–	–	0
15	4	0	0	0	1	1	1	**3**	1	–	–	–	–	–	1	1
16	3	1	0	0	0	0	1	**2**	1	–	1	–	–	–	–	1
17	43	2	1	0	1	9	29	**42**	1	–	–	–	–	–	1	1
18	1	0	1	0	0	0	0	**1**	–	–	–	–	–	–	–	0
19	8	0	2	0	4	0	1	**7**	1	–	–	–	–	–	–	1
20	15	0	0	0	0	0	12	**12**	–	3	3	–	–	–	–	
21	2	0	0	0	0	2	0	**2**	–	–	–	–	–	–	–	0
22	3	0	0	0	0	2	0	**2**	1	–	–	–	1	–	–	1
23	20	0	1	4	0	1	11	**17**	3	–	3	–	–	–	–	3
24	3	0	0	0	1	2	0	**3**	–	–	–	–	–	–	–	0
25	14	0	0	0	2	4	4	**10**	–	4	–	–	4	–	–	4
26	10	1	0	0	4	0	1	**6**	–	4	–	–	–	–	–	4
27	1	0	0	0	0	1	0	**1**	–	–	–	–	–	–	–	0
28	150	1	30	7	24	19	34	**115**	–	–	35	–	–	–	–	35
29	1	1	0	0	0	0	0	**1**	–	–	–	–	–	–	–	0
30	4	0	0	0	0	0	2	**2**	2	2	–	–	–	–	–	2
31	2	1	1	0	0	0	0	**2**	–	–	–	–	–	–	–	0
32	8	0	0	0	0	0	0	**0**	–	–	–	–	–	8	–	8
33	140	5	3	1	21	4	97	**131**	9	–	9	–	–	–	–	9
34	2	0	0	0	0	2	0	**2**	–	–	–	–	–	–	–	0
35	1	0	0	1	0	0	0	**1**	–	–	–	–	–	–	–	0
36	3	1	0	0	0	2	0	**3**	–	–	–	–	–	–	–	0

one rule that involves CD, AD and SMD. The column immediately to the left of the combo ID column is the overall total of all the rules for that diagram(s) combination: it is the sum of the (a) columns and the sum of the (b) columns. For instance, for combination combo ID 1 (i.e., SMD, as per Table 4, i.e., State Machine Diagram as per Section 3.3), the initial set of rules involving only the State Machine Diagram contained 69 rules (15 + 54), of which we removed 54 and kept only 15. Columns (a) indicate the reasons for deleting the rules and the magnitude of each reason for each combination of diagrams. For instance, of the 54 deleted rules involving only the State Machine Diagram, ten were deleted because they were ambiguous (beyond repair), 24 were already in the standard, eight were redundant (with rules that we kept), eight were not consistency rules, and four were incorrect.

The results show that the diagram combination that involves the largest number of rules in the initial set, specifically 150 rules (35 + 115), is combination 28, i.e., the Class Diagram: the majority of the rules involve only the Class Diagram. This is followed by combination 33, i.e., rules involving the Class Diagram and the Sequence Diagram, with 140 rules (9 + 131). These two sets of rules make up 40.74% (290 out of 687 rules) of the total number of initial rules, and 36.97% of the final set of rules (44 out of 119). It is also important to mention that there is a huge gap between these two most frequently involved diagram combinations and the next most frequently involved diagram combinations, which are the State Machine Diagram (69 rules, combo 1), Class Diagram with State Machine Diagram (67 rules, combo 11), Sequence Diagram (51 rules, combo 5), and State Machine Diagram with Sequence Diagram (43 rules, combo 17).

The absence of rules for some diagrams or some diagram combinations can be explained by the fact that the semantics and syntax of some diagrams overlap (e.g., package, object and class diagrams). However, it is possible to argue that the fact that the semantics and syntax of those diagrams overlap, and that there is a need for these different diagrams, should justify the definition of rules to ensure that the diagrams are consistent, unless all the rules required are already in the UML standard (which is unlikely). Another possible reason for a lack of rules for some diagrams (or diagram combinations) is that users of the UML notation follow a specific UML modeling methodology that is assumed to be standard and does not, therefore, require the definition of consistency rules. Nevertheless, since the UML allows different modeling methodologies, rules underpinning such methodologies should be specified so as to be clear for all stakeholders.

Another interesting finding is the high percentage (88.33%) of rules eliminated in the case of the Sequence Diagram: 45 out of 51, for combination ID 5. If we consider all the UML diagram combinations in which the Sequence Diagram is involved (combo ID 3, 5, 13, 17, 33 and 34), we notice that, when starting with the total of 260 rules collected, only 28 (10.77%) made it to the final set of 119 rules. The main reason for discarding rules was redundancy, i.e., rules presented several times by authors: for instance, 25 of the 45 discarded rules involving only the Sequence Diagram (combo ID 5) were redundant. One possible explanation for this is that the sequence diagram is one of the most ill-specified diagrams in the UML specification and researchers are attempting to make sense of it by specifying rules which, when coalesced, happen to be redundant with one another (155 of the 232 eliminated rules, 66.81%, were redundant with other rules).

In this section, we discussed the 36 different combinations of UML diagrams (detailed in Table 4) covered by the final set of 119 rules. Our results show that the rules collected describe consistency in only eight

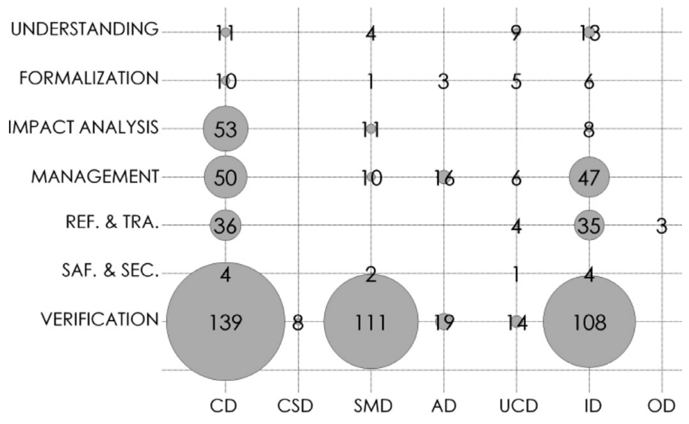

Fig. 1. Summary of rules between UML diagrams and uses in Software Engineering activities.

of the 14 UML diagrams. We did not find any rule involving the Package, Component, Profile, Timing, Interaction Overview and Deployment diagrams. We conjecture that there are several reasons for the lack of rules involving these diagrams. First, as mentioned previously, the fact that the Package Diagram and the Deployment Diagram overlap with the Class Diagram might be a reason for the lack of rules involving these diagrams. However, as argued previously, this similarity between the diagrams could be the very reason needed to justify more rules involving these diagrams or the Package Diagram with other diagrams. The lack of widespread tool support and underspecification/ambiguity in the specification for the Timing and Interaction Overview diagrams is a possible reason for the lack of rules involving these diagrams.

The right-hand side of Table 5, columns (b), shows only seven diagrams because we grouped the rules for the Sequence, Communication, Interaction Overview and Timing diagrams together in the Interaction Diagram (ID) category, since these diagrams are different variants of the ID (OMG, 2011).

Table 5 (b) shows the rules involved in each of the seven diagrams: for instance, in combo ID 8 we identified four rules that involve the Class Diagram (CD) and the Activity Diagram (AD). These data are taken directly from Table 4, although they are presented in a different form. For instance, the Activity Diagram (AD) was found to be involved in 17 rules (1 + 4 + 7 + 1 + 4). Table 4 indicates that one rule involves AD with CD and SMD (combination number 2), four rules involve AD and CD (combination 8), seven rules involve AD and SD (combination 13), one rule involves AD and COD (combination 22) and four rules involve AD and UCD (combination 25). The total number of rules found that involve each diagram resulted in 37 rules for ID, 18 rules for UCD, 64 rules for CD, one rule for OD, 17 rules for AD, eight rules for CSD, and 25 rules for SMD. Note that, upon summing up the rule count of each of the seven diagrams totals, we obtain 170 rather than 119, because when a rule involves more than one diagram it contributes to more than one sum.

Not surprisingly, the UML diagrams that are most involved in the final set of 119 rules are the Class Diagram (64 rules, 53.78%), the Interaction Diagram (37 rules 31.09%), and the State Machine Diagram (25 rules, 21.01%). Research on UML consistency rules has paid much less attention to the Use Case Diagram (18 rules, 15.13%) and the Activity Diagram (17 rules, 14.29%). The diagrams that are least covered are the Composite Structure Diagram and the Object Diagram, which only had 6.72% (eight rules) and 0.84% (one rule) of the total of 119 rules, respectively.

5.4. *For what software engineering activities are UML consistency rules used? (RQ4)*

In this section, we discuss the results of the different software engineering uses (i.e., activities) of rules. As explained earlier in Section 5, we answered this RQ by considering the set of 478 rules rather than only the final set of 119 rules, because redundant rules may have been defined in the context of different Software Engineering activities.

The results show that 52.93% (253 out of 478) of the rules are proposed for model consistency verification, 15.06% (72 out of 478) for consistency management, 11.92% (57 out of 478) for impact analysis, 11.3% (54 out of 478) for model refinement and transformation, 4.39% (21 out of 478) for model understanding, 2.93% (14 out of 478) for model formalization, and 1.46% (7 out of 478) for safety and security consistency.

In the light of these figures, the reader may be surprised to discover that more than half of the consistency rules have been defined without any additional context of use, such as specific model-driven activities. This may suggest that not many UML-based software engineering activities require UML diagrams to be consistent, which would be surprising, or at least that the papers describing UML-based software engineering activities do not discuss (or need to rely on?) consistency rules. This may rather mean that typical UML-based software engineering activities require the same set of consistency rules, which can, therefore, be described independently of their context of use. We believe that the most likely reason is the latter.

5.5. *Combining RQ4 and RQ3*

In this section we combine the data presented for RQ4 in Section 5.4 with the data presented in the analysis of RQ3 in section 5.3 to investigate the relationship between the rules collected for each UML diagram and the different Software Engineering activities, i.e., in order to understand what, for each UML diagram, are the Software Engineering activities for which most rules where proposed. In Fig. 1 and Fig. 2, we use bubble plots to represent multiple dimensions of the results in one figure: the bubble plot essentially embodies a two x–y scatter plot with bubbles at the category intersections. This synthesis method is useful, since it provides both a map and a rapid overview of a research field (Petersen et al., 2008).

