Journal of Computer Languages 58 (2020) 100963

Contents lists available at ScienceDirect

Journal of Computer Languages

journal homepage: www.editorialmanager.com/cola/default.aspx

Reviews

Understanding Practitioners' Challenges on Software Modeling: A Survey

Mert Ozkaya[*,a], Ferhat Erata[b]

[a] Yeditepe University, Department of Computer Engineering, Istanbul, Turkey
[b] Yale University, Department of Computer Science, New Haven, CT, USA

ARTICLE INFO

Keywords:
model-driven software development
software modeling challenges
practitioners
survey

ABSTRACT

Software modeling is considered as the high-level design technique for describing abstract statements about software systems. While some practitioners create models for the early analysis of their design decisions and generating code from their models, some practitioners create models for the eased communication among stakeholders. There also exist practitioners who ignore modeling and directly proceed with coding. We aim in this paper to understand the challenges that practitioners face with in their software modeling activities. We surveyed 80 practitioners from 18 countries who work in 18 different industries. We focussed on 8 categories of software modeling challenges: *(i)* managing the language complexity, *(ii)* extending modeling languages, *(iii)* domain-specific modeling environments, *(iv)* developing formal modeling languages, *(v)* analysing models, *(vi)* separation of concerns, *(vii)* transforming models, and *(viii)* managing models. As the results show, the separation of concerns is the least challenging category for practitioners, while analysing models is the top challenging category. Various concrete challenges in different categories have been observed, including *(i)* using the modeling languages with steep learning-curve, *(ii)* extending the language semantics without inconsistencies and updating the language tools accordingly, *(iii)* evolving the DSL tools with new requirements, *(iv)* defining the languages' formal semantics in terms of the translations in any formal languages, *(v)* decomposing models into separate viewpoints and analysing the consistencies between different viewpoint models, *(vi)* the consistent model transformation and the model synchronisations, *(vii)* using model checkers for formal analysis, and *(viii)* versioning models.

1. Introduction

Modeling has been used in many branches of engineering for so many years as the method of abstraction for managing the complexity of systems. With models, complex systems can actually be described in terms of abstract statements and models are considered to be correct if those statements are correct with regard to the systems modeled [1]. Indeed, one cannot construct an automobile that consists of several systems without creating meaningful models and dealing with each model separately. Again, a bridge that are to be used by vehicles cannot be handled without modeling. The situation is exactly the same for software systems, where the complexity of software can be handled by means of abstraction (i.e., modeling) [2,3]. Software modeling is extremely useful in many aspects, including the understandability and communication of the software solutions via a simplified notation, the precision of the modeled solutions, making judgements about the real system to be built, producing the executable systems from models.

To create software models, many alternatives are available essentially. These include the use of simple boxes-and-lines, any natural languages, and the modeling languages with concrete syntax and semantics. While the simple boxes-and-lines and natural languages may essentially be used for communication and documentation, the models created with languages can be processed via the supporting tools. Unified Modeling Language (UML) [4] is one of the most popular software modeling languages, which offers a comprehensive graphical notation set for the modeling of software systems from different perspectives (e.g., logical, behaviour, deployment, information, etc.). Following UML, many software modeling languages have been proposed, including architecture description languages, formal specification languages, domain-specific modeling languages, and UML-based software modeling languages [5,6].

1.1. Motivation and Goal

While software modeling has been actively researched since the nineties and that led to tens of different software modeling languages and their supporting toolset, practitioners in different industries still do not show the expected level of interest to software modeling. As Selic indicated [2], practitioners are concerned about many issues that make them consider models as untrustworthy and requiring too much effort.

