SysML Modeling Mistakes and Their Impacts on Requirements

Mounifah Alenazi*, Nan Niu*, and Juha Savolainen†
* Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
† Global Software and Control R&D, Danfoss Drives A/S, Denmark
alenazmh@mail.uc.edu, nan.niu@uc.edu, juha.savolainen@danfoss.com

***Abstract*—The Systems Modeling Language (SysML) represents a significant and increasing segment of industrial support for building critical systems. Because modeling is a human-centric activity, mistakes are unavoidable. Although there exist several software defect classifications, little is known about the mistakes pertaining to SysML modeling and the implications of those mistakes in model-driven requirements engineering. In this paper, we report a systematic mapping through which 42 SysML modeling mistakes are identified from 19 primary studies. With an emphasis on the evidence of industrial relevance, we further uncover that, despite some mistakes hurt requirements satisfaction, others help make the requirements more complete and the specifications more precise. Our work sheds light on understanding the scope of the SysML mistakes and checking requirements fulfillment in the face of the mistakes.**

***Index Terms*—Systems Modeling Language (SysML), modeling mistakes, model defects, evidence-based software engineering.**

I. Introduction

Unlike in traditional software development where the software (or more exclusively, the working code) is the main artifact, in model-driven development (MDD) the main artifact is a model or a set of models. These models encapsulate the modeler's knowledge and views of the subject system, so that the stakeholder concerns can be managed throughout the development life cycle. For systems engineering applications involving interdisciplinary teams to design, build, and evolve complex systems like railway controls and autonomous vehicles, Systems Modeling Language (SysML) [1] has become a *de facto* choice. Such a choice allows for structural and behavioral representations of the system, and for reasoning about the extent to which the requirements are met [2].

Because modeling is a human activity, mistakes are unavoidable. SysML modeling mistakes can occur for many reasons: human errors during the modeling process, lack of language support at the meta-model level, insufficient or overly constrained tooling, and so on. Orthogonal to the mistake sources, the consequences are typically *defects* manifested in the models themselves. Since these models are the main artifacts of MDD, understanding the defects is imperative.

One kind of understandings is to classify the defects. To this end, prior work has contributed several classification schemes to help distinguish the defect types, characterize the inherent attributes, and inform the resolution strategies. These schemes include the IEEE 1044-2009 standard for software anomalies [3], Chillarege's classification for in-process measurements [4], and Grady's software failure for root-cause analysis [5]. However, existing classifications are not concerned with models in the MDD context, let alone SysML models. In the absence of this knowledge, Briand and his colleagues [6] proposed a defect seeding strategy specific to SysML and further seeded four defect types in their study: incorrect association navigation of a block definition diagram, incorrect association multiplicity of a block definition diagram, incorrect operation ordering of an activity diagram, and incorrect effect on transitions of a state machine diagram. Admittedly, the seeding strategy and the actual defects were based on the researchers' subjective opinions and experience [6].

A different kind of understandings, influenced by the paradigm of evidence-based software engineering [7], focuses on *systematic* literature review or mapping. The goal is not necessarily creating new classifications but collecting evidence of the state-of-the-art, so that the trends of a given field can be depicted and the knowledge gaps can be identified. To that end, Granda *et al.* [8] reported a closely related study in MDD by concentrating on the defects in UML-based conceptual models. A set of 28 articles was selected to serve as the primary studies of their systematic literature mapping. The mapping results indicated a tendency of reporting only "incorrect" defects (80%) rather than "missing" (8%) or "unnecessary" (12%) ones. The work of Granda *et al.* [8] also pointed out the need to develop more mature defect detection mechanisms beyond static methods (e.g., manual or automated inspections, checking consistency rules, and checking OCL constraints).

In this paper, we present a systematic literature mapping of SysML modeling defects to fill the gap elucidated in [6]. Our objective is to collect evidence to not only understand the defects in SysML models presented in the contemporary literature, but to do so with an explicit emphasis on practice, real-world relevance, and industrial readiness. For example, we adopt a hierarchy from our earlier literature review [9] to assess the evidence level of all the selected primary studies. This hierarchy ranges from "no evidence" and "evidence obtained from working out toy examples" on the weaker end to "evidence obtained from industrial case studies" and "evidence obtained from industrial practice" on the stronger end. It is on the basis of these practitioner-oriented criteria that we conduct our survey and analyze the results. Moreover, we discuss the implications of our findings to model-driven requirements engineering, shedding light on the way ahead.

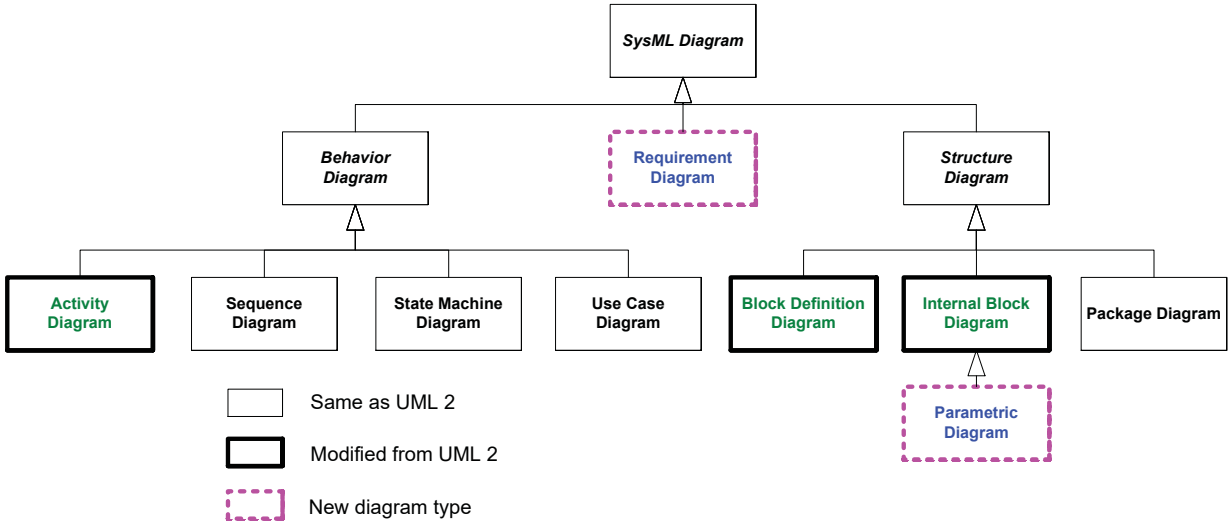

Fig. 1. SysML diagrams and their relationships with UML 2 (*adapted from [10]*).

The remainder of the paper is organized as follows. Section II provides the background of SysML and reviews related work on defect classifications. Section III explains our literature mapping's study design. Section IV analyzes the results in terms of the SysML modeling mistakes, and further discusses how those mistakes link to requirements. Section V presents concluding remarks.