The bubble plot in Fig. 1 shows that the Class Diagram (CD) is mostly involved in rules used for verification (139 rules), impact analysis (53 rules) and management (50 rules) activities. Furthermore, the State Machine Diagram (SMD) and the Interaction Diagram (ID) have 111 and 108 verification rules, respectively. This is not entirely surprising, since these three UML diagrams are likely those most frequently used in these activities.

Fig. 1 shows a total of 751 rules rather than 478 accurate rules because we grouped all the rules related to each of the seven UML diagram separately, resulting in some rules being repeated (e.g., a rule involving two diagram types counts twice).

In Fig. 2, we present a bubble plot distribution that was obtained by combining the results of RQ3 and RQ4 with the years in which the rules

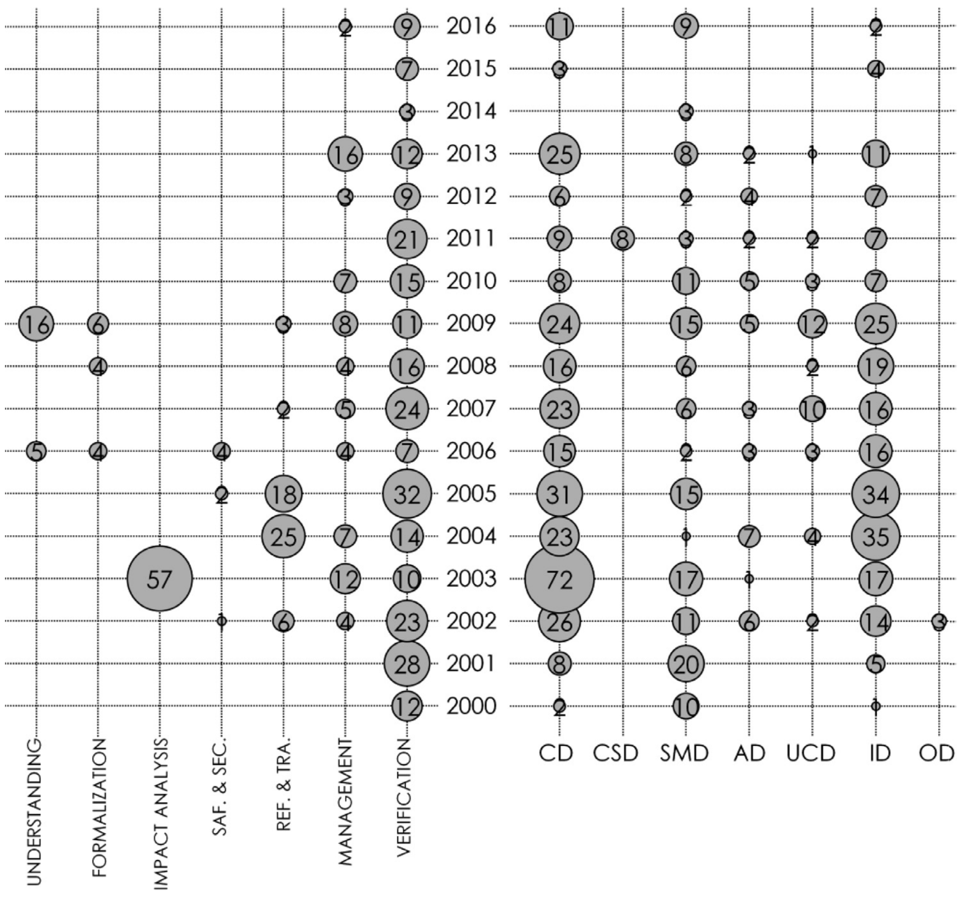

Fig. 2. Summary of rules between UML diagrams presented over the years, coupled with diagrams and Software Engineering activities.

were published (Appendix B of our online report (Torre et al., 2015a)). By combining the latter research questions with the years, we aim to show the complete overview of the distribution of the set of 478 rules according to UML diagrams and Software Engineering activities and how they span the last 16 years; for instance, we wish to identify in which period/year most rules were proposed for each UML diagram and Software Engineering activity. In Fig. 2, we coupled the year in which a rule was published with the UML diagram covered (right-hand side), and with the Software Engineering activity that used the rule (left-hand side).

The right-hand side of Fig. 2 shows the number of rules presented in the period between 2000 and 2016, classified by the seven UML diagrams covered by the rules. As in the previous two sections, we grouped all the rules related to each of the seven UML diagrams, resulting in 749 rules rather than 478 rules, since some rules were repeated. For instance, the seven rules presented for CD and SMD (combination 11 in Table 4) were included in both the CD count and in the SMD. However, the left-hand side of Fig. 2 shows only 478 rules in the same period classified in the eight Software Engineering activities presented in section 3.3. No rule is repeated because each of the 105 primary study papers (Torre et al., 2014) was assigned to only one of the eight Software Engineering activities presented in Section 3.3. The rules presented in each paper were consequently classified according to only one of these seven activities.

The results show (left-hand side of Fig. 2) that a large number of rules were proposed in four of the 16 years considered in this study: 44.14% (211 out of 478 rules) of the rules were published between 2002 and 2005. In these four years, 152 out of 302 (50.33%) of the total number of rules were related to the Class Diagram and 100 out of 220 (45.45%) of the total number of rules were related to the Interaction Diagram (right-hand side). Moreover, in these four years we can observe three peaks of Software Engineering activity use (left-hand side): 1) 57 rules presented in 2003 for impact analysis, 32 rules in 2005 for verification, and finally, 25 rules in 2004 for refinement and transformation. We can conclude that these four years represent the most prolific period in literature in terms of publishing UML consistency rules to date.

Another important aspect that should be highlighted is that the number of UML consistency rules covering the Verification activity (left-hand side of Fig. 2) remained relatively stable throughout the entire period, with a peak of 32 new rules in 2005. Furthermore (right-hand side of Fig. 2), the Class and Interaction Diagrams remained stable in the period between 2002 and 2009, with a peak of 72 rules in 2003 and 35 rules in 2004, respectively. The fact that researchers consistently continued to focus on those diagrams during these years shows the vital importance of these diagrams.

5.6. Which UML diagram elements are involved in UML consistency rules? (RQ5)

Fig. 3 shows the 200 most frequently used words in the plain English specification of the 119 rules we found (4761 words in total), in which we can recognize the 60 UML element names already presented in Table 4. This is a weighted word cloud of the 119 rules found in the 105 papers included as primary studies in our SMS (Torre et al., 2014). Fig. 3 was created using Tagxedo,[1] which uses a streaming algorithm to filter the text input. The aim of Fig. 3 is to provide a first impression of the main UML diagram elements referenced in the final set of 119 rules. Not surprisingly, the UML elements most involved in the rules are

[1] http://www.tagxedo.com/

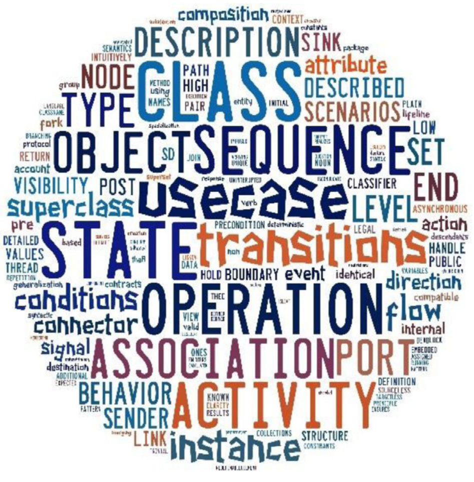

Fig. 3. Summary of the UML diagram elements involved in UML Consistency.

related to the class, sequence, state machine, use case and finally activity diagrams. These results concerning the model elements most frequently involved in rules may be useful to help focus teaching activities on specific elements. Indeed, if an element is not rigorously specified, then there is a risk of many issues related to semantics appearing in the model.

5.7. *What software development phases are involved with UML consistency rules? (RQ5)*

UML is not tied to a specific software development method (Brambilla et al., 2016). However, the language concepts of UML are designed to support an iterative and incremental process (Rumbaugh et al., 2004), such as the Unified Process (Arlow and Neustadt, 2005), or others, such as object oriented software development methodologies (for example (Dathan and Ramnath, 2015) and (Bruegge and Dutoit., 2009)).

In this section, we present the results of how the 478 UML consistency rules are involved with the three different software development methodologies presented in Section 3.3: SDM#1, SDM#2 and SDM#3. We chose to follow these SDMs for the following reasons: the Unified Process (Hunt, 2003) (i.e., the SDM#1 (Arlow and Neustadt, 2005; Hunt, 2003)) is an iterative, incremental use case driven process and was selected because the authors of UML suggested that it be used in a development process of this type (Rumbaugh et al., 2004); SDM#2 was chosen because its authors (Dathan and Ramnath, 2015) suggested it should be used with the most recent version of UML, 2.5 (OMG, 2015); SDM#3 was chosen from the book by Brugge and Dutoit (Bruegge and Dutoit., 2009),which we believe, with its third edition, represents a reliable source for researchers and practitioners alike who work in the field of object-oriented software engineering and UML.

We consider the software development phases of the three software development methodologies, in which at least one UML diagram was used in each of these phases. Each of the 478 rules involved one or more UML diagrams, which was assigned to be part of one or more SDM phases. For instance, rule 16 (see Table 6) involves SD and UCD. If we consider SDM#2, this rule is consequently involved in both requirements and interaction models. When a rule involves two or more UML diagrams and two or more diagrams involve the same phase, the two or more same phases were counted as only one phase. For instance, if we consider SDM#1, rule 15 (see Table 6) involves CD (involved in analysis and design), SMD (involved in design), and AD (involved in analysis), and this rule was, therefore, counted only once for both analysis and design rather than twice for each phase.

In Table 4, we presented the raw results for each of the three SDMs considered. Here we describe the results in detail:

- SDM#1: 1) Requirement: 38 rules out of 478 (7.95%); 2) Analysis: 424 rules out of 478 (88.7%); 3) Design: 398 rules out of 478 (83.26%); 4) Implementation: 0 rules. The sum of the four SDM#1 phases return a total of 860 rules rather than 478 accurate rules because some rules were grouped in two or more different phases (e.g., a rule involving two diagram types and two different phases may count twice). CD was counted for both the analysis and design phases for each rule this UML diagram was involved in.
- SDM#2: 1) Requirements model: 38 rules out of 478 (7.95%); 2) Interaction model: 233 rules out of 478 (48.74%); 3) Dynamic model: 170 rules out of 478 (35.56%); 4) Logical model: 294 rules out of 478 (61.51%); 5) Deployment model: 0 rules. The sum of the five SDM#2 phases returns a total of 735 rules rather than 478 accurate rules because some rules were grouped in two or more different phases.
- SDM#3: 1) Functional model: 38 rules out of 478 (7.95%); 2) Object model: 424 rules out of 478 (88.7%); 3) Dynamic model: 398 rules out of 478 (83.26%). The sum of the three SDM#3 phases returns a total of 701 rules rather than 478 accurate rules because some rules were grouped in two or more different phases.