* Corresponding author.

https://doi.org/10.1016/j.cola.2020.100963
Received 25 August 2019; Received in revised form 24 January 2020; Accepted 16 March 2020
Available online 06 April 2020

Indeed, models cannot always be specified precisely, which makes it really hard to validate the models before considering them for implementing the actual software systems. Also, there is a semantic gap between models and code that make practitioners have some big difficulties in reflecting their design decisions into the actual software implementation. Software modeling languages suffer from many issues too, which has been revealed with several surveys conducted recently, e.g., [6–9]. These include the steep learning curve of the modeling languages with formal semantics, informal languages' lack of precision and analysis support, languages' weak tool support for forward and reverse engineering, lack of extensibility, lack of support for multiple viewpoints, etc. Given all, one may wish to learn about the challenges that practitioners in different industries face with on software modeling. However, as discussed in Section 4, while many survey studies have been conducted on software modeling, none of them really focus on the practitioners' challenges on software modeling.

So, in this paper, a survey is aimed to be conducted among practitioners who work in diverse industries to understand the categories of challenges that practitioners face with and the concrete challenges of the practitioners in each category. The survey results are expected to be very helpful for understanding the particular types of challenges that practitioners are commonly concerned about in their software modeling activities. The challenges determined herein will also trigger any potential contributions between academia and industry for proposing solutions to the challenges determined.

To determine the categories of challenges, France et al.'s seminal work has been considered [10], which provides a comprehensive and systematic overview of the major challenges on software modeling and enables the reader to understand the different categories of modeling challenges along with the discussions of the concrete challenges in each category that practitioners may face with. Note here that practitioners can either be the meta-modelers who design and develop modeling languages or modelers who use those modeling languages for their specific problems. France et al. essentially proposed 8 different types of challenges, which are *(i)* managing the language complexity, *(ii)* extending modeling languages, *(iii)* domain-specific modeling environments, *(iv)* developing formal modeling languages, *(v)* analysing models, *(vi)* separation of concerns, *(vii)* transforming models, and *(viii)* managing models. Managing the language complexity is concerned with any challenges that meta-modelers face with while describing their language definitions (i.e., syntax and semantics) and modelers' experiences with the syntax and semantics of the languages. Extending modeling languages is concerned with any challenges that modelers face with while using any extensible modeling languages to extend the language definitions for their particular problems. Domain-specific modeling environments are concerned with any challenges that modelers face with while using domain-specific languages (DSLs) and their supporting tools. Developing formal modeling languages is concerned with any challenges that meta-modelers face with while defining the formal semantics of their DSLs. Analysing models is concerned with any challenges that modelers face with while analysing their software models to detect the design errors. The separation of concerns is concerned with any challenges that modelers face with while modeling their software systems from different viewpoints that each deal with a particular concern. Transforming models is concerned with meta-modelers' challenges on designing, implementing, and testing the adequate model transformation tools and techniques and modelers' challenges on using those techniques for transforming the models into any other useful forms. Lastly, managing models is concerned with any issues that modelers face with while performing different modeling activities such as model specification, model versioning, model transformation, and realising models.

2. Research Questions

In this survey study, the following research questions are focussed for the purpose of meeting the goal indicated in Section 1.1.

RQ1 - What levels of challenges do practitioners face with for the different categories of software modeling challenges? In this research question, France et al.'s 8 categories of modeling challenges that have been introduced in Section 1.1 are focussed upon. So, the goal is to *(i)* determine for each challenge category of France et al. the level of cruciality considered by the practitioners and *(ii)* compare the different challenge categories in terms of the cruciality levels chosen by the practitioners. The cruciality levels can be either *(i)* no challenges at all, *(ii)* some challenges, *(iii)* average level of challenges, *(iv)* many challenges, *(v)* lots of challenges. Note here that practitioners are also allowed to type any other challenges that are not given in France et al.'s list.

RQ2 - What are the concrete challenges that practitioners face with in each category of challenges while modeling software systems? In this research question, the goal is to determine for each challenge category of France et al. which concrete challenges are the top-concerns of practitioners. The list of concrete challenges to be considered for each challenge category has been determined from France et al.'s challenge descriptions in [10], which have also been summarised in Section 5. Note that practitioners are also allowed to state any other concrete challenges for each challenge category that are different from those included in the pre-defined list of concrete challenges.