II. Background and Related Work

SysML, first adopted by the Object Management Group (OMG) in 2006, is a visual modeling language designed to provide simple but powerful constructs for modeling a wide range of systems engineering problems [1]. It supports the specification, analysis, design, verification and validation of complex systems that include components for hardware, software, data, personnel, procedures, and facilities. SysML extends UML 2, which tends to be software-centric. It reuses seven of UML 2's fourteen diagrams, and adds two new diagrams (requirement and parametric diagrams) for a total of nine diagram types. Figure 1 shows these diagrams, which cover four main perspectives of systems modeling:

- **Behavior**: The behavior diagrams include the use case diagram, activity diagram, sequence diagram, and state machine diagram. A use case diagram provides a high-level description of functionality that is achieved through user interactions with systems or system parts. The activity diagram represents the flow of data and control between activities. A sequence diagram represents the interaction between collaborating constituencies of a system. The state machine diagram describes the state transitions and actions that a system or its parts perform in response to events.
- **Structure**: The system structure can be represented by block definition diagram which describes the hierarchy of system, subsystems, and all the system elements. The structure can also be shown in internal block diagram which depicts system parts, ports, and connectors. Finally, the package diagram is used to organize the dependencies between the components that make up the system.
- **Parametric**: The parametric diagram represents constraints on system property values such as performance, reliability, and mass properties. It serves as a means to integrate the specification and design models with engineering analysis models.
- **Requirements**: The requirement diagram captures requirements hierarchies and the derivation, satisfaction, verification, and refinement relationships. The relationships provide the capability to relate requirements to one another and to trace requirements to system design models and test cases.

Compared to UML 2, a couple of SysML features offers advantages for systems engineers [11]: (1) SysML reduces the bias of UML toward software since SysML's semantics are more flexible and expressive. In particular, UML classes are replaced with blocks in SysML. A block is a modular unit of structure in SysML that is used to define physical entities (e.g., system, system component part, external systems, or items that flow through the system), as well as conceptual entities or logical abstractions. (2) The built-in requirement diagram allows for natural language requirements to be modeled and traced throughout the system's life cycle.

Across the life span of a system, especially a software-intensive system, defects can appear at different stages and in different forms. Several defect classifications have been proposed in the software engineering literature and are summarized in Table I. While the IEEE 1044-2009 standard [3] focuses on the source code and the orthogonal defect classification [4] focuses on the code change, the other schemes listed in Table I are mainly concerned with early phases of

TABLE I
SOFTWARE DEFECT CLASSIFICATIONS

	IEEE 1044 STD [3]	Orthogonal Defect Classification [4]	HP Scheme [5]	Conceptual Models [8]	SysML Designs [6]
main artifact (life cycle phase)	source code (implementation)	product & process (software change)	process (early phases)	UML diagrams (conceptual modeling)	SysML diagrams (systems design)
main categories (total # of leaf -level categories)	unnecessary, missing, and incorrect (5)	extra, omission, and commission (8)	incorrect, missing, unclear, changed, and better way (23)	unnecessary, missing, and incorrect (6)	incorrect (4)
known usage	testing reporting [12]	vulnerability discovery [13]	process improvement [14]	teaching UML [15]	safety compliance [16]

defect detection and defect prevention (e.g., in requirements engineering and design).

The main artifacts and life cycle phases also shape how the defect classifications are used. Centered around the code, the IEEE 1044-2009 standard is instrumental in software testing (e.g., being an inspiration to get more structure into the incident reporting [12]) and the orthogonal defect classification is applied to understand what might be special about the code defects that compromise the security of a system [13]. Although detecting the erroneous code and code change is important, defects occur in requirements would significantly cripple the resulting system [17]. For this reason, improving software processes cannot afford to overlook the business and requirements angles [14]. Learning conceptual modeling [15] and practicing safety inspections should also pay attention to the various defects [6, 8]. Note that the "known usage" of Table I is based on our knowledge and is meant to illustrate the subtleties of the classification schemes.

Despite the subtleties, the classifications themselves share certain similarities. In most cases, "unnecessary", "missing", and "incorrect" requirements, model elements, code, etc. are considered to be defects and are further differentiated, though the terminologies are by no means unanimous [18, 19]. One can probably map "extra", "omission", and "commission" of [4] to "unnecessary", "missing", and "incorrect" of [3] respectively. To mitigate ambiguity, a hierarchy of sub-categories is often formed.

In the work done by Granda and her colleagues [8], a two-level hierarchy is presented for the UML-based conceptual modeling defects, e.g., "unnecessary" is decomposed into "redundant" and "extraneous". This results in 6 leaf-level categories as shown in Table I. As far as the SysML defects are concerned, the strategy proposed by Briand *et al.* [6] leads to four types of seeds mentioned earlier, all of which are "incorrect" operations injected into existing designs. Questions remain about whether other types of defects like "unnecessary" occur in SysML models, how frequent and severe the defects are, what kinds of models are susceptible to which mistakes, how believable the reported evidence is, etc. These motivate us to search the literature more systematically in order to map the state-of-the-art.

III. MAPPING STUDY DESIGN

Before teasing out our research questions, we clarify the terminology appeared in the relevant literature. According to IEEE 1044-2009, an *anomaly* is: "Any condition that deviates from expectation based on requirements specifications, design documents, user documents, standards, etc. or from someone's perception or experience", whereas a *defect* is: "An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced" [3]. Instead of gearing toward work product and even repair or replacement actions, we choose the term *mistake*[1] to incorporate the human and social aspects in SysML modeling. For example, our intention is to cover mistakes like *error*—"a human action that produces an incorrect result" [3]—so that the cause of a defect, and not just the defect manifested in the work product, could be understood. The implications of the SysML modeling mistakes are surveyed mainly from the MDD requirements engineering perspective in our work.

Due to the broad contextual considerations such as cause and implication, we carry out a systematic mapping study on SysML modeling mistakes. Compared with a systematic literature review, a mapping study follows the same process of formulating research questions, defining literature search criteria, determining primary studies, extracting data, and reporting [7]. Differences include that a systematic mapping deals with a broader research topic and its data extraction and reporting tend to use summaries rather than techniques like meta-analysis or narrative synthesis. Napoleão *et al.* [20] highlighted quality assessment as being the only practical difference between systematic literature reviews and systematic mapping studies. In our work, quality assessment of primary studies emphasizes evidence strength as it relates to MDD practitioners.

A. Research Questions

We set out to answer four research questions:

RQ$_1$: What are the SysML modeling mistakes, their types, and their causes?

RQ$_2$: Which SysML diagrams are subject to the modeling mistakes?

RQ$_3$: What are the evidence levels of the reported SysML modeling mistakes?

[1]Merriam-Webster (http://www.m-w.com/) defines *mistake* as: "a wrong action or statement proceeding from faulty judgment, inadequate knowledge, or inattention". This explanation fits the purpose of our study.