Since the CSD was not included in the SDM#1 and SMD#3, and considering that, according to the UML specification (OMG, 2015), the constructs used to describe CD are the same as those used for CSD (i.e., Structured Classifier), we counted the rules that involve CSDs as they involve CDs instead. It is possible to conclude that, assuming the three software development methodologies we selected are to a great extent representative of a large set of users of the UML, the rules we collected are widely applicable. The list of consistency rules we report on in this manuscript should, therefore, be of interest to a very large community of educators, researchers and practitioners.

6. Threats to validity

In the following subsections, we analyze the potential threats to the validity that could affect our study according to the four categories suggested by Wohlin et al. (2012).

6.1. *Construct validity*

Construct validity concerns the relation between the measures used in the study and the theories on which the research questions are grounded (Borg et al., 2014). In this SMS, this concerns the systematic identification of UML consistency rules, which is inherently dependent on the coherence of the terminology in the field. In order to mitigate these threats, we described the terminology and concepts used to measure the frequency of UML consistency rules regarding the different UML consistency issues. We collected the concepts and terminology that were used to carry out those measurements from our previous systematic study (Torre et al., 2014), which focused on UML consistency. We observed that the terminology used by researchers is not always consensual and that authors in this field sometimes discuss UML consistency issues using different terms such as “Horizontal Consistency” and "Intra-model Consistency" to refer to the same concept. We used these synonyms when searching for the primary studies. Among other issues, researchers may use a different terminology for a particular topic (Wohlin, 2014), such as rules concerning diagram (model) synthesis, rules in standard UML-driven software development processes discussed in textbooks, traceability rules, and rules for domain specific languages; these types of rules are not covered in this study. Another threat to construct validity is represented by the fact that UML can be seen as a mechanism with which to define the notation and semantics for several kinds of software development phases and the exact software development phase under which a particular UML diagram falls may, therefore, be a matter of discussion (Dathan and Ramnath, 2015). In order to reduce the impact of this threat, we chose

Table 6
Set of UML consistency rules.

1 - State Machine Diagram			
1	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. The initial states of the state machine diagrams U′ and U must be identical.	**I**	**Se**
2	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. Every transition of U′ which is already in U has at least the same source states and sink states as it has in U.	**I**	**Se**
3	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. For each transition t in U′ that is already present in U, the guard condition g′(t) in U′ must be at least as strong as the guard condition g(t) for t in U: g′(t) → g(t).	**I**	**Se**
4	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. A transition of U in U′ cannot therefore receive an additional source state or sink state that is already present in U.	**I**	**Se**
5	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. A transition added to U′ does not receive a source state or a sink state that was already present in U.	**I**	**Se**
6	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. The set of transitions of U′ is a superset of the set of transitions of U.	**I**	**Se**
7	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. A transition in U′ which is already present in U has in U′ at most the source states that the transition has in U.	**I**	**Se**
8	Consider two State Machines U′ and U of a class O′ and its superclass O, where U′ extends the state machine of U by adding states and transitions. For each transition t in U′ that is already present in U, U: g(t) → g′(t).	**I**	**Se**
9	Consider two State Machine U″ and U of a class O″ and its superclass O, where U″ refines the state diagram of U by means of a refinement function h that maps transitions onto transitions and states of U″ and that maps simple states of U″ onto simple states of U. Intuitively, the concept of the refinement function means that if a simple state s of U has been refined to a composite state in U″, then h maps the substates of s in U″ and transitions between these states to s, and h maps the transitions in U″ that are incident to the substates of s into transitions of U that are incident to s. s is a state in U, so s Є U (from the rule: "a simple state s of U","s of h(t′)") s′ is a refined state in U′, so s′ Є U′ (from the rule: "a state s′ in U′ ","sink state s′ of t′ in U′ ") Source + (t): t → set of states. Source + (t) = {source of t} U sub-states*{source of t} s′ Є Source + (t) For every transition t′ in S′: for every source state s of h(t′), there exists a state s′ in S′ such that h(s′) = s, and for every sink state s of h(t′) there exists a sink state s′ of t′ in S′ such that h(s′) = s. Informal rule definition: 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) Є Source + (t).	**E**	**Se**
10	Consider two State Machine U″ and U of a class O″ and its superclass O, where U″ refines the state diagram of U by means of refinement function h that maps transitions onto transitions and states of U″ and that maps simple states of U″ onto simple states of U. Intuitively, the concept of the refinement function means that if a simple state s of U has been refined to a composite state in U″, then h maps the substates of s in U″ and transitions between these states to s, and h maps the transitions in U″ that are incident to the substates of s into transitions of U that are incident to s. s is a state in U, so s Є U (from the rule: "a simple state s of U","s of h(t′)") s′ is a refined state in U′, so s′ Є U′ (from the rule: "a state s′ in U′ ","sink state s′ of t′ in U′ ") Source + (t): t → set of states. Source + (t) = {source of t} U sub-states*{source of t} s′ Є Source + (t) For every source state s′ of a transition t′ in S″, where s″ and t″ do not belong to the same refined state (i.e., h(s″) / = h(t″)), h(s″) is therefore a source state of h(t″), and for every sink state s″ of a transition t″ in S″, where s′ and t′ do not belong to the same refined state, h(s″) is therefore a sink state of h(t″). Informal rule definition: This rule is about transitions that go between states that are not substates of a complex state that is a refinement of a simple state. The rule says that such transitions have their sources and sinks in U′ as refinements of those ones in U.	**E**	**Se**
11	Using a signal/message on a transition in a state diagram that no object sends, creates structural and syntactic inconsistencies. The information about the sending object can be found in another state machine, in another partition of the same state machine, in a sequence diagram, or any other diagram where one can specify an object can send a signal/message.	**H**	**Sy**
12	A state machine should be deadlock-free.	**H**	**Se**
13	A state machine must be deterministic, that is, in every state, only one transition (accounting for the different levels of nested states) should fire on a reception of an event.	**H**	**Sy**
14	An abstract operation cannot be invoked in a state machine.	**H**	**Sy**
117	The joint effect of a sequence of refined operations should exactly adheres to the refined abstract operation. I.e. if u = operation if u1 + u2 + u3 refine u, - > the joint effect of u1 + u2 + u3 should exactly adheres to u.	**I**	**Se**
2 - Class Diagram, State Machine Diagram and Activity Diagram			
15	There is an inconsistency if a precondition on an operation is in contradiction with a state machine or an activity diagram including a call of that operation.	**H**	**Sy**
3 - Sequence Diagram and Use Case Diagram			
16	If a use case is further specified by one or more sequence diagram, then every scenario described in one of those sequence diagrams should match a sequence of steps in that use case's use case description.	**H**	**Se**
17	If a sequence diagram depicts all the behavior required for successful completion of a use case, it follows that each post-condition specified in the use case description must be achieved by a message (or a set of messages) in the sequence diagram (including referred diagrams) for that use case.	**H**	**Sy**
18	If a sequence diagram depicts all the behavior required for successful completion of a use case, it follows that results achieved by alternative message flows in the sequence diagram (including referred diagrams) must correspond to post-conditions specified in the use case description.	**H**	**Sy**
19	Each action specified or implied in a use case description should be detailed in a corresponding message or set of messages in the sequence diagram corresponding to that use case. Depending on the clarity and completeness of the use case description text, the author of the sequence diagram may need to infer some of the operations.	**H**	**Se**
20	A use case is complemented by a set of sequence diagrams, each sequence diagram representing an alternative scenario of the use case.	**H**	**Se**
4 - Activity Diagram (*no rule*)			
5 - Sequence Diagram			
21	Arguments to messages must represent information that is known to the sender. This includes attribute values of the sender, navigation expressions starting with the sender, constants; This should account for inheritance.	**H**	**Sy**
22	Variables used in the guard of a message must represent information that is known to the sender. This includes attribute values of the sender, navigation expressions starting with the sender, constraints; This should account for inheritance.	**H**	**Sy**

(*continued on next page*)

Table 6 (*continued*)