3. Survey Design, Execution, and Sampling

3.1. Survey Design

Previously, we have conducted some other survey studies [7,11], which actually made us gain the necessary knowledge and experience in designing a survey. So, in our survey discussed in this paper, we easily conducted such activities as preparing the survey questions and answers, separating them into meaningful sections, conducting an effective plot study, and searching for the potential participants.

Before releasing the survey, we conducted a pilot study among 5 different participants. These participants include the academics who are highly experienced on the survey design and execution and the practitioners who have considerable experiences on software modeling. Each participant has been contacted via e-mail and given 7 days to return their feedback for the survey questions attached to the e-mails as a pdf file. In the e-mail, the participants have been kindly requested to pay attention to the following set of aspects: *(i)* any missing, ambiguous, or misleading questions and answers, *(ii)* the coherence of the survey sections, *(iii)* any needed supplementary materials, *(iv)* the minimum and maximum amount of time required to fill in the survey, and *(v)* the completeness and consistency of the survey questions with regard to the research questions. As a result, we got very useful feedback from the participants and managed to correct all the issues detected.

After the pilot study, the survey design has been finalised with 19 different questions that are given in Table 1. So, the first two questions are intended for understanding the participants' profiles (i.e., work country and industry). Moreover, as aforementioned, the survey focuses on the list of software modeling challenge categories that has been proposed by France et al. [10]. Thus, a separate survey section is introduced for France et al.'s each challenge category, which consists of two questions. That is, one question is for learning the level of challenges faced with and targets the first research question in Section 2, and another question is for learning the type(s) of challenges faced with on that category and targets second research question. Note here that the question for determining the types of challenges focus on the list of challenges that France et al. discuss for each category. While the questions for determining the level of challenges are designed as single-choice questions, the questions for determining the types of challenges are designed as multiple-choice questions with the free text options. The single-choice answers for a question have been structured precisely

Table 1
The survey questions

Res. Que.	Survey Questions	Multiple Answer	Free Text	Single Answer
Profile Questions	1-Which country do you work in?		X	X
	2-Which industry(ies) do you develop software products for?	X	X	
RQ1	3-Indicate the level of challenges that you face in the category of ”Managing Language Complexity”.			X
RQ2	4-What type(s) of challenges do you face in the category of ”Managing Language Complexity”?	X	X	
RQ1	5-Indicate the level of challenges that you face in the category of ”Extending Modeling Languages”.			X
RQ2	6-What type(s) of challenges do you face in the category of ”Extending Modeling Languages”?	X	X	
RQ1	7-Indicate the level of challenges that you face in the category of ”Domain-specific Modeling Environments”.			X
RQ2	8-What type(s) of challenges do you face in the category of ”Domain-specific Modeling Environments”?	X	X	
RQ1	9-Indicate the level of challenges that you face in the category of ”Developing Formal Modeling Languages”.			X
RQ2	10-What type(s) of challenges do you face in the category of ”Developing Formal Modeling Languages”?	X	X	
RQ1	11-Indicate the level of challenges that you face in the category of ”Analysing Models”.			X
RQ2	12-What type(s) of challenges do you face in the category of ”Analysing Models”?	X	X	
RQ1	13-Indicate the level of challenges that you face in the category of ”Supporting Separation of Design Concerns”.			X
RQ2	14-What type(s) of challenges do you face in the category of ”Supporting Separation of Design Concerns”?	X	X	
RQ1	15-Indicate the level of challenges that you face in the category of ”Transforming Models”.			X
RQ2	16-What type(s) of challenges do you face in the category of ”Transforming Models”?	X	X	
RQ1	17-Indicate the level of challenges that you face in the category of ”Managing Models”.			X
RQ2	18-What type(s) of challenges do you face in the category of ”Managing Models”?	X	X	
RQ2	19-List below the challenges that you face in ”Other Modeling Categories”.		X	