TABLE II
PRIMARY STUDIES LISTED CHRONOLOGICALLY (YEAR OF PUBLICATION) AND THEN WITHIN THE SAME YEAR ALPHABETICALLY (FIRST AUTHOR)

ID	Source	DOI or Grey Literature
PS1	C. Choppy and G. Reggio, "A Method for Developing UML State Machines", in *SAC*, 2009	10.1145/1529282.1529365
PS2	Y. Jarraya, *et al.* "On the Meaning of SysML Activity Diagrams", in *ECBS*, 2009	10.1109/ECBS.2009.25
PS3	R. Karban, *et al.* "MBSE in Telescope Modeling", *International Systems Engineering Newsletter*, 2009	10.1002/inst.200912424
PS4	C. L. Delp, "FireSAT: Model vs Documents Alone", 2010	grey[1]
PS5	L. Mi and K. Ben, "A Method of Software Specification Mutation Testing Based on UML State Diagram for Consistency Checking", in *CEIS*, 2011	10.1016/j.proeng.2011.08.023
PS6	G. Reggio, *et al.* "'Precise is Better Than Light" a Document Analysis Study about Quality of Business Process Models", in *EmpiRE*, 2011	10.1109/EmpiRE.2011.6046257
PS7	Z. Andrews, *et al.* "Model-Based Development of Fault Tolerant Systems of Systems", in *SysCon*, 2013	10.1109/SysCon.2013.6549906
PS8	R. Steiner, "Common SysML Conceptual Stumbling Blocks", in *San Diego INCOSE Mini-Conference*, 2013	grey[2]
PS9	B. K. Aichernig, *et al.* "Model-Based Mutation Testing of an Industrial Measurement Device", in *TAP*, 2014	10.1007/978-3-319-09099-3_1
PS10	S. Ali, *et al.* "Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines?", *Software & Systems Modeling*, 2014	10.1007/s10270-012-0293-5
PS11	É. André, *et al.* "Activity Diagrams Patterns for Modeling Business Processes", *Software Engineering Research, Management and Applications*, 2009	10.1007/978-3-319-00948-3_13
PS12	E. A. Antonio, *et al.* "Verification and Validation Activities for Embedded Systems–A Feasibility Study on a Reading Technique for SysML Models", in *ICEIS*, 2014	10.5220/0004887302330240
PS13	L. Briand, *et al.* "Traceability and SysML Design Slices to Support Safety Inspections: A Controlled Experiment", *ACM Transactions on Software Engineering and Methodology*, 2014	10.1145/2559978
PS14	H. Kruus, *et al.* "Teaching Modeling in SysML/UML and Problems Encountered", in *EAEEIE*, 2014	10.1109/EAEEIE.2014.6879380
PS15	Shannon (GenMyModel Community Manager), "5 Common UML Mistakes", 2014	grey[3]
PS16	S. Feldmann, *et al.* "Towards Effective Management of Inconsistencies in Model-Based Engineering of Automated Production Systems", in *INCOM*, 2015	10.1016/j.ifacol.2015.06.200
PS17	K. Hampson, "Technical Evaluation of the Systems Modeling Language (SysML)", in *CSER*, 2015	10.1016/j.procs.2015.03.054
PS18	S. Pavalkis, "MBSE in Telescope Modeling: European Extremely Large Telescope – World's Biggest Eye on the Sky: Tool Vendor Perspective", in *Space Symposium*, 2015	grey[4]
PS19	H. Sun, *et al.* "Improving Defect Detection Ability of Derived Test Cases Based on Mutated UML Activity Diagrams", in *COMPSAC*, 2016	10.1109/COMPSAC.2016.136

grey[1]: https://mbse.gfse.de/documents/SpaceSystemsIW10.pdf
grey[2]: https://sdincose.org/wp-content/uploads/2013/11/11-Rick-Steiner-SysML-Conceptual-Stumbling-Blocks.r.04.pdf
grey[3]: http://blog.genmymodel.com/5-common-uml-mistakes.html
grey[4]: https://www.spacesymposium.org/wp-content/uploads/2017/10/S.Pavalkis_31st_Space_Symposium_Tech_Track_paper.pdf

RQ$_4$: How do the SysML modeling mistakes impact requirements engineering in MDD practice?

It is important to note that our goal is not to devise a new classification scheme. We thus use the 6 leaf-level categories presented by Granda *et al.* [8] as a baseline: "missing", "inconsistent", "incorrect", "ambiguous", "redundant", and "extraneous". Meanwhile, we are open to emerging categories or facets. It is also worth noting that **RQ**$_4$ has a direct relevance to MDD practice, and for that reason, we choose only those studies with industrial-strength evidence (as opposed to the weaker levels of evidence) to discuss the influences of SysML modeling mistakes on requirements engineering.

B. Search Criteria

Our search for the primary studies was carried out in June 2019 and involved two stages: an automatic one over Elsevier's Scopus and a manual one including the grey literature. We relied on Scopus due to its structured and advanced ways to specify query. Our first attempt issued the search string:

```
TITLE-ABS-KEY ( ( "SysML" OR "SysML diagram" OR "SysML design" OR "SysML model" ) AND ("mistakes" OR "design mistakes" OR "design error" OR "defect") ) AND PUBYEAR > 2006 AND PUBYEAR < 2020 AND ( LIMIT-TO ( DOCTYPE , "cp" ) OR LIMIT-TO ( DOCTYPE , "ar" ) OR LIMIT-TO ( DOCTYPE , "ch" ) ) AND ( LIMIT-TO ( LANGUAGE, "English" ) )
```

Although Scopus returned 14 papers, only one was regarded as relevant by us. We reconsidered the query by removing the year restrictions even though SysML was first adopted by the OMG in 2006. We further expanded the query with the UML diagrams reused by SysML, as well as an additional mistake possibility of "modeling error". The refined search string was as follows:

```
TITLE-ABS-KEY ( ( "SysML" OR "SysML diagram" OR "SysML design" OR "SysML model" OR "UML state machine" OR "UML activity diagram" ) AND ("mistakes" OR "design mistakes" OR "design error" OR "modeling error" OR "defect") ) AND ( LIMIT-TO ( DOCTYPE , "cp" ) OR LIMIT-TO ( DOCTYPE , "ar" ) OR LIMIT-TO ( DOCTYPE , "ch" ) ) AND ( LIMIT-TO ( LANGUAGE, "English" ) )
```

This search resulted in 41 papers. Two researchers collaboratively judged relevance by going through the abstract and the content of each paper. At the end, only five papers were relevant: PS1, PS10, PS11, PS12, and PS19 of Table II. We then engaged in a manual search via Google Scholar to check recursively the references and the citations of those five relevant papers. Unlike the first stage, we included both peer-reviewed publications and grey literature such as presentations, white papers, and blogs. The final list consisted of 19 primary studies as shown in Table II.