4 - Activity Diagram (*no rule*)			
5 - Sequence Diagram			
23	If SD2 is a sequence diagram referred to by an Interaction Use (a nested sequence diagram) embedded in sequence diagram SD1, then for every matched pair of message to and from SD2 in SD1, there is a corresponding matched pair of sourceless message and targetless message in SD2.	**H**	**Sy**
24	In a sequence diagram, if an attribute is assigned the return value of a message, then the types have to be compatible	**H**	**Sy**
25	Return messages from ExecutionSpecification instances should always be shown.	**H**	**Sy**
26	Arguments of messages should always be shown.	**H**	**Sy**
6 – Object Diagram and Collaboration Diagram (*no rule*)			
7 - Object Diagram and Class Diagram			
27	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.	**H**	**Sy**
8 - Activity Diagram and Class Diagram			
28	A class name that appears in an activity diagram also appears in the class diagram.	**H**	**Sy**
29	An action that appears in an activity diagram must also appear in the class diagram as an operation of a class.	**H**	**Sy**
30	When an activity diagram specifies an exchange of information, through control or data flow, between two different instances, the class diagram should specify a mechanism (e.g., direct association) so that those instances can indeed communicate.	**H**	**Sy**
31	Swimlanes (Activity pattern in UML 2.0) in an Activity diagram (represented as className in activity state) must be present as a unique class in a class diagram.	**H**	**Sy**
9 - State Machine Diagram and Activity Diagram (*no rule*)			
10 – Use Case Diagram and State Machine Diagram (*no rule*)			
11 - Class Diagram and State Machine Diagram			
32	When a state machine specifies the behavior of a class, the actions and activities in the state machine should be operations of the class (in the class diagram) which behavior the state machine specifies. An action or activity can be part of a navigation expression, in which case the navigation expression must legal according to the class diagram and the context (class) of the specified behavior in the state machine.	**H**	**Sy**
33	When a state machine specifies the behavior of a class, any property (i.e., attribute, navigation) in the state machine should be one of the class (in the class diagram) which behavior the state machine specifies. A property can be part of a navigation expression, in which case the navigation expression must legal according to the class diagram and the context (class) of the specified behavior in the state machine.	**H**	**Sy**
34	When the state machine diagram specifies the behavior of a class in the class diagram, the class is an active class with a classifierBehavior, then the triggers of the transitions in the state machine diagram are operations of the active class.	**H**	**Sy**
35	No operation can be called on a state machine or from a state machine if this breaks the visibility rules of the class diagram (public, protected, private).	**H**	**Sy**
36	When behavior is triggered from a state machine diagram (e.g., calling an operation, sending a signal) that describes the behavior of a class, and the triggered behavior belongs to another class, then the former must have a handle to the latter as specified in the class diagram. Another way of saying this is that the former must have visibility to the latter. A specific case of this situation is when the former class has an association (possibly inherited) to the latter class.	**H**	**Sy**
37	For a send action, there should be a reception within the classifier of the receiver instance that corresponds to the signal of the send action describing the expected behavior response to the signal.	**H**	**Sy**
38	For all call or send events specified in the class diagram for a classifier (context) which behavior is specified with a state machine, there should be transitions in that state machine describing the detailed behavior of the events.	**H**	**Sy**
12 – Protocol State Machine Diagram and Class Diagram (*no rule*)			
13 - Sequence Diagram and Activity Diagram			
39	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.	**H**	**Sy**
40	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.	**H**	**Sy**
41	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a sequence of messages in the sequence diagram, uninterrupted by control flow structures, must correspond to one activity node in the activity diagram which name is the series of message names of the sequence diagram.	**H**	**Sy**
42	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a synchronous message between objects running in different threads of control in the sequence diagram must match a join node for the receiving side (thread) in the corresponding activity diagram, and a fork node for the asynchronous reply.	**H**	**Sy**
43	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, an asynchronous creation of an active object in the sequence diagram must match a fork node in the corresponding activity diagram: the input action node is matching the calling thread message; two outgoing edges should flow out of the fork node, one for the node matching continuation of execution in the calling thread and one for the node matching the newly created active object.	**H**	**Sy**
44	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, an asynchronous message sent between two active objects in the sequence diagram should match, in the corresponding activity diagram, a join node on the receiver side and a fork node on the sender side.	**H**	**Sy**
45	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a InteractionUse in the sequence diagram must match an activity node in the activity diagram that refers to the activity diagram matching the sequence diagram referred to in the InteractionUse.	**H**	**Sy**
14 – Use Case Diagram and Object Diagram (*no rule*)			
15 - State Machine Diagram and Communication Diagram			
46	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.	**H**	**Sy**
16 - Communication Diagram and Class Diagram			

(*continued on next page*)

Table 6 (*continued*)

16 - Communication Diagram and Class Diagram			
47	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. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	**H**	**Sy**
17 - State Machine Diagram and Sequence Diagram			
48	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.	**H**	**Sy**
18 – Protocol State Machine Diagram (*no rule*)			
19 - Communication Diagram			
49	If communication diagram A is a specialization of communication diagram B, all the messages present in B have to be included in A.	**E**	**Sy**
20 - Use Case Diagram and Class Diagram			
50	The noun of the use case's name should equal the name of one class in the class diagram.	**H**	**Sy**
51	The verb of the use case's name should equal the name of an operation of a class in the class diagram.	**H**	**Sy**
52	Entity classes correspond to data manipulated by the system as described in a use case description.	**H**	**Sy**
21 – Object Diagram and Activity Diagram (*no rule*)			
22 - Communication Diagram and Activity Diagram			
53	If an activity diagram specifies a flow of interaction between objects, and those objects (or a subset of those objects) appears in a communication diagram, then the communication diagram should specify the same flow of interaction.	**H**	**Sy**
23 - Communication Diagram and Class Diagram			
54	Objects involved in a communication diagram should be instances of classes of the class diagram.	**H**	**Sy**
55	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. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	**H**	**Sy**
56	Each class in the class diagram appears with at least one instance in at least one communication diagram.	**H**	**Sy**
24 – Communication Diagram and Use Case Diagram (*no rule*)			
25 - Use Case Diagram and Activity Diagram			
57	If a use case U (the including use case) includes use case V (the included use case) in the use case diagram, and both use case U's flows and use case V's flows are specified as activity diagrams, then the activity diagram specifying use case U should contain an action node (likely a CallBehaviorAction node) that refers to the activity diagram specifying use case V.	**H**	**Sy**
58	Each use case is described by at least one activity diagram.	**H**	**Sy**
59	Each event (flow of steps) specified or implied in the use case description should be detailed in a corresponding event of the activity diagram. This rule is valid only if the activity diagram further specifies the use case.	**H**	**Sy**
60	An actor that is associated with a use case will be an activity partition in the activity diagram describing that use case.	**H**	**Sy**
26 - Use Case Diagram			
61	The use case description of a use case in the use case diagram should contain at least one event (flow of steps). Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**
62	An event (flow of steps) in a use case description of a use case of the use case diagram has to be of either a basic or an alternate type. Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**
63	Each use case in the use case diagram must have a use case description specifying event flows. Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**
64	The name of a use case must include a verb and a noun (for instance "Validate User").	**H**	**Sy**
28 - Class Diagram			
65	A class invariant must be satisfied by any non-trivial instantiation of the class diagram (i.e., an instantiation that is not reduced to having no instance of any class).	**H**	**Sy**
66	The type of a relationship between two classes at a (high) level of abstraction (e.g., plain association, aggregation, composition, generalization) must be the same as the type of a refinement of that relationship at a more concrete (low) level of abstraction. For instance, a plain association at a low level of abstraction being abstracted as an aggregation (a high level of abstraction) denotes an inconsistency.	**V**	**Se**
67	A class diagram is consistent if it can be instantiated without violating any of the constraints in the diagram (e.g., association end multiplicities).	**H**	**Sy**
68	Two classes are equivalent if they denote the same set of instances whenever the constraints imposed by the class diagram are satisfied. Determining equivalence of two classes allows their merging, thus reducing the complexity of the diagram.	**H**	**Sy**
69	A class relationship at a (low) level of abstraction must have an abstraction at a higher level of abstraction, either a class or a relation.	**V**	**Se**
70	If a class relationship A refines relation B, A must have the same destinations classes as the destination classes of the abstraction of B.	**V**	**Se**
71	If a class relationship A refines relation B, A must have the same type as B.	**V**	**Se**
72	The group of relationships between any two high level classes must be identical with the group of relationships between their corresponding low-level classes: ensures the same interactions among low level classes and high-level classes.	**V**	**Se**
73	If a navigation expression is used in an operation contract, then the expression must be a legal one (according to the syntax of the language and the class diagram).	**H**	**Sy**
74	This Liskov's substitution principle holds.	**O**	**Se**

(*continued on next page*)

Table 6 (*continued*)

28 - Class Diagram			
75	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).	**H**	**Sy**
76	An abstract operation can only belong to an abstract class.	**H**	**Sy**
77	If an operation appears in a pre or post condition then it must have the property isQuery equal to true.	**H**	**Sy**
78	No (public) method of a class violates, as indicated by its pre and post-conditions, the class invariant of that class.	**H**	**Sy**
79	In a class, the names of the association ends (on the opposite side of associations from this class) and the names of the attributes (of the class) are different.	**H**	**Sy**
80	No precondition should violate the class invariant.	**H**	**Sy**
81	No post-condition should violate the class invariant.	**H**	**Sy**
82	A class that contains an abstract operation must be abstract.	**H**	**Sy**
83	There must be no cycle in the directed path of aggregation associations: A class cannot be a part of an aggregation in which it is the whole; A class cannot be a part of an aggregation in which its superclass (or ancestor) is the whole.	**H**	**Sy**
84	A class cannot be a part of more than one composition - no composite part may be shared by two composite classes.	**H**	**Sy**
85	Each concrete class, i.e., it is not abstract, must implement all the abstract operations of its superclass(es).	**H**	**Sy**
86	If an attribute's type is a class, then that class has to be visible to the class containing the attribute. Example: same package or there exists a path in the class diagram that allows the class containing the attribute to have a hold on that type.	**H**	**Sy**
87	If the return type of an operation is a class, then that class has to be visible to the class containing the operation. Example: same package or there exists a path in the class diagram that allows the class containing the operation to have a hold on that type.	**H**	**Sy**
88	An operation may not be overridden by a descendant class only if its isLeaf attribute (from metaclass RedefinableElement) is defined accordingly.	**H**	**Sy**
89	A static operation cannot access an instance attribute (as indicated by its pre and post conditions, for instance).	**H**	**Sy**
90	A static operation cannot invoke an instance operation (as indicated by its pre and post conditions, for instance).	**H**	**Sy**
91	If an association end has a private visibility, then the class at this end can only be accessed via the association by the class at the other association end (e.g., as indicated by pre and post conditions of operations of the class at this other end of the association).	**H**	**Sy**
92	If an association end has a protected visibility, then the class at this end can only be accessed via the association, by the class at the other association end and its descendants (e.g., as indicated by pre and post conditions of operations of the class at this other end of the association).	**H**	**Sy**
93	The multiplicity range for an attribute must be adhered to by all elements (operation contracts, guard conditions) that access it.	**H**	**Se**
94	For class A's operations to use another class B, as indicated by contracts in A, there must be a means (e.g., in the form of a path involving associations, generalization and/or dependencies) in the class diagram for A to get a hold on B.	**H**	**Sy**
95	A class at a low-level of abstraction refines at most one class from a higher-level of abstraction.	**V**	**Sy**
96	A class at a high-level of abstraction is refined by at least one class from a lower-level of abstraction.	**V**	**Sy**
97	There should not be semantically redundant paths between any two classes in the class diagram graph, unless precisely specified by a constraint (e.g., specified in OCL). For instance, from class A, it may be possible to navigate as self.theA.theB, along with self.theC.theB. The question is then whether the two collections self.theA.theB and self.theC.theB are identical.	**H**	**Sy**
118	Parent Class should not have a Method with a Parameter referring to a Child Class	**H**	**Sy**
119	A Class should not have an Attribute which as a Class typed as Child Class.	**H**	**Sy**
29 – Interaction Diagram and Class Diagram (*no rule*)			
30 - Use Case Diagram and Interaction Diagram			
98	Each interaction diagram corresponds to a use case in the use case diagram, and each use case in the use case diagram is specified by an interaction diagram.	**H**	**Sy**
99	If a use case is further specified by one or more interaction diagrams, then every scenario described in one of those interaction diagrams should match a sequence of steps in that use case's use case description.	**H**	**Sy**
31 – Protocol State Machine Diagram and State Machine Diagram (*no rule*)			
32 - Composite Structure Diagram			
100	A delegation connector connects a port on the boundary of a classifier to a port on the internal structure (or parts) of the classifier. It basically delegates signals or operation calls arriving on the boundary port to the internal port, or those coming from the internal port to the boundary port. Therefore, these ports at the end of the delegation connector must have the same (or compatible) interfaces. If both ports require the interface, then the direction of delegation is from the boundary port to the internal port. If both ports provide the interface, then the direction of delegation is from the internal port to the boundary port.	**H**	**Sy**
101	If an assembly connector exists between two ports, one of the ports (the source) must be a required port and the other (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 (i.e., ones that have the same operations or signal receptions that are annotated as provided on one interface and required on the other).	**H**	**Sy**
102	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.	**H**	**Sy**
103	If a link outgoing from a port is statically typed with an association, then the association must be navigable to an interface which is part of the set of interfaces and the type indicated by the association must belong to the set of transported interfaces for that link.	**H**	**Sy**
104	If the link originates from a component, then the link must be statically typed with an association, and the type of the entity at the other end of the link must be compatible with (i.e. be equal or a subtype of) the type at the corresponding end of the association.	**H**	**Sy**
105	The set of transported interfaces by a link should not be empty.	**H**	**Sy**
106	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.	**H**	**Sy**
107	The union of the sets of interfaces transported by each of the connectors originating from a port P must be equal to the set of interfaces provided/required by P.	**H**	**Sy**
33 - Sequence Diagram and Class Diagram			
108	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.	**H**	**Sy**
109	In case a message in a sequence diagram is referring to an operation, that operation must not be abstract.	**H**	**Sy**
110	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.	**H**	**Sy**
111	Interactions between objects in a sequence diagram, specifically the numbers of types of interacting objects, must comply with the multiplicity restrictions specified by the class diagram (e.g., association end multiplicities).	**H**	**Sy**