in terms of multiple options one of which may only be chosen by the participants. These options are *(i)* no challenges at all, *(ii)* some challenges, *(iii)* average level of challenges, *(iv)* many challenges, and *(v)* lots of challenges. The multiple-choice answers for a question enable the participants to choose one or more answers for the question. The free-text option enables the participants to type their own answers for the question. So, the participants who are not satisfied with the existing answer list of the question may state their own answers via the free-text option. The free-text answers have been analysed using the coding strategy [12]. That is, each free-text answer for a question is first evaluated to determine whether it can be considered as any of the answers in the answer-list. Otherwise, a new answer is introduced for the question. If the free-text answer does not make sense for the question, we omit that without introducing a new answer. Lastly, the question 19 in Table 1 is included in the survey for letting the participants type in free-text mode any other challenges that do not match with France et al.’s category list.

3.2. Survey Execution

The survey has been available online via google forms[1] and open for accepting responses between March 2019 and August 2019. Our survey essentially focuses on determining the software modeling challenges that practitioners from different industries face with. To this end, we aimed to invite in our survey the participants who have some experiences on software modeling and development. However, we also accepted any participations from those who do not actually have any experiences in software modeling but do have something to say about the challenges on software modeling. By doing so, the goal is to learn also from the participants who have not performed modeling before due to some challenges.

To reach as many practitioners as possible who can fill the survey in, we followed a systematic approach. That is, we initially used our personal contacts that we have obtained from the previous research projects which we have been involved in. We sent e-mails to 50 different industrial partners of our past projects and requested them to participate in the survey and share the survey with anyone whom they know and consider as the potential participant(s) (i.e., snow-balling). As a result, we got 15 participations from our industrial partners. Moreover, we determined 125 different practitioners who contributed to the academic papers that are about software modeling. Note here that we considered the academic papers on software modeling that are highly cited and indexed by google-scholar. However, we just got 10 participations from the authors of the papers. Besides, we determined all relevant mailing lists that focus on software modeling & development. These include the mailing lists of the eclipse (e.g., sirius-dev, graphiti-dev, papyrus-rt-dev, emf-dev, and papyrus-sysml-users), AADL mailing list, Netbeans mailing list, the mailing lists of netbeans, IEEE architecture description mailing list, and some other e-mailing lists (e.g. the apache software foundation, UML Sculptor, and UML DWG). Having sent e-mails to all of these mailing-lists, we got 25 participations from the practitioners who have been registered to the mailing-lists. We also used the social platforms effectively. To this end, we firstly shared the survey link in many linkedin groups, including software architecture, software developer, model driven architecture, software design patterns and architecture, model-based software engineering, and the enterprise architecture network. Besides linkedin, we use the twitter platform and shared the survey on our twitter accounts. We also requested the software modelers followed by thousands of people to retweet (i.e., share) our post. We further shared the survey in some popular software development forums, including code-project, dani-web, code-guru, bytes, eclipse forums, code ranch, and source-forge. While the linkedin groups and twitter brought us approximately 20 different participations, the forums also brought 10 participations. So, as a result, we have received 80 different participations for our survey[2].

3.3. Survey Sampling

We initially determined the sampling unit that represents the cluster of the population from which we have selected the participants. So, we focussed on the group of practitioners who have been involved in software modeling and development in different industries or do have negligible experiences but faced with challenges on software modeling.

To create the sample that suits the definition of our sampling unit, we actually considered two choices – i.e., probability sampling or non-probability sampling. The probability sampling requires that every single member has the equal chance of being selected for the survey participation. In the non-probability sampling, the participants shall be selected non-randomly depending on their conveniences. While the probability sampling is better than the non-probability sampling in minimising the biases as each individual in the population have equal

[1] https://docs.google.com/forms

[2] The survey questions and the collected survey data can be accessible via the following link: https://doi.org/10.5281/zenodo.3571492

chance of being selected randomly, it is not necessarily feasible as we do not have enough time and budget to reach every member of the population. Therefore, we used the non-probability sampling and selected the participants non-randomly in the way explained in Section 3.2. However, we also tried to imitate the random selection by sharing the survey link in the social platforms, including linkedin, forums, and mailing lists, where most of the practitioners who develop software systems have been registered with and are expected to follow the discussions and any posts shared. Indeed, any participants using those social platforms whom we do not know have equal chance of participating in the survey.