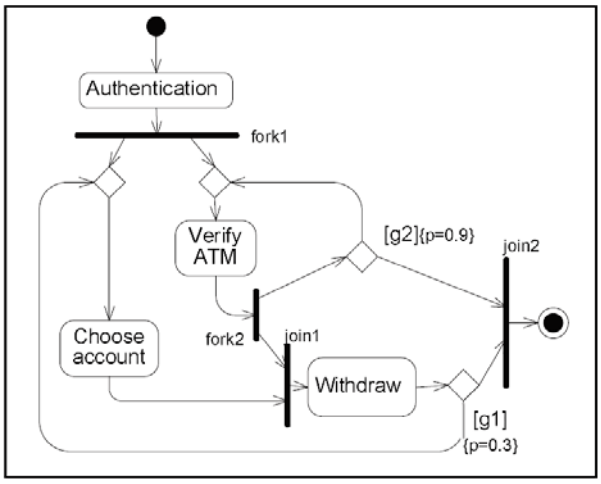

Fig. 2. SysML activity diagram reviewed in our study (*Figure 13 in PS2*).

C. Data Extraction

While the actual extracted data[2] are presented in the next section to answer our research questions, we describe here the process of how we extracted data from the selected primary studies. We used a two-phase process. First, two researchers individually reviewed five randomly selected papers (PS13, PS8, PS5, PS2, and PS12). The researchers followed a pre-defined data extraction form, and then compared their results in a two-hour meeting. The observations were that their agreement levels were high, and consensus was established after the meeting.

As an example of the first phase, the researchers independently reviewed PS2 on SysML activity diagrams. In both data extraction results, the mistake of "a join node placed after a decision node" was recorded. Figure 2 illustrates this mistake with PS2's hypothetical design of the behavior corresponding to banking operations on an automated teller machine (ATM). The guards [g1] and [g2] denote the probability of triggering new operations or looping back of re-performing some earlier operations. In Figure 2, if [g1] is being evaluated twice, i.e., "Choose account" is performed twice, then a deadlock may occur depending on how [g2] is evaluated. The researchers classified this mistake as "incorrect" by following the scheme presented by Granda *et al.* [8]. The other data extraction fields of this mistake were also consistent between the two researchers.

Building on the first phase, we randomly assigned the remaining 14 primary studies to those researchers: 7 per person. The final data extraction results were aggregated and consolidated. We report these results next to answer the four research questions of our mapping study.

IV. Results and Analysis

A. Forty-Two Mistakes

Our literature mapping identifies 42 distinct mistakes. The majority (86%) are presented in only one primary study. All the remaining six mistakes are mentioned in two primary studies. We list them in Table III. Although PS5 and PS10 examine consistency and readability, state machine diagram is what both studies focus on. As a result, all the four mistakes discussed in PS5 are also mentioned in PS10. Additionally, PS10 covers "a state is subsumed by another state" which is also presented in PS1: another study on state machines.

TABLE III
Mistakes mentioned in more than one primary study

Mistake Description (diagram type)	Mentioned
a state is subsumed by another state (state machine)	PS1, PS10
a transition that comes from or leads to a wrong state or moves with wrong conditions (state machine)	PS5, PS10
a transition is missing (state machine)	PS5, PS10
a transition is subsumed by another (state machine)	PS5, PS10
a state is missing (state machine)	PS5, PS10
replacing a fork/join node with a control (activity)	PS15, PS19

Table III shows the positive contributions of the grey literature. Specifically, "replacing a fork/join node with a control" in the activity diagram is not only identified as an incorrectness by the most recent work (PS19), but also recognized as a common mistake in an earlier blog post (PS15). For the four pieces of grey literature that we have surveyed, six mistakes are found only in them. In another word, these six SysML modeling mistakes would be hidden if the grey literature were not explicitly searched or considered.

Adopting the scheme by Granda *et al.* [8] allows us to classify the 42 mistakes identified. Figure 3 shows the classification distribution. The largest proportion ("incorrect") accounts for 45% of the mistakes, including those listed in Table III. In line with the findings in [8], our results confirm that "incorrect" remains the most frequently reported mistake type, though our proportion here (45%) is based on the mistakes (N=42) while that of Granda *et al.* [8] (80%) is based on the primary studies (N=28).

Different from the trend of 8% revealed in [8], "missing" makes up 40% of the mistakes in our data. A closer look shows that many instances are about what are desired but currently "missing". For example, PS3 points out SysML does not differentiate intrinsics between various interfaces like logical and mechanical. Therefore, "missing" in our results includes not only model defects (e.g., "state machine transitions are

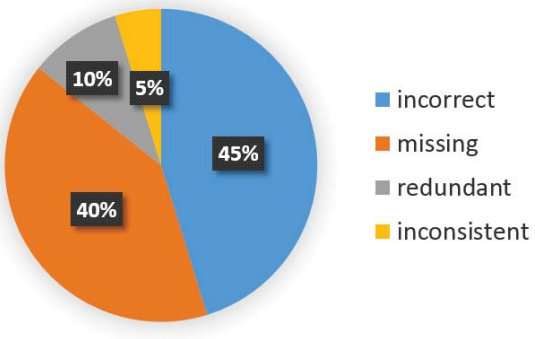

Fig. 3. Distribution of the 42 SysML mistake types.

[2]The entire data of our study are shared in an institution-wide repository, Scholar@UC [21], for replication and cross-validation purposes.

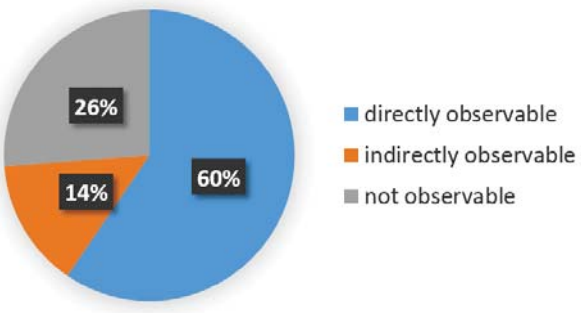

Fig. 4. Observability of the 42 mistakes in SysML models.

modeled without triggers" in PS1), but also *modeling* deficiencies/weaknesses (or even feature requests) like the one pointed out in PS3.

Four mistakes are classified as "redundant", implying the precision of using the leaf-level category ("redundant" or "extraneous") rather than the higher-level one ("unnecessary"). For instance, in SysML activity diagram modeling, PS19 shows that it would be redundant to add a new pair of fork and joint within an existing pair of fork and joint. Our study uncovers two "inconsistent" mistakes. Both are discussed in PS16 and are syntactic in nature: One is about using improper type conversions or castings, and the other refers to using incompatible value, data, or primitive types.