(*continued on next page*)

Table 6 (*continued*)

33 - Sequence Diagram and Class Diagram			
112	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. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	**H**	**Sy**
113	The behavioral semantics of a composition or aggregation association in the class diagram must be inferred in sequence diagrams. For instance, in a whole-part (composition) relation, the part should not outlive the whole.	**H**	**Sy**
114	Each public method in a class diagram triggers a message in at least one sequence diagram.	**H**	**Sy**
115	Each class in the class diagram must be instantiated in a sequence diagram.	**H**	**Sy**
116	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).	**H**	**Sy**
34 - Sequence Diagram and Communication Diagram (*no rule*)			
35 - Class Diagram, State Machine Diagram and Communication Diagram (*no rule*)			
36 - Class Diagram, Sequence Diagram and Use Case Diagram (*no rule*)			

to answer RQ6 (What software development phases are involved with UML consistency rules?) according to three different software development methodologies SDM#1, SDM#2, and SDM#3 (see Section 5.7 for our motivations). Finally, by following a systematic research method, we developed a set of terms with the intention of providing a useful and accessible set of definitions for the terms that are used within the UML consistency domain.

6.2. Internal validity

Threats to internal validity are influences that can affect the independent variable with respect to causality, without the researcher's knowledge (Wohlin et al., 2012). The main idea behind conducting this study was to collect UML consistency rules in literature without, as far as possible, introducing any researcher bias and internal validity would, therefore, appear to be a major challenge for the study. Since in this study we relied solely on rules that have been published in literature (Torre et al., 2014), the fact that we did not find rules involving certain diagram(s) (described in Section 5.3) does not mean that none have been created. Moreover, we know that there is a lot of tool support for UML, and since these tools support some kinds of model-driven development activities, they must rely on, and likely enforce, UML diagram consistency. However, the rules they rely on are not necessarily in the public domain, or at least are not discussed in published literature. In addition, some rules might also be for specific domains (industry, organization, project, team specific, etc.)

6.3. External validity

External validity refers to generalization from this study. In general, the external validity of a systematic study is strong, as the key idea is to aggregate as much of the available literature as possible (Borg et al., 2014). Moreover, our specific research goal focused on UML consistency rules (see the inclusion/exclusion criteria in Section 3.2), and the fact that we do not claim that our map covers other types of UML rules such as traceability rules, synthesis rules, rules presented in books and domain specific rules, helps to reduce the threats to external validity. We used seven search engines to search journals, conference and workshop proceedings that are relevant to UML consistency rules (Torre et al., 2014). We did not consider grey literature (e.g., PhD theses, books) because it might have affected the validity of our results. We then organized the selection of rules by following a rigorous multi-phase process, during which our filtering criteria were applied, to attain a consolidated set of UML consistency rules. The limited number of rules in this research could be an inherent external threat to validity, despite the fact that we attempted to mitigate this by following rigorous guidelines that are well-known in the empirical software engineering community (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015).

6.4. Reliability

This aspect is concerned with the extent to which the data and the analysis are dependent on the specific researchers (Wohlin et al., 2012). In order to achieve reliability, the research steps must be repeatable, i.e., other researchers have to be able to replicate our results (Badampudi et al., 2016). While conducting this type of study, there is a possibility of missing relevant papers and consequently UML consistency rules. It is impossible to thoroughly search for every rule published on UML consistency, and we acknowledge that some rules might not have been included. The main issues that may constitute a threat to the reliability of our research are related to publication bias, selection bias, inaccuracy in data extraction, and misclassification (Sjoberg et al., 2005). Selection bias refers to the distortion of a statistical analysis owing to the criteria used to select the rules. We attempted to mitigate this risk by carefully defining (1) our inclusion and exclusion criteria in order to select primary studies (Torre et al., 2014), and (2) our exclusion criteria in order to select rules in this work, based on our predefined research questions. In order to increase reliability, two researchers (the first and second authors) were responsible for applying these criteria, with the help of the fourth author who was contacted in the case of a disagreement regarding the inclusion or exclusion of a rule. The entire systematic procedure was performed by the first author beforehand and supervised and evaluated by the third and second authors.

7. Conclusions and future work

In recent years, a great number of UML consistency rules have been proposed by researchers in order to detect inconsistencies between the diagrams of a UML model. However, no previous study has, to the best of our knowledge, summarized and analyzed these UML consistency rules by following a systematic process. This work presents the results obtained after following a systematic protocol, whose aim was to identify, present and analyze a consolidated set of UML consistency rules obtained from literature. No such mapping study or review existed prior to our work.

We obtained the set of UML consistency rules by following a systematic procedure inspired by well-known guidelines for performing systematic literature reviews in software engineering (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015). A significant amount of space in this paper has, therefore, been used to discuss this systematic process, since we felt that it is of the upmost importance to allow readers to understand precisely how we attained

Table 7
Research questions.

Research questions	Main motivation
RQ1: What are the UML versions used by researchers in the approaches found?	To discover what UML versions are used in the approaches that handle UML consistency.
RQ2: Which types of UML diagrams have been tackled in each approach found?	To discover the UML diagrams that research has focused upon, to reveal which UML diagrams are considered more important than others, in addition to identifying opportunities for further research.
RQ3: What are the UML consistency rules to be checked?	To find the UML consistency rules to be checked and to assess the state of the field.
RQ4: Which types of consistency problems have been tackled in the rules found?	To find the types of consistency problems tackled in the rules. The data found are categorized into three consistency dimensions split into three sub-dimensions: 1) horizontal, vertical and evolution consistency; 2) syntactic and semantic consistency; 3) observation and invocation consistency.
RQ5: Which research type facets are used in research on UML model consistency?	To determine whether the field is generally more applied or more basic research, along with identifying opportunities for future research. The papers found were categorized into six types: evaluation research, validation research proposal of solution, philosophical papers, opinion papers and personal experience papers.
RQ6: Is the approach presented automatic, manual or semi-automatic?	To discover how the approaches used to check UML consistency are implemented, in other words, whether their check system is presented in an automatic, manual or semi-automatic manner.
RQ7: How are the UML consistency rules specified? How are the UML consistency rules checked?	To discover how the consistency rules used to check the consistency of the UML diagrams are specified (e.g., Plain English, OCL, Promela) and to discover with which tools those consistency rules are checked (e.g., SPIN, OCL-Checker)

the results and reached our conclusions, and to allow other researchers to replicate the study or compare their results with ours.

From an initial set of 687 UML consistency rules, published in literature and extracted from seven scientific research databases, a total of 119 rules were eventually selected and analyzed in depth, by following a precise selection protocol driven by six research questions. It is possible to make the following observations, in no particular order of importance.

The software development phases in which UML consistency rules were most involved are: analysis (88.7%) and design (83.26%) for SDM#1, interaction model (48.74%), dynamic model (35.56%), and logical model (61.51%) for SDM#2, and object model (88.7%) and dynamic model (83.26%) for SDM#3. The phases that had the highest frequencies of rules are those that involve Class, Interaction, and State Machine diagrams. In fact, the UML diagrams that are most frequently involved in the final set of 119 rules are the Class Diagram (64 rules, 53.78%), followed by the Interaction Diagram (37 rules 31.09%), and the State Machine Diagram (25 rules, 21.01%). This is not entirely surprising, since these are likely the most frequently used UML diagrams (Dobing and Parsons, 2006), despite the fact that other research (Reggio et al., 2013) suggests that the Activity Diagram is the second most frequently used UML diagram after the Class Diagram. Considering the small number of rules, we found for the Activity Diagram (17 of 119 rules, 14.28%), we believe that future research should focus more on this diagram. However, if we compare the last UML 2.5 specification (released in 2015) (OMG, 2015) with the most frequently used UML 2.0 specification (released in 2005) (OMG, 2005), we can see that the UML has put more effort into the definition of the Activity Diagram, which has consequently led to (explicitly or otherwise) consistency rules concerning that diagram. Users of the UML may, therefore, not need to specify (and publish) a significant number of new rules involving the Activity Diagram. Another diagram to which research on UML consistency rules has paid much less attention is the Use Case Diagram (18 rules, 15.13%). We believe that one of the reasons for these results is that this diagram has a simple notation and semantics, even though the semantics for this diagram is not optimally clear to everyone: Cockburn argues (Cockburn, 2000), for instance, that generalization between use cases is not clear and necessary, while other authors (Chanda et al., 2009; Fryz and Kotulski, 2007; Ibrahim et al., 2011a) argue that there are consistency rules between use cases, and more so use case descriptions and other diagrams such as class and sequence diagrams. In addition, the mapping between semantics in the use case diagram and other diagrams is not clear either: for instance, if we have a generalization between use cases and use cases that maps to sequence diagrams, how does the generalization (between use cases) translate into relations between the sequence diagrams?