4. Related Work

The literature includes several survey studies that aim to reveal the challenges which practitioners face with while performing software modeling activities. Many of these works focus on particular problem domains and determine practitioners' challenges while applying the software modeling for those problems. These include modeling in particular domains (e.g., embedded systems, high-performance computing, and industry 4.0), variability modeling, using UML for modeling, applying product-line engineering for the modeling language developments, tool support for modeling, and considering non-functional properties in modeling. Some other works focus on the modeling languages and surveyed practitioners to understand their perspectives towards different types of languages or analysed the existing languages to determine any missing points. However, unlike our survey, none of these works essentially aimed to learn practitioners' challenges on software modeling in general with no particular focus on any domains and do this in a systematic way that uses any well-established list of general challenges on software modeling.

In [7], the author surveyed among 115 different practitioners from 28 different countries to understand practitioners' knowledge and experience on the formal and informal software modeling languages. In his survey, the author also aimed to understand practitioners' motivations/demotivations for the modeling languages. In [8], Malavolta et al. interviewed 48 practitioners from 40 IT companies with the goal of understanding their expectations from the architecture modeling languages. Malavolta et al. revealed in their survey the practitioners' challenges on the architectural languages, including formal languages' steep learning curve and the lack of support for the model analysis. In [13], Forward et al. surveyed 113 practitioners to understand their experiences on software modeling and why many practitioners stay away from software modeling. To this end, Forward et al. asked 18 questions to different types of practitioners, including software developer, software modeler, code generators, software veterans, and real-time developers, and analysed their survey results separately. In [14], Liebel et al. surveyed 113 practitioners so as to understand their use of the model-based software development (MBSD) for the embedded systems domain. Liebel et al.'s survey further identified a number of challenges, such as the weak tool support (e.g., tool integration and interopability) and the high effort needed for the MBSD training. In [15], Mohagheghi et al. surveyed the 25 papers that discuss some empirical studies on model-driven engineering (MDE) and aimed to understand the industries where MDE is used, MDE's level of maturity in those areas, MDE's effect on the productivity and software quality properties. While Mohagheghi et al. focused mainly on the advantages of MDE, some challenges have also been reported including the lack of support for the scalable modeling environment and domain-specific modeling, using models in software processes, the complexity of modeling, specifying platform-independent models, and creating models for legacy systems. In [16], Berger et al. focused on the variability modeling and surveyed 42 practitioners so as to understand the modeling notations and tools used by the practitioners for the variability modeling and practitioners' perspectives towards the variability modeling (i.e., the benefits and challenges). Concerning the challenges, Berger et al. considered any complexity issues that practitioners suffer from, which includes the model visualisation, dependency management, model evolution, configuration process, and traceability. In [17], Lange et al. surveyed 80 architects to understand their UML usage for the modeling of software systems from the viewpoints of Kruchten [18]. The architects also stated the challenges on their UML modeling that are about the scattered information, incompleteness, disproportion, and inconsistency. In [19], Tomassetti