To explore the cause(s) of each identified mistake, we apply open-ended coding without adopting any pre-existing schemes. To our surprise, many primary studies lack the information on this. Referring back to Figure 2, though the mistake is clearly presented in PS2, we could not locate relevant information about why such a mistake happened, who made it, under what circumstances it occurred, etc. The two plausible causes that we have identified from the primary studies are: "meta-model limitation" and "tool limitation". An example of the former is SysML's lack of support for time being a first-class element (PS4), and that of the latter is Cameo's MDD tool may corrupt the model if extensive links are made (PS18).

Given that most causes remain unknown, we turn our attention to how the mistakes manifest themselves in the SysML models. We perform this analysis with three degrees of observability: "directly observable" (e.g., cyclic associations in a block definition diagram discussed in PS15), "indirectly observable" (e.g., shifting down the fork node while lifting up the join node in PS19's study of activity diagrams), and "not observable" (e.g., no timing element in block definition diagram according to PS4). Figure 4 shows the results of our observability analysis, where 60% of the mistakes are directly observable, and hence can be syntactically checked, in the SysML models. The 14% indirectly observable ones would require semantic interpretations that are oftentimes needed from the modelers. Unfortunately, 26% of the mistakes are not observable in the resulting models themselves, many of which are "meta-model limitation" and "missing" mistakes. In another word, if there is currently no way of expressing a certain construct in SysML (meta-model is limited), then that construct will be missing and impossible to observe in existing models.

In summary, 42 mistakes are identified in our mapping study, a majority of which represents incorrectness in SysML modeling. Broadening the mapping study to incorporate causes of the mistakes allows us to show a nontrivial proportion of insufficiencies reported in the literature. These insufficiencies can further lead to new and improved features of the meta-model or the modeling tool. Despite our explicit consideration of causes, they are difficult to extract. Our updated analysis shows that, independent of the causes, a majority of the mistakes can be readily detected in the SysML models syntactically.

B. Five Diagrams

Our study shows that, out of the nine model types of SysML (cf. Figure 1), mistakes have appeared in five diagrams. From the mostly discussed to the least discussed, these five are: activity diagram, block definition diagram, state machine diagram, requirement diagram, and internal block diagram. The number of mistakes specifically applicable to these diagrams is 15, 11, 10, 3, and 1 respectively. The total here is 40, leaving the following two mistakes unaccounted for:

- "extensive linking by modelers has side effects (introduced by changes) as these changes can go unnoticed and corrupt the model" (PS18), which we believe can affect a set of interrelated SysML diagrams; and
- "an open issue about navigation to the different views of a block (mechanical, optical, ...)" (PS3), which is a tool limitation affecting multiple diagram types, e.g., block definition and internal block diagrams.

For reasons of being crosscutting [22], we exclude the above two mistakes from our current $\mathbf{RQ}_2$ analysis. Figure 5 helps visualize the modeling mistakes that the SysML diagrams are susceptible to. In Figure 5a, "missing" appears in all the 5 diagrams. If "missing" suggests new modeling capabilities, then it seems all SysML diagram types are open to improvements. Even though "incorrect" accounts for 45% of the identified mistakes (cf. Figure 3), they are reported to appear in only state machine, activity, and block definition diagrams. One reason may be that these diagrams are used more often; another might be that they are difficult to be correctly practiced.

An interesting pattern of Figure 5b is that, in behavioral diagrams (state machine and activity diagrams), more than half of the mistakes are directly observable. In state machines, this ratio is as high as 90%. Such a pattern does not hold for structural diagrams (block definition and internal block diagrams) or requirement diagram. The results depicted in Figure 5 indicate that, in SysML, the way in which the system works (or is intended to work) can be more readily checked than the way in which the system components are arranged.

In summary, five SysML diagrams are shown to be susceptible to mistakes, and our literature mapping fails to recognize

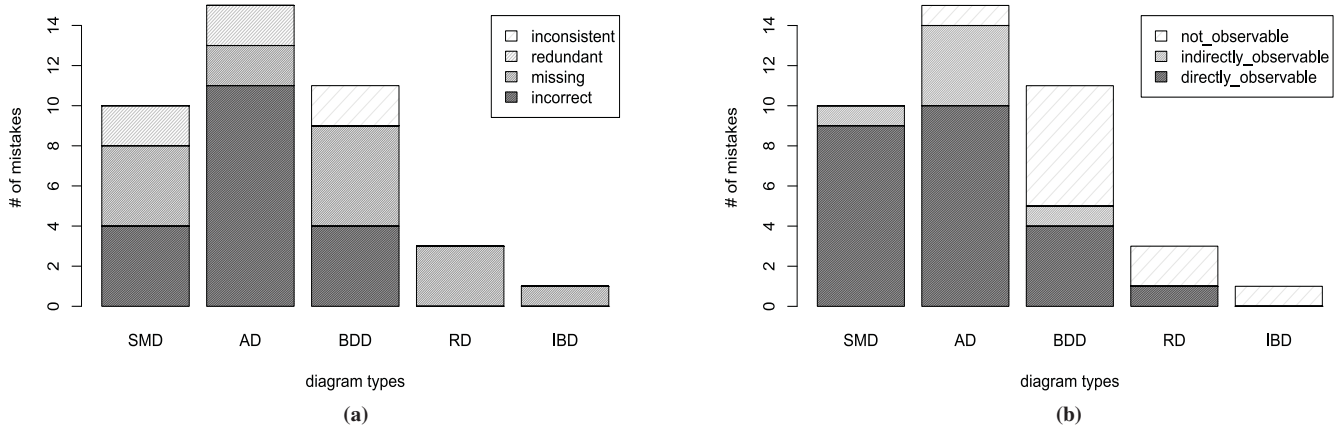

Fig. 5. **(a)** SysML diagrams and mistake types, and **(b)** SysML diagrams and mistake observability. In both cases, the total number of mistakes is 40. SMD: State Machine Diagram. AD: Activity Diagram. BDD: Block Definition Diagram. RD: Requirement Diagram. IBD: Internal Block Diagram.

any mistakes in sequence, use case, package, or parametric diagrams. Although improvements can be made to all diagrams (e.g., filling in the "missing"), "incorrect" practices tend to happen when state machine, activity, and block definition diagrams are built. Once the SysML models are built, the mistakes in behavioral diagrams are easier to spot than those in structural and requirement diagrams. To better understand in what contexts (e.g., research or industry) and application domains SysML models are developed, we next assess the evidence level of the primary studies.

C. Seven Industrially Relevant Pieces of Evidence

Different from answering $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$ where mistakes serve as the units of analysis, we address $\mathbf{RQ}_3$ by treating each primary study as our analysis unit. $\mathbf{RQ}_3$ is our main effort to emphasize industrial relevance because the evidence level reported in the literature is critical for practitioners to believe the research findings. In our context, the strength of evidence would directly inform the practitioners about how likely the SysML modeling mistakes are and how much attention they shall pay to specific mistakes.