Besides the Class, Interaction, and State Machine diagrams, there is a need for much additional research on consistency rules involving other UML diagrams. For example, we found no rule for the Package, Component, Timing, Profile, Interaction Overview and Deployment Diagrams. One consideration that could partially explain the lack of rules for some of these diagrams is the fact that they overlap with the Class diagram syntax/semantics. Another hypothesis regarding these results is that some of these diagrams are not used in SysML(OMG, 2017) (e.g., Component, Timing, Interaction Overview and Deployment), and since SysML is usually more frequently used than UML in rigorous applications, then not much attention has been paid to them.

Additionally, 40.75% (280 of 687) of the collected rules that have been proposed by researchers over the years are redundant (similar or even identical) rules. This highlights the need for a central repository for such rules, such as this manuscript. Another appropriate place in which to record these rules would be the UML specification itself. The consolidated set of 119 UML consistency rules we describe could, therefore, be a good input for the UML revision task force for inclusion in a forthcoming revision of the standard.

We found that 39.9% (79 of 198) of the non-redundant accurate UML consistency rules presented by different authors had already been presented in one of the previous UML standards.

We observed that the main software development activity that justified the description of UML diagram consistency rules is UML consistency verification, 52.93%, followed by consistency management, 15.06%, impact analysis, 11.92%, and model refinement and transformation, 11.3%. We conjecture that this suggests that the same set of UML consistency rules is needed for most of, if not all, the UML-based software engineering activities.

We obtained that 44.14% of the rules were published between the years 2002 and 2005, and this period represents the most prolific period in literature in terms of publishing UML/model consistency rules.

The results show that the vast majority of the rules are horizontal and syntactic rules, and that very few rules address vertical, semantic, invocation, observation and evolution consistency. One reason for this could be the fact that the UML syntax is more formally described (by the UML metamodel) in the specification than semantics and creating syntactic rules may, therefore, prove more feasible. Another reason could be the fact that users mostly employ a few types of diagrams that they understand or find suitable. Despite the fact that the topic of UML consistency is mature, it still needs to evolve to include more definitions of UML consistency rules in all dimensions.

By demonstrating that there are various areas concerning UML consistency rules that require improvement, our research calls for future work in these areas. Firstly, we pose the following question: should some of the rules presented in this work, such as the most frequently referenced rules numbered 112, 110, 48 and 115 (see Appendix A), be added to the UML standard? Should all 119 rules be enforced in a UML model, and if not, which subset of those 119 rules should?

Finally, we believe that the consolidated set of UML consistency rules that eventually resulted from this research could be used as a reference in the future by researchers in the field of UML model consistency.

Different avenues for future work can be considered. During our research on UML consistency rules, we identified that other rules could be collected from other sources, rather than only academic peer-reviewed papers on UML consistency rules, such as: 1) textbooks on UML-based object-oriented software development (Torre et al., 2015b) that implicitly or explicitly suggest consistency rules; 2) techniques that synthesize UML diagrams from other diagrams (Torre et al., 2018), in other words, the process of enforcing some consistency rules between diagrams, and 3) consistency rules presented in the context of Domain Specific Languages (Hoisl and Sobernig, 2015). These represent a limitation of this manuscript and a set of potential future works in the area of UML/model consistency rules.

In our previous publication (Torre et al., 2014), we found that very few evaluation works on consistency rules have, as yet, been reported in literature. We, therefore, initiated the process of validating the rules collected by organizing the 1st International Workshop on UML Consistency Rules (WUCOR) (Torre et al., 2016) during the Models 2015 conference.

As part of our future work, we will develop an online survey that will be used to obtain the opinions of experts from academia and industry regarding UML/model consistency issues and the set of 119 UML consistency rules. In addition, we are planning to evaluate the set of 119 UML consistency rules directly on UML models (Torre, 2016). In order to do this, the following steps will be executed: 1) specifying the UML consistency rules (Torre et al., 2015a), currently specified in plain language, in OCL (OMG, 2016); 2) collecting UML models from different repositories (such as the UML Repository,[2] The Lindholmen Dataset,[3] and ReMoDD[4]); 3) importing them to the UML tool Eclipse Papyrus[5]; 4) and then running the OCL constraints on them. This activity will ultimately contribute to validating the relevance of the UML consistency rules (Torre et al., 2015a) and would complement this work (Torre, 2014).

Acknowledgments

This work has been developed within the SEQUOIA Project (TIN2015-63502-C3-1-R) (MINECO/FEDER) funded by "Fondo Europeo de Desarrollo Regional and Ministerio de Economía y Competitividad", the GEMA project funded by "Consejería de Educación, Cultura y Deportes de la Dirección General de Universidades, Investigación e Innovación de la JCCM" and a NSERC discovery grant.

Appendix A

Table 6 lists the 119 UML consistency rules. The top row, in grey, presents a reference number for the UML diagram combination referred to in the rules (reference numbers from Table 4) and the name of the UML diagram/s involved. The first column is the univocal number of each UML consistency rule. The second column describes the rules. The third column indicates whether the rule covers the Horizontal (H) Consistency, Vertical (V) Consistency, Evolution (E) Consistency, Invocation (I) Consistency, or Observation (O) Consistency dimension. The fourth column indicates whether this is a Syntactic (Sy) or Semantic (Se) rule.

Appendix B

In this appendix, we present the main components of the protocol we followed during our SMS (Torre et al., 2014), along with its update up to March 2017. We carried out the SMS following a well-known procedure (Kitchenham et al., 2015; Kitchenham and Charters, 2007; Petersen et al., 2015), since this was the starting point employed to build the consolidated set of 119 UML consistency rules.

B.1. Research questions

The underlying motivation for the research questions was to determine the current state of the art as regards UML consistency. We considered seven research questions (RQs): Table 7.

B.2. Search strategy

Conducting a search for primary studies requires the identification of search strings (SS), and the specification of the parts of primary studies (papers) in which the search strings are sought (the search fields). We identified our search strings by following the procedure of Brereton et al. (2007): (1) Define the major terms; (2) Identify alternative spellings, synonyms or related terms for major terms; (3) Check the keywords in any relevant papers that are already available; (4) Use the Boolean OR to incorporate alternative spellings, synonyms or related terms, and (5) Use the Boolean AND to link the major terms.

The major search terms were “UML” and “Consistency” and the alternative spellings, synonyms or terms related to the major terms were (uml OR unified modeling language OR unified modelling language) for “UML” and (consistency OR inconsistency) for “Consistency”.

We considered various alternatives during the selection of the SS. For example, the SS used in the SLR on consistency management (Lucas et al., 2009) was discarded owing to the fact that it might not strictly focus on UML consistency rules: we are much more interested in collecting rules than in identifying consistency management issues and solutions. (This is the main reason why we obtained a different set of primary studies.) Other SSs were experimented with, but it is not possible to discuss all the alternative search strings here owing to space limitations. We selected the following SS from the set of alternatives, as it allowed us to retrieve the largest number of useful papers, i.e., the largest number of papers focusing on UML consistency:

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

((uml OR unified modeling language OR unified modelling language) AND (consistency OR inconsistency))

We did not establish any restrictions on publication years up to March 30, 2017. We used the aforementioned SS with the following seven search engines: IEEE Digital Library, Science Direct, ACM Digital Library, Scopus, Springer Link, Google Scholar, and WILEY. The searches were limited to the following search fields: title, keywords and abstract.

B.3. Selection procedure and inclusion and exclusion criteria

In this section, we discuss the inclusion and exclusion criteria used. We then discuss the process followed to include a primary study in this SMS. The inclusion criteria were:

- Electronic Papers focusing on UML diagram consistency which contained at least one UML consistency rule;
- Electronic Papers written in English;
- Electronic Papers published in peer-reviewed journals, international conferences and workshops;
- Electronic Papers published up to March 31, 2017.
- Electronic Papers which proposed UML consistency rules with a restriction (or extension) of the UML models that do not strictly follow the OMG standard (OMG, 2015).

The exclusion criteria were:

- Electronic Papers not focusing on UML diagram consistency;
- Electronic Papers which did not present a full-text paper (title, abstract, complete body of the article and references) but were reduced to, for example, an abstract;
- Electronic Papers focusing on UML diagram consistency which did not contain at least one UML consistency rule;
- Duplicated Electronic Papers (e.g., returned by different search engines);
- Electronic Papers which discussed consistency rules between UML diagrams and other non-UML sources of data, such as requirements or source code.

B.4. Data extraction strategy

We extracted the data from the primary studies according to a number of criteria, which were directly derived from the research questions detailed in Table 7. We read the full text of each of the 105 primary studies, using each criterion to extract the data required. Once recorded, we collected the data on an Excel spreadsheet that we employed as our data form. The following information from each primary study was extracted and collected on the Excel data form:

- Search engines: where the paper was found (see Section 2 of Appendix B);
- Inclusion and Exclusion Criteria being used;
- Data related to Research Questions:

○ What UML version was used (RQ1);
○ Which UML consistency rules are discussed (see Appendix A) (RQ3);
○ What diagrams are involved in the consistency rules (see Section 3.3 of this paper) (RQ2);
○ UML consistency dimensions: UML diagram consistency is discussed according to several dimensions (Mens et al., 2005) found in literature (Torre et al., 2014) (RQ4):
§ Horizontal, Vertical and Evolution Consistency: Horizontal consistency, also called intra-model consistency, refers to consistency between different diagrams at the same level of abstraction (e.g., class and sequence diagrams during analysis) in a given version of a model (Huzar et al., 2005). Vertical Inconsistency, also called inter-model consistency, refers to consistency between diagrams at different levels of abstraction (e.g., analysis vs. design) in a given version of a model (Engels et al., 2002). Evolution consistency refers to consistency between diagrams of different versions of a model in the process of evolution (Huzar et al., 2005).
§ Syntactic versus Semantic consistency: Syntactic consistency ensures that a specification conforms to the abstract syntax specified by the metamodel, and requires that the overall model be well formed (Engels et al., 2002). Semantic consistency requires diagrams to be semantically compatible (Engels et al., 2002). This does not mean that semantic consistency is necessarily restricted to behavioral diagrams. For instance, operation contracts (e.g., pre and post conditions) provided in the class diagram specify behavior. Semantic consistency applies at one level of abstraction (with horizontal consistency), at different levels of abstraction (vertical consistency), and during model evolution (evolution consistency) (Ahmad and Nadeem, 2010).
§ Observation versus Invocation consistency: Observation consistency requires an instance of a subclass to behave like an instance of its superclass, when viewed according to the superclass description (Engels et al., 2001a) . In terms of UML state machine diagrams (corresponding to protocol state machines), this can be rephrased as “after hiding all new events, each sequence of the subclass state machine diagram should be contained in the set of sequences of the superclass state machine diagram.” Invocation consistency requires an instance of a subclass of a parent class to be used wherever an instance of the parent is required (Engels et al., 2001b). In terms of UML state machine diagrams (corresponding to protocol state machines), each sequence of transitions of the superclass state machine diagram should be contained in the set of sequences of transitions of the state machine diagram for the subclass.
○ Tool support (Automatic, Semi-Automatic, Manual):
§ Automatic means that a tool automatically checks the UML consistency rules with no human intervention;
§ Semi-automatic means that the checking of the UML consistency rules is partially automated (for instance, when the checking of a UML model must be supported by a user for the process to be completed);
§ Manual means that checking the UML consistency rules is not supported by any implemented and automatic tool.
○ What mechanisms were used to specify the rules: e.g., plain language, Promela (RQ7);

Table 8
Summary of primary studies selection.