et al. surveyed 155 Italian practitioners so as to understand the extent to which the companies in Italy apply modeling in their software development projects. Tomassetti et al.'s survey results indicated some challenges such as the lack of popularity in industry, complex modeling notations (e.g., UML's variety of diagrams), weak tool support for code-generation, the lack of support for an integrated toolset for the entire development life-cycle (including versioning and deployment). In [20], Wortmann et al. focus on the use of modeling in industry 4.0 and analysed 408 different publications that consider modeling for industry 4.0. Wortmann addressed several research questions that are concerned about the modeling in industry 4.0, including the benefits of using modeling languages for industry 4.0, the challenges that practitioners face with in modeling for industry 4.0, the modeling languages used, and the research methods employed by the papers. In [21], Kosar et al. analysed 390 different papers on DSLs so as to understand the current researches conducted on the DSL development and the observed challenges. Kosar et al. considered many research questions including DSLs' contributions, the types of research methods used by DSLs, DSLs' focus area, the trends in the DSL research, the top-cited DSL papers, the most active institutions on the DSL research, the number of publications per year, and the top conferences/journals. In [22], Czech et al. analysed 189 best practices from 19 different publications that are employed for dealing with the challenges on developing the domain-specific modeling languages and their tools. Czech et al. grouped the best practice challenges in terms of the planing, analysis, design, implementation, maintenance, and testing phases categorised them depending on their relevances (i.e., the domain model, language design and concepts, language notation, generators, DSL-tooling, metamodel tooling, and the entire DSM-solution). In [23], Mendez-Acuna et al. analysed 38 different articles that discuss the application of the software product line engineering for the DSL development and considered their support for the life-cycle of the language product line engineering, the technological spaces of the approaches (the support for abstract and concrete syntaxes and semantics), and the open issues on the language product line engineering. The open issues are intended for revealing the challenges on the analysis, design, testing, and evolution of the language product line engineering. In [24], Lelandais et al. focus on modeling in the area of high performance computing (HPC). To this end, Lelandais et al. considered two of their projects that offer the modeling platforms (with the DSL and modeling editor support) for designing the simulators to be used in HPC. Lelandais et al. stated many lessons learned from those projects on applying the model-driven approach on HPC. Moreover, Lelandais et al. also stated the challenges observed, including the interaction between heterogenous models, the lack of tool support for adopting DevOps, extending the meta-models and models, multiple users' concurrent access on the same model, and technological dependencies. In [25], Storrle surveyed 96 practitioners to understand the practitioners' motivations for modeling and how they create models. Storrle aimed to determine to what extent UML (or any other conceptual modeling languages) is used in industry, the reasons for the practitioners to create software models, the different usage modes (i.e., the activities in which the models are used), and the frequency of the user modes. Storrle also received the participants' opinions about modeling, which revealed some challenges such as lack of adequate tool support, generating code, and formal modeling. In [26], Whittle et al. interviewed 17 practitioners to determine a taxonomy of the modeling tools, which categorises the modeling tool support in terms of the technical factors, internal and external organisational factors, and social factors. Each