Building on our experience in conducting systematic mappings and reviews [9, 23, 24], we use a hierarchy of evidence levels to make our assessment more practical. In Table IV, we present the evidence levels from weakest (top of the table) to strongest (bottom of the table), along with the number of primary studies at each level. Table IV shows that all the primary studies, except for one, provide evidence to contextualize the mistakes, though some are demonstrations explaining the mistakes or word-of-mouth opinions from individual experts. Although academic studies may encompass the rigor of executing well-controlled experiments, we favor the evidence obtained from studying contemporary real-world problems or systems in their industrial contexts.

TABLE IV
EVIDENCE LEVELS OF THE PRIMARY STUDIES

Evidence Level	# of Primary Studies
No evidence	1
Evidence obtained from demonstration or working out toy examples	3
Evidence obtained from expert opinions or observations	5
Evidence obtained from academic studies, e.g., controlled lab experiments	3
Evidence obtained from industrial studies, e.g., causal case studies	4
Evidence obtained from industrial practice	3

Seven primary studies present the SysML modeling mistakes grounded in industrial-strength applications or drawn from the actual practices. The mistakes reported in these studies are rooted in much stronger evidence levels and are thus more believable. The extent to which these mistakes are applicable to other systems engineering projects depends on many factors, including the application domains as well as the size and complexity of the models. We therefore list the relevant information in Table V.

The domains shown in Table V are truly interdisciplinary, requiring expertise in not only software engineering but also automotive, aerospace, and more [25]. Mistakes occur in SysML models of various sizes, ranging from a couple of dozen elements to tens of thousands of elements. This finding suggests that mistakes do not necessarily correlate with size or even complexity; rather, they may be due to human errors or collaboration breakdowns. Another observation of Table V is the crucial role that requirements play in industrial SysML projects. In PS3 and PS4, for example, the requirements information is explicitly presented. Next, we discuss how the SysML modeling mistakes, especially the industrially relevant ones, influence requirements engineering in MDD.

D. Three Impacts on Model-Driven Requirements Engineering

Building on the answers to $\mathbf{RQ}_3$, we address $\mathbf{RQ}_4$ with an interdisciplinary system (namely an emergency response

TABLE V
MISTAKES REPORTED IN INDUSTRIALLY RELEVANT PRIMARY STUDIES

ID	Mistakes	Domain (model size)
PS3 (industrial practice)	hiding internal blocks also hides the nested connector	Optical Telescope (13000 model elements, 700 symbols, 150 diagrams, 50 high level requirements, 50 control systems requirements, refined by 150 use cases)
	no intrinsic differentiation between various interfaces	
	relationship of the ports is hardly shown	
	difficult to relate the different parts into the associated context	
	an open issue about navigation to the different views of a block	
PS4 (industrial practice)	putting activity on internal block diagram creates separate usage	Space Systems; FireSat Mission (10 stakeholder requirements, 12 systems requirements, 4 test requirements, 13 blocks
	impossible for time to be a first-class element	
PS5 (industrial study)	a transition to/from a wrong state or with wrong conditions	Control Sub-System (32 states, 45 transitions)
	a transition is missing	
	a transition is subsumed by another	
	a state is missing	
PS7 (industrial study)	no external send signal action to an event or no data handling can lead to a deadlock	Radio System; Mobile Phone System; Emergency Response (not provided)
PS9 (industrial study)	incorrect time/signal triggers in a state machine diagram	Automotive (19 states, 39 transitions)
PS18 (industrial practice)	extensive linking can corrupt the model	[same as PS3]
PS19 (industrial study)	incorrect guard condition of activity diagram's decision node	Aircrafts (23 model elements)
	input pin or output pin is missing in an activity diagram	

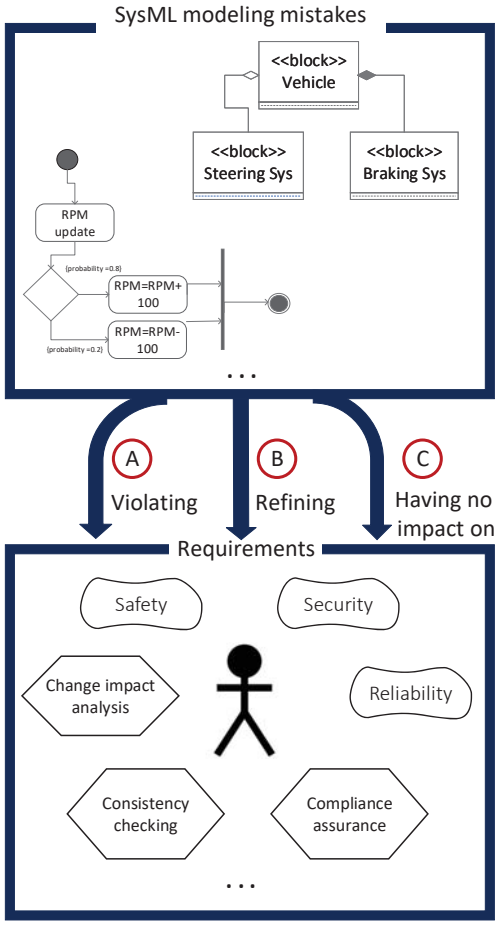

Fig. 6. SysML mistakes' impacts on requirements.

system) by focusing primarily on the mistakes listed in Table V. In addition, the primary studies of Table V provide the goals like "safety" and tasks like "compliance assurance" [26] that a requirements engineer considers and performs in the MDD context. In Figure 6, we thus adopt the i^* graphical notations [27] to represent these softgoals and tasks.

It is not surprising to us that one of the impacts that SysML mistakes have is to lead the requirements to be violated (Ⓐ of Figure 6). To illustrate the impact, we show some SysML modeling fragments in Figure 7, and consider a high-level requirement of the emergency response system: "For every call received, send an emergency response unit (ERU) with correct equipment to the correct target". The mistakes of mixing aggregation and composition in a block definition diagram (① and ② of Figure 7a) would violate the requirement demanding the phone system to be operated and managed by external communication systems and not by the emergency response system. In addition, the multiplicity mistake (③ of Figure 7a) would violate the requirement of allowing the emergency response system to communicate with many ERUs each providing equipment needed for the aid, and not with one and only one ERU.

The SysML modeling mistakes can sometimes be valuable to requirements engineer. In Figure 7b, the mistakes of adding control flows instead of a fork node (④) and adding a join node after a decision node (⑤) could assist in requirements refinement (Ⓑ of Figure 6). In particular, a synchronization from "Divert ERU" to "Log diversion" and to the merge node must be performed. In other words, when there is no ERU available at the time of the rescue event and the case is critical, the diverted ERU shall receive the rescue information immediately. To satisfy this requirement, a fork node should be modeled instead of the control flows. To avoid the deadlock, the join node (⑤ of Figure 7b) shall be removed since it will be waiting for an input that will never be delivered. One refinement option is to relax the timing constraint for the diverted ERU to receive the rescue information, e.g., requiring there exists a next state, rather than requiring all the next states, with tolerable rescue receiving delay. "Start rescue" event (⑥ of Figure 7b) could contribute to the deadlock if there is no external signal sent. Recognizing such a mistake helps uncover new requirements expressing the desire of having some external-signal-sent action in the SysML design.