Sub phase	IEEE	Scopus	Springer Link	Google Scholar	WILEY	ACM	Science Direct	Total
(a) Results of the UML consistency rules: a systematic mapping study (Torre et al., 2014)								
SP1: Raw results	363	601	163	341	9	87	39	1603
SP2: No duplicates	279	325	159	247	9	80	36	1135
SP3: First selection	62	64	62	28	4	33	14	267
SP4: Primary studies	16	21	21	12	1	16	8	**95**
(b) Results of the UML consistency rules: a systematic mapping study (Torre et al., 2014) update								
SP1: Raw results	197	342	69	192	6	41	18	865
SP2: No duplicates	127	189	50	160	6	30	16	578
SP3: First selection	27	31	19	10	0	8	4	99
SP4: Primary studies	5	2	3	0	0	0	0	**10**

○ How are the UML consistency rules checked: e.g., SPIN, OCL-Checker (RQ6);
○ Research type facet followed in the paper, for which we used the following classification (Wieringa et al., 2006) (RQ5):
§ Evaluation research: this is a paper that investigates techniques that are implemented in practice and an evaluation of the technique is conducted. This means that the paper shows how the technique is implemented in practice (solution implementation) and what the consequences of the implementation are in terms of benefits and drawbacks (implementation evaluation).
§ Proposal of solution: this is a paper that proposes a solution to a problem and argues for its relevance, without a full-blown validation.
§ Validation Research: this is a paper that investigates the properties of a solution that has not yet been implemented in practice.
§ Philosophical papers: this is a paper that provides a new way of looking at things, a new conceptual framework, etc.
§ Opinion paper: this is a paper that contains the author's opinion about what is wrong or good about something, how something should be done, etc.
§ Personal experience paper: this is a paper that places more emphasis on what and not on why.

B.5. Execution

In this section, we present the execution of the search with the SS in the seven search engines and the selection of primary studies according to the inclusion/exclusion criteria described previously. In order to document the review process with sufficient details (Kitchenham and Charters, 2007), Table 8 provides a description of the multi-phase process of the four sub-phases we followed for both the first SMS (Torre et al., 2014) (section (a) of Table 8) and its update (section (b) of Table 8).

- First sub-phase (SP1): the search string was used to search the seven search engines, as mentioned earlier.
- Second sub-phase (SP2): we deleted duplicates automatically, by using the RefWorks tool[6]; we also removed some duplicates manually.
- Third sub-phase (SP3): we obtained an initial set of studies by reading the title, abstract and keywords of all the papers obtained after SP2 while enforcing the inclusion and exclusion criteria. When reading only the title, abstract and keywords of a paper was not sufficient to decide whether to include or exclude it, we checked the full-text.
- Fourth sub-phase (SP4): all the papers identified in SP3 were read in their entirety and the exclusion criteria were applied again. This resulted in the final set of primary studies.

Table 5 breaks down the number of papers we have found by sub-phases. Row SP1 shows the first results, which were obtained by running the SS in the seven search engines selected. The next two rows show the results obtained after applying SP2 and SP3 of the study selection process. We eventually collected 105 (95 papers from (Torre et al., 2014) plus the 10 new papers from the (Torre et al., 2014) update) primary studies for further analysis. The complete list of primary studies can be found elsewhere (Appendix B of (Torre et al., 2015a)).

References

Ahmad, M.A., Nadeem, A., 2010. Consistency checking of UML models using Description Logics: A critical review. In: 6th International Conference on Emerging Technologies. Islamabad, Pakistan. IEEE Computer Society, pp. 310–315. https://doi.org/10.1109/ICET.2010.5638468.

Arlow, J., Neustadt, I., 2005. UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd ed. Addison-Wesley Professional.

Badampudi, D., Wohlin, C., Petersen, K., 2016. Software component decision-making: In-house, OSS, COTS or outsourcing - A systematic literature review. J. Syst. Softw. 121, 105–124. https://doi.org/10.1016/j.jss.2016.07.027.

Borg, M., Runeson, P., Ardö, A., 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Softw. Eng. 19, 1565–1616. https://doi.org/10.1007/s10664-013-9255-y.

Brambilla, M., Cabot, J., Wimmer, M., 2016. Model-Driven Software Engineering in Practice, 2nd ed. Morgan & Claypool Publishers. https://doi.org/10.2200/S00751ED2V01Y201701SWE004.

Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80, 571–583. https://doi.org/10.1016/j.jss.2006.07.009.

Briand, L.C., Labiche, Y., O'Sullivan, L., 2003. Impact analysis and change management of UML models. In: International Conference on Software Maintenance. Amsterdam, The Netherlands. IEEE Computer Society, pp. 256–265. https://doi.org/10.1109/ICSM.2003.1235428.

Bruegge, B., Dutoit, A., 2009. Object-Oriented Software Engineering Using Uml, Patterns, and Java. In: Press, P.H. (Ed.), 3rd ed. Upper Saddle River, NJ, USA.

Chanda, J., Kanjilal, A., Sengupta, S., Bhattacharya, S., 2009. Traceability of requirements and consistency verification of UML use case, activity and class diagram: A formal approach. In: International Conference on Methods and Models in Computer Science. New Delhi, India. IEEE Computer Society, pp. 1–4. https://doi.org/10.1109/ICM2CS.2009.5397941.

Cockburn, A., 2000. Writing Effective Use Cases, 1st ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dathan, B., Ramnath, S., 2015. Object-Oriented Analysis, Design and Implementation: An Integrated Approach, 2nd ed. Springer Publishing Company, Incorporated. https://doi.org/10.1007/978-3-319-24280-4.

Dobing, B., Parsons, J., 2006. How UML is used. ACM 49, 109–113. https://doi.org/10.1145/1125944.1125949.

Engels, G., Hausmann, J.H., Heckel, R., 2002. Testing the Consistency of Dynamic UML diagrams, Integrated Design and Process Technology. Pasadena, California.

Engels, G., Heckel, R., Küster, J.M., 2001a. Rule-based specification of behavioral

[6] http://www.refworks.com/

consistency based on the UML meta-model. In: Gogolla, M., Kobryn, C. (Eds.), 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer-Verlag, Toronto, Ontario, Canada, pp. 272–286. https://doi.org/10.1007/3-540-45441-1_21.

Engels, G., Küster, J.M., Heckel, R., Groenewegen, L., 2001b. A methodology for specifying and analyzing consistency of object-oriented behavioral models. Sigsoft Softw. Eng. Notes 26, 186–195. http://dx.doi.org/10.1145/503209.503235.

Fernández-Sáez, A.M., Genero, M., Caivano, D., Chaudron, M.R.V., 2016. Does the level of detail of UML diagrams affect the maintainability of source code?: A family of experiments. Empiri. Softw. Eng. 21, 212–259. https://doi.org/10.1007/s10664-014-9354-4.

Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V., Caivano, D., Ramos, I., 2015. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments. IST 57, 644–663. http://dx.doi.org/10.1145/2460999.2461008.

Fryz, L., Kotulski, L., 2007. Assurance of system consistency during independent creation of UML diagrams. In: 2nd International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX '07). IEEE Computer Society, pp. 51–58. https://doi.org/10.1109/DEPCOS-RELCOMEX.2007.11.

Genero, M., Fernández-Saez, A.M., Nelson, H.J., Poels, G., Piattini, M., 2011. A systematic literature review on the quality of uml models. J. Database Manage. 22, 46–70. https://doi.org/10.4018/jdm.2011070103.

Hoisl, B., Sobernig, S., 2015. Consistency rules for UML-based domain-specific language models: A literature review. In: Torre, D., Labiche, Y., Genero, M., Elaasar, M. (Eds.), 1st International Workshop on UML Consistency Rules (WUCOR 2015) Co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015). Ottawa, Canada. CEUR.

Hunt, J., 2003. Guide to the Unified Process Featuring UML, Java and Design Patterns, 2nd ed. Springer-Verlag, London.

Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.L., 2005. Consistency problems in UML-based software development. In: Nunes, N.J., Selic, B., da Silva, A.R., Alvarez, A.T. (Eds.), International Conference on UML Modeling Languages and Applications. Springer-Verlag, Lisbon, Portugal, pp. 1–12. https://doi.org/10.1007/978-3-540-31797-5_1.

Ibrahim, N., Ibrahim, R., Saringat, M.Z., 2011a. Definition of consistency rules between UML use case and activity diagram. In: Kim, T.-h., Adeli, H., Robles, R.J., Balitanas, M. (Eds.), 2nd International Conference. Daejeon, South Korea. Springer-Verlag, pp. 498–508. https://doi.org/10.1007/978-3-642-20998-7_58.

Ibrahim, N., Ibrahim, R., Saringat, M.Z., Mansor, D., Herawan, T., 2011b. Consistency rules between UML use case and activity diagrams using logical approach. Int. J. Softw. Eng. Appl. 5, 119–134.

IEEE, 1992. IEEE standard for software verification and validation plans (IEEE Std 1012-1986). https://standards.ieee.org/findstds/standard/1012-1986.html (last accessed on May 2018).

Kalibatiene, D., Vasilecas, O., Dubauskaite, R., 2013. Rule based approach for ensuring consistency in different UML models. In: 6th SIGSAND/PLAIS EuroSymposium 2013. Gdańsk, Poland. Springer-Verlag, pp. 1–16. https://doi.org/10.1007/978-3-642-40855-7_1.

Kitchenham, B., Budgen, D., Brereton, P., 2015. Evidence-Based Software Engineering and Systematic Reviews. Chapman & Hall/CRC.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews In Software Engineering, Ebse-2007-01. Keele University.

Labiche, Y., 2008. The UML is more than boxes and lines. In: Chaudron, M.R.V. (Ed.), Workshops and Symposia at MODELS 2008. Springer-Verlag, Toulouse, France, pp. 375–386. https://doi.org/10.1007/978-3-642-01648-6_39.

Lamancha, B.P., Polo, M., Caivano, D., Piattini, M., Visaggio, G., 2013. Automated generation of test oracles using a model-driven approach. Inf. Softw. Technol. 55 (2), 301–319. http://dx.doi.org/10.1016/j.infsof.2012.08.009.

Lucas, F.J., Molina, F., Toval, A., 2009. A systematic review of UML model consistency management. Inf. Softw. Technol. 51, 1631–1645. https://doi.org/10.1016/j.infsof.2009.04.009.

Mens, T., Van der Straeten, R., Simmonds, J., 2005. A Framework For Managing Consistency of Evolving UML models, Software Evolution With UML and XML. IGI Publishing, pp. 1–30.

Mukerji, J., Miller, J., 2003. Overview and guide to OMG's architecture, MDA Guide V1.0.1. Object Management Group.

OMG, 2005. OMG Unified Modeling LanguageTM - Superstructure Version 2.0. https://www.omg.org/spec/UML/2.0/About-UML/ (last accessed on May 2018).

OMG, 2011. OMG Unified Modeling LanguageTM - Superstructure Version 2.4.1. https://www.omg.org/spec/UML/2.4.1/About-UML/ (last accessed on May 2018).

OMG, 2015. OMG Unified Modeling LanguageTM - Superstructure Version 2.5. https://www.omg.org/spec/UML/2.5/About-UML/ (last accessed on May 2018).

OMG, 2016. Object Management Group - Object Constraint Language (OCL). http://www.omg.org/spec/OCL/ (last accessed on May 2018).

OMG, 2017. OMG System Modeling Language Specification Version 1.5. https://www.omg.org/spec/SysML/About-SysML/ (last accessed on May 2018).

Paige, R.F., Kolovos, D.S., Polack, F.A.C., 2005. Refinement via consistency checking in MDA. Electronic Notes Theor. Comput. Sci. 137, 151–161. http://dx.doi.org/10.1016/j.entcs.2005.04.029.

Pap, Z., Majzik, I., Pataricza, A., Szegi, A., 2005. Methods of checking general safety criteria in UML statechart specifications. Reliab. Eng. Syst. Safety 87, 89–107. http://dx.doi.org/10.1016/j.ress.2004.04.011.

Pender, T., 2003. UML Bible, 1st ed. John Wiley & Sons, Inc, New York, NY, USA.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering, in: Visaggio, G. In: Baldassarre, M.T., Linkman, S., Turner, M. (Eds.), 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008). Bari, Italy. British Computer Society, pp. 71–80.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007.

Petre, M., 2013. UML in practice. In: 35th International Conference on Software Engineering. San Francisco, CA, USA. IEEE Press, pp. 722–731. https://doi.org/10.1109/ICSE.2013.6606618.

Pilskalns, O., Williams, D., Aracic, D., Andrews, A., 2006. Security consistency in UML designs. In: 30th Annual International Computer Software and Applications Conference. Chicago, USA. IEEE Computer Society, pp. 351–358. https://doi.org/10.1109/COMPSAC.2006.76.

Reggio, G., Leotta, M., Ricca, F., Clerissi, D., 2013. What are the used UML diagrams? A preliminary survey. In: Proceedings of 3rd International Workshop on Experiences and Empirical Studies in Software Modeling - CEUR Workshop Proceedings (EESSMod 2013). Miami, Florida,USA. pp. 3-12.

Rumbaugh, J., Jacobson, I., Booch, G., 2004. Unified Modeling Language Reference Manual, 2nd ed. Pearson Higher Education.

Simmonds, J., Straeten, R.V., Jonkers, V., Mens, T., 2004. Maintaining consistency between UML models using description LogicZ. RSTI – L'Object LMO'04 10. https://doi.org/10.3166/objet.10.2-3.231-244.

Sjoberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K., Rekdal, A.C., 2005. A survey of controlled experiments in software engineering. IEEE Trans. Softw. Eng. 31, 733–753. https://doi.org/10.1109/TSE.2005.97.

Song, I.-Y., Khare, R., An, Y., Hilsbos, M., 2008. A multi-level methodology for developing UML sequence diagrams. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (Eds.), 27th International Conference on Conceptual Modeling. Barcelona, Spain. Springer-Verlag, pp. 114–127. https://doi.org/10.1007/978-3-540-87877-3_10.

Spanoudakis, G., Zisman, A., 2001. Inconsistency management in software engineering: Survey and open research issues. In: Chang, S.K. (Ed.), Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing Co, Singapore, pp. 329–380.

Thomas, D., 2004. MDA: Revenge of the modelers or UML utopia? http://dx.doi.org/10.1109/MS.2004.1293067.

Torre, D., 2014. On collecting and validating UML consistency rules: A research proposal, Doctoral Symposium at. In: 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014). ACM, London, UK. http://dx.doi.org/10.1145/2601248.2613084.

Torre, D., 2016. Verifying the consistency of UML models. In: 27th IEEE International Symposium on Software Reliability Engineering, Fast Abstract (ISSRE 2016). Ottawa, Canada. IEEE. https://doi.org/10.1109/ISSREW.2016.32.

Torre, D., Labiche, Y., Genero, M., 2014. UML consistency rules: a systematic mapping study. In: 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014). London, UK. ACM. http://dx.doi.org/10.1145/2601248.2601292.

Torre, D., Labiche, Y., Genero, M., Baldassarre, M.T., Elaasar, M., 2018. UML diagram synthesis techniques: a systematic mapping study. In: the 10th Workshop on Modelling in Software Engineering (MiSE'2018). IEEE. https://doi.org/10.1145/3193954.3193957.

Torre, D., Labiche, Y., Genero, M., Elaasar, M., 2015a. https://bit.ly/2xk68Ic (last accessed on May 2018).

Torre, D., Labiche, Y., Genero, M., Elaasar, M., 2015b. UML consistency rules in technical books. In: IEEE International Symposium on Software Reliability Engineering, Fast Abstract (ISSRE 2015). Gaithersburg, MD, USA. IEEE, pp. 68. https://doi.org/10.1109/ISSREW.2015.7392047.

Torre, D., Labiche, Y., Genero, M., Elaasar, M., Das, T.K., Hoisl, B., Kowal, M., 2016. 1st International Workshop on UML Consistency Rules (WUCOR 2015): Post workshop report. SIGSOFT Softw. Eng. Notes 41, 34–37. http://dx.doi.org/10.1145/2894784.2894801.

Usman, M., Nadeem, A., Tai-hoon, K., Eun-suk, C., 2008. In: A Survey of Consistency Checking Techniques for UML Models, Advanced Software Engineering and Its Applications. Hainan Island, China. IEEE Computer Society, pp. 57–62. http://dx.doi.org/10.1109/asea.2008.40.

Wieringa, R., Maiden, N.A.M., Mead, N.R., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requirements Eng 11, 102–107. https://doi.org/10.1007/s00766-005-0021-6.

Wohlin, C., 2014. Writing for synthesis of evidence in empirical software engineering. http://dx.doi.org/10.1145/2652524.2652559.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated. https://doi.org/10.1007/978-3-642-29044-2.

Yang, J., 2009. A framework for formalizing UML models with formal language rCOS. In: 4th International Conference on Frontier of Computer Science and Technology. Shanghai, China. IEEE, pp. 408–416. https://doi.org/10.1109/FCST.2009.72.

Damiano Torre is a Cotutelle Ph.D. Candidate at Carleton University (Canada) and University of Castilla-La Mancha (Spain). He received his bachelor in Technologies for Software Development at the University of Bari, Italy (2009), and his master in Advanced Technologies in Computer Science at the University of Castilla-La Mancha (2011). He has worked in industry and academia in Spain and Canada. He is currently a member of the SQUALL laboratory and a contracted researcher in the Department of Systems and Computer Engineering at Carleton University. His research interests are MDE, Model consistency, and Green Software. He is a member of IEEE.

Yvan Labiche received a bachelor degree in Computer System Engineering and a master's degree in fundamental computer science and production systems in 1995 (Clermont

Ferrand, France), and a PhD in software engineering in 2000 at LAAS/CNRS in Toulouse, France. In January 2001, Dr. Labiche joined the Department of Systems and Computer Engineering at Carleton University as an assistant professor and is now associate professor. His research interests include objectoriented analysis and design, software verification, validation and testing, empirical software engineering and technology evaluation. He is a senior member of the IEEE.

Marcela Genero is Full Professor at the University of Castilla-La Mancha, Ciudad Real, Spain. Her research focuses on empirical software engineering, software quality, conceptual models quality, software modelling effectiveness, etc. She has published more than 100 peer review papers in journals (EMSE, IST, JSS, SOSYM, etc.) and conferences (CAISE, E/R, MODELS, ESEM, EASE, etc.) She co-edited the books titled "Data and Information Quality" (Kluwer, 2001) and "Metrics for Software Conceptual Models" (Imperial College, 2005), among others. She has organized several events on empirical studies in software engineering and quality in conceptual modelling (EASE, ESEM, etc.) She is member of ISERN and ACM.

Maged Elaasar is a software architect with 20 + years of experience. He currently works at NASA's Jet Propulsion Laboratory, leading R&D projects in Model Based Systems Engineering. Prior to that, he worked at IBM for 13 years, where he developed leading UML modeling tools. Maged is also the founder of Modelware Solutions, a software consultancy and training company. He holds a Ph.D. in Electrical and Computer Engineering (Carleton University, 2012). He also authored 12 U.S. patents and 30 + peer-reviewed articles. He also contributed to open software standards at OMG (e.g., UML), and to open-source projects at Eclipse (e.g., Papyrus).