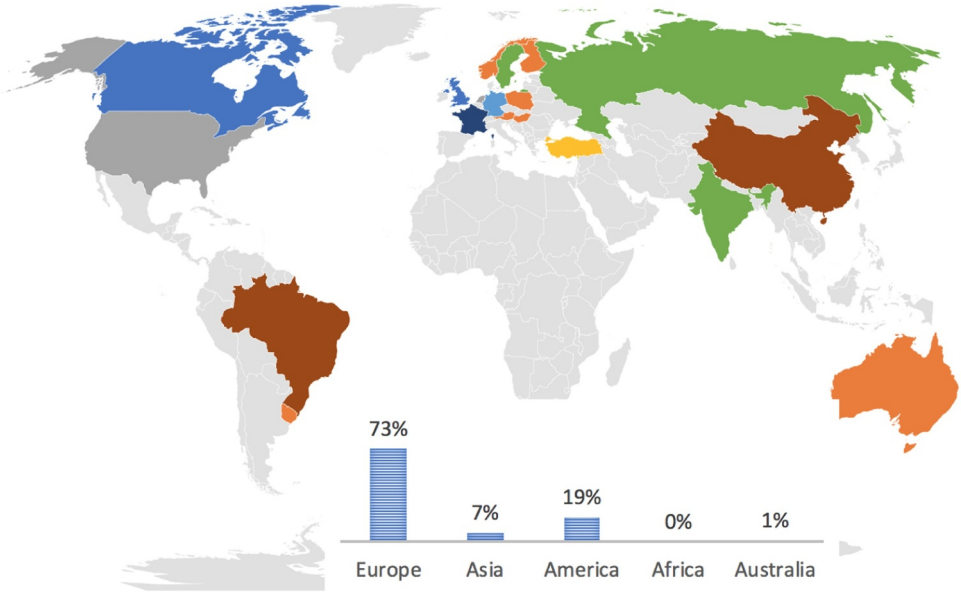

Fig. 1. The countries of the participants

factor has been categorised into some generic tool-related features, which have been further sub-categorised into specific features. Whittle et al. later on interviewed another set of practitioners to understand what kinds of challenges that the interviewees face with regarding each of these features. In [27], Ameller et al. interviewed the practitioners from 18 companies so as to understand to what extent the non-functional properties (NFPs) of software systems are considered in the software modeling activities. To this end, Ameller et al. focused on understanding the existing model-driven development (MDD) approaches' support for NFPs, practitioners' experiences with modeling the NFPs, and the benefits of managing NFPs during software modeling. Ameller et al. also considered the challenges that practitioners face with when the NFPs are not supported by the MDD approaches.

5. Survey Findings and Results

5.1. Q1: Participants' Country of Work

The survey attracted participants from 18 different countries. As Figure 1 shows, most of the participants are from the countries in the Europe continent, which is followed by those from America. While the number of the participants from the countries in Asia is relatively less, we could not attract any from Africa. Note that just 1% of the participants are from Australia.

5.2. Q2: Participants' Work Industries

As Figure 2 shows, the top industries in which the participants work are the IT & Telecommunications and Automotive & Transportation. These are followed by the Finance & Accounting, Defense/Military & Aviation, Research, and Computer industries. The industries such as Retail, Semiconductor, Robotics, and Supply Chain showed the least interest to the survey.

5.3. Q3, Q5, Q7, Q9, Q11, Q13, Q15, Q17: The level of challenges that the participants face with for each category of modeling challenges

Figure 3 shows the level of challenges that the participants face with for each category of challenges considered in the survey. So, the separation of design concerns is relatively less concerning for the participants. Indeed, the percentage of the participants who chose either "no challenges at all" or "some challenges" is the greatest for the separation of design concerns. Extending modeling languages is not so concerning either because of the same reason. However, analysing models can be considered highly concerning for many participants, given the fewest participants who have chosen the "no challenges at all" or "some challenges" levels. Likewise, developing formal languages and managing language complexity are also quite concerning. The other categories of the modeling challenges (i.e., transforming models, domain-specific modeling environments, and managing models) are essentially found challenging at an average level by many participants.

5.4. Q4: The type(s) of challenges that practitioners face in the category of "Managing Language Complexity"

France et al. discuss a list of challenges on managing the language complexity. Meta-modelers occasionally need to modify the language definition that consists of the language syntax and semantics for new domain requirements. However, this may cause a number of challenges, including the consistency of the changes made on the meta-model, the changed meta-model's impact on the other elements, and the completeness of changed meta-model. Also, modelers may be challenged by the languages whose notation set requires a steep learning curve and thus difficult to learn and use. Another challenge that modelers may face with is to do with using a modeling language with a small notation set that may not necessarily be helpful for modeling a large set of problems in different domains of industries.

As Figure 4 shows, modelers' top concern (43-45%) is the languages' complex notation sets with steep learning curve. On the other hand, practitioners have got the least concerns on using any languages with a small notation set for the modeling of diverse domains of problems. Some participants stated other challenges on managing the language complexity that are not included in the question's answer list. These challenges are on *(i)* defining/using the design patterns/styles, *(ii)* the languages' ambiguous notation sets, *(iii)* the lack of user guides that explain when/how to use the language notations, *(iv)* composing multiple languages together to benefit their capabilities at the same time, *(v)* the lack of technical support teams, and *(vi)* the lack of support for the information modeling (e.g., data flow, data state, and data life-cycle).

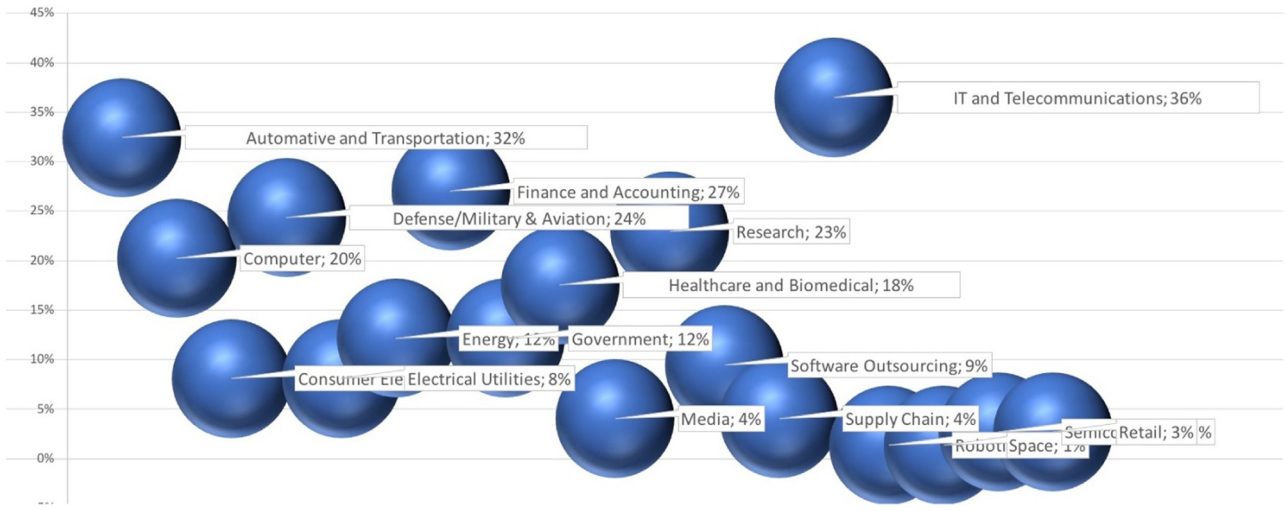

Fig. 2. The work industry of the participants

5.5. Q6: The type(s) of challenges that practitioners face in the category of "Extending Modeling Languages"

Many extensible modeling languages have been proposed so far, which support various extension mechanisms (e.g., annex description, tool support, sub-typing, inheritance, property annotation, and XML) for extending the language syntax and semantics definitions for different domain needs [6]. Modelers who use the extensible languages may however face with challenges that hinder the extension process for the specific needs of the modelers. These challenges are on extending an extensible language with some domain-specific notations, modifying the semantics of the language, providing a new semantics definition for the language (e.g., the formal, process-algebraic semantics for formal analysis), ensuring the consistency between existing and newly-defined language semantics, and updating the language tools in accordance with the language semantics.

As Figure 5 shows, the top concerns of the language users are to do with adding some new language semantics without causing any inconsistencies and updating the language tools in accordance with the newly defined semantics. Modifying the existing semantics of a language or introducing new notations are not so challenging for the users.

5.6. Q8: The type(s) of challenges that practitioners face in the category of "Domain-specific Modeling Environments"

Domain-specific modeling environments enable the modelers to create domain-specific models and perform some useful operations on them such as the quality property analysis and code-generation. AADL [28] is for instance one of the most popular domain-specific languages (DSLs) for the real-time embedded systems domain, which offers several tools for the modeling, analysis, simulation, and implementation of the embedded systems. While domain-specific modeling environments help the modelers address many domain-specific problems, modelers may face with several challenges in