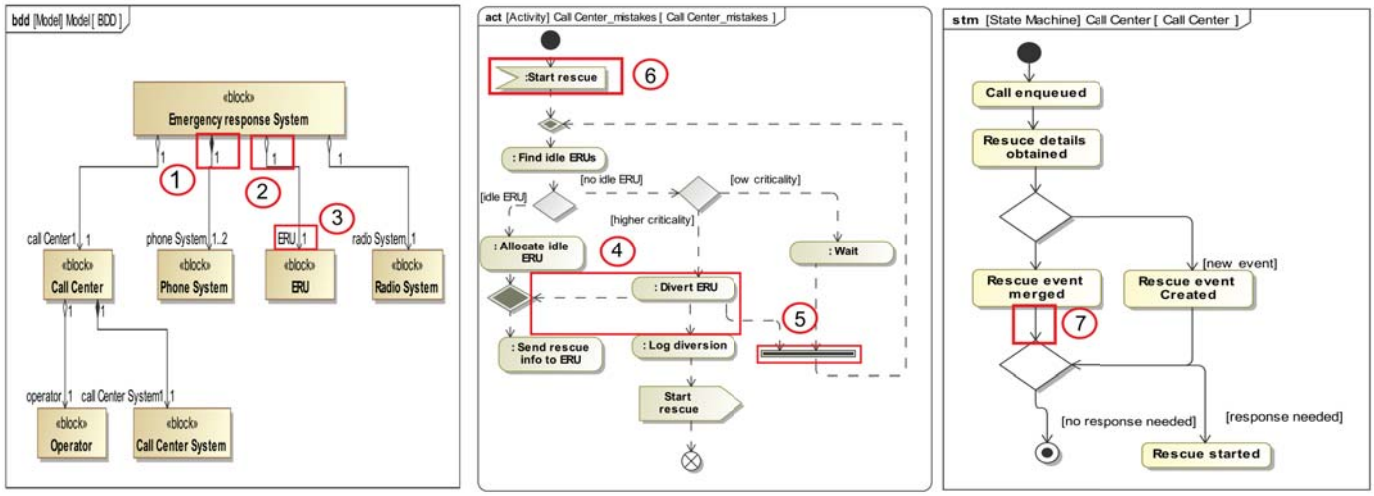

Fig. 7. Illustration of SysML mistakes' impacts on requirements.

To our surprise, a third relation suggests that some mistakes have no impact on requirements (Ⓒ of Figure 6). The state machine diagram of Figure 7 illustrates this. The mistake of modeling transitions without triggers (⑦ of Figure 7c) would neither violate the requirement: "call center shall generate and manage rescue events" nor suggest new or improvement conditions/constraints related to this requirement. One main reason is that the modeling mistake appears outside the design slice of the targeted requirement [6]. This is an important issue given the increasing size and complexity of SysML models and only a set of critical requirements (e.g., safety and security) shall be reasoned about thoroughly to support tasks such as compliance assurance. Understanding the scope of the SysML modeling mistakes, combined with developing better methods to trace critical requirements in the MDD context [28], will be valuable for practitioners to resolve crucial mistakes while tolerating or delaying the resolution of others [29].

V. Concluding Remarks

This paper reports our systematic mapping study of SysML modeling mistakes and the impacts of the mistakes in model-driven requirements engineering. Based on the 19 primary studies, we summarize our mapping results as follows.

- Forty-two SysML modeling mistakes fall into incorrect, missing, redundant, and inconsistent categories. While the causes of the mistakes have not been explicitly reported in the literature, most mistakes can be directly observed and thus syntactically identified in the SysML models.
- Five out of nine SysML diagram types are subject to the modeling mistakes, spanning from structure (block definition and internal block diagrams) through requirement to behavior (state machine and activity diagrams). This could indicate that these diagrams are practiced more often and/or are difficult to be practiced correctly. In line with the work of Granda *et al.* [8], more mature defect detection mechanisms beyond static methods (e.g., manual or automated inspections, checking consistency rules, and checking OCL constraints) should be considered for uncovering behavioral mistakes.
- Seven primary studies show higher-level evidence rooted in industrial studies and practices. Unlike UML, SysML mistakes are made in truly interdisciplinary systems such as space systems and emergency response systems. These industrially relevant studies suggest that modeling mistakes appear no matter how large or complex the system is, and due to the interdisciplinary nature of systems engineering, identifying the mistakes and performing root cause analysis of the mistakes would likely involve subject domain experts and engineers with diverse backgrounds in software, electrical, mechanical, etc.
- Three impacts on model-driven requirements engineering come from SysML modeling mistakes, emphasizing the recognition of requirements engineers' critical concerns like safety and security as well as the tasks that they must accomplish. While certain mistakes, especially those in the "incorrect" category, violate the requirements, not all mistakes should be considered harmful. It turns out that some mistakes (e.g., the "missing" ones) could lead to new requirements to be discovered to alleviate the omissions; yet some other mistakes (e.g., the "inconsistent" ones) could help overcome requirements over-specification or under-specification. Finally, our work calls for better ways of understanding the scope of the modeling mistake so that their impacts on requirements can be properly reasoned about, resolved, or tolerated.

Like all the systematic mapping studies, ours is limited by the literature search strategies implemented. A threat to construct validity is our formulation of search queries and the sources of our search. We combined both automatic and manual search in our work, and included terms such as "defect" and "error" in the queries. Nevertheless, others may have different views about the key construct of "SysML modeling mistakes", and therefore our mapping results shall

be interpreted only within the 19 primary studies that we identified. We believe that the threats to internal validity are minimal due to the descriptive nature of our data extraction effort and the fact that we were not interested in creating any new classification schemes. As far as external and conclusion validities are concerned, we have shared the entire data of our systematic mapping study in an institution-wide repository, Scholar@UC [21], and would welcome replications (including theoretical ones [30, 31]), cross-validations, and evolutions.

Our work reported here can be continued in many directions. We are interested in learning why the four SysML models (sequence, use case, package, and parametric diagrams) are less susceptible to mistakes: Should certain practices be transformed to other model types, or should some diagrams become candidates for depreciation? We also want to better understand the causes behind the mistakes. Of special value is the synergy with human error taxonomies like [32] as MDD and requirements engineering are intrinsically human-centric activities. Although our results show that mistakes happen regardless of model size and complexity, we believe automated tooling [33] is key for practitioners to cope with the SysML modeling mistakes, ranging from detecting and tracing them to resolving or tolerating them.

ACKNOWLEDGEMENT

The work is funded in part by the U.S. National Science Foundation (CCF 1350487).

REFERENCES

[1] Object Management Group, "Systems Modeling Language (SysML)," http://www.omgsysml.org, Last accessed: August 2019.

[2] W. Schäfer and H. Wehrheim, "The challenges of building advanced mechatronic systems," in *International Conference on the Future of Software Engineering (FOSE)*, Minneapolis, MN, USA, May 2007, pp. 72–84.

[3] IEEE Standard Board, "IEEE standard classification for software anomalies," https://standards.ieee.org/standard/1044-2009.html, Last accessed: August 2019.

[4] R. Chillarege, *Orthogonal Defect Classification*. McGraw-Hill, 1996.

[5] R. B. Grady, "Software failure analysis for high-return process improvement decisions," *Hewlett-Packard Journal*, pp. 2:1–2:12, August 1996.

[6] L. C. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, "Traceability and SysML design slices to support safety inspections: A controlled experiment," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 1, pp. 9:1–9:43, February 2014.

[7] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," in *International Conference on Software Engineering (ICSE)*, Edinburgh, UK, May 2004, pp. 273–281.

[8] M. F. Granda, N. Condori-Fernández, T. E. J. Vos, and O. Pastor, "What do we know about the defect types detected in conceptual models?" in *International Conference on Research Challenges in Information Science (RCIS)*, Athens, Greece, May 2015, pp. 88–99.

[9] V. Alves, N. Niu, C. F. Alves, and G. Valença, "Requirements engineering for software product lines: A systematic literature review," *Information & Software Technology*, vol. 52, no. 8, pp. 806–820, August 2010.

[10] S. Friedenthal, A. Moore, and R. Steiner, "OMG SysML tutorial," https://user.eng.umd.edu/˜austin/enes489p/lecture-resources/SysML-Friedenthal-Tutorial-INCOSE2006.pdf, Last accessed: August 2019.

[11] S. Nejati, M. Sabetzadeh, D. Falessi, L. C. Briand, and T. Coq, "A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies," *Information & Software Technology*, vol. 54, no. 6, pp. 569–590, June 2012.

[12] A. M. Hass, *Guide to Advanced Software Testing*. Artech House, 2014.

[13] P. Morrison, R. Pandita, X. Xiao, R. Chillarege, and L. Williams, "Are vulnerabilities discovered and resolved like other defects?" *Empirical Software Engineering*, vol. 23, no. 3, pp. 1381–1421, June 2018.

[14] A. Raninen, T. Toroi, H. Vainio, and J. J. Ahonen, "Defect data analysis as input for software process improvement," in *International Conference on Product-Focused Software Process Improvement (PROFES)*, Madrid, Spain, June 2012, pp. 3–16.

[15] W. A. F. Silva, I. F. Steinmacher, and T. U. Conte, "Is it better to learn from problems or erroneous examples?" in *International Conference on Software Engineering Education and Training (CSEE&T)*, Savannah, GA, USA, November 2017, pp. 222–231.

[16] J. L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R. K. Panesar-Walawege, Á. López, I. del Río, and T. Kelly, "Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel," *Information & Software Technology*, vol. 72, pp. 16–30, April 2016.

[17] F. P. Brooks, *The Mythical Man-Month*. Addison-Wesley, 1975.

[18] N. Niu and S. Easterbrook, "Analysis of early aspects in requirements goal models: A concept-driven approach," *Transactions on Aspect-Oriented Software Development*, vol. 3, pp. 40–72, 2007.

[19] ——, "So, you think you know others' goals? A repertory grid study," *IEEE Software*, vol. 24, no. 2, pp. 53–61, March/April 2007.

[20] B. Napoleão, K. R. Felizardo, É. F. de Souza, and N. L. Vijaykumar, "Practical similarities and differences between systematic literature reviews and systematic mappings: A tertiary study," in *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Pittsburgh, PA, USA, July 2017, pp. 85–90.

[21] M. Alenazi, N. Niu, and J. Savolainen, "Data of SysML modeling mistakes," http://dx.doi.org/10.7945/sz4r-zx36, Last accessed: August 2019.

[22] N. Niu, Y. Yu, B. González-Baixauli, N. A. Ernst, J. C. S. do Prado Leite, and J. Mylopoulos, "Aspects across software life cycle: A goal-driven approach," *Transactions on Aspect-Oriented Software Development*, vol. 6, pp. 83–110, 2009.

[23] G. Valença, C. F. Alves, V. Alves, and N. Niu, "A systematic mapping study on business process variability," *International Journal of Computer Science & Information Technology*, vol. 5, no. 1, pp. 1–21, February 2013.

[24] T. Vale, E. S. de Almeida, V. Alves, U. Kulesza, N. Niu, and R. de Lima, "Software product lines traceability: A systematic mapping study," *Information & Software Technology*, vol. 84, pp. 1–18, April 2017.

[25] M. Alenazi, N. Niu, W. Wang, and B. Vogel-Heuser, "Traceability for evolving automated production systems," in *Grand Challenges of Traceability: The Next Ten Years (GCT)*, Slade, KY, USA, March-April 2017, pp. 28–30.

[26] M. Alenazi, D. Reddy, and N. Niu, "Assuring virtual PLC in the context of SysML models," in *International Conference on Software Reuse (ICSR)*, Madrid, Spain, May 2018, pp. 121–136.

[27] E. Yu, "Towards modeling and reasoning support for early-phase requirements engineering," in *International Symposium on Requirements Engineering (RE)*, Annapolis, MD, USA, January 1997, pp. 226–235.

[28] M. Alenazi, N. Niu, W. Wang, and A. Gupta, "Traceability for automated production systems: A position paper," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Lisbon, Portugal, September 2017, pp. 51–55.

[29] N. Niu, W. Wang, and A. Gupta, "Gray links in the use of requirements traceability," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 384–395.

[30] N. Niu, A. Koshoffer, L. Newman, C. Khatwani, C. Samarasinghe, and J. Savolainen, "Advancing repeated research in requirements engineering: A theoretical replication of viewpoint merging," in *International Requirements Engineering Conference (RE)*, Beijing, China, September 2016, pp. 186–195.

[31] C. Khatwani, X. Jin, N. Niu, A. Koshoffer, L. Newman, and J. Savolainen, "Advancing viewpoint merging in requirements engineering: A theoretical replication and explanatory study," *Requirements Engineering*, vol. 22, no. 3, pp. 317–338, September 2017.

[32] V. K. Anu, W. Hu, J. C. Carver, G. S. Walia, and G. Bradshaw, "Development of a human error taxonomy for software requirements: A systematic literature review," *Information & Software Technology*, vol. 103, pp. 112–124, November 2018.

[33] W. Wang, N. Niu, M. Alenazi, and L. D. Xu, "In-place traceability for automated production systems: A survey of PLC and SysML tools," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3155–3162, June 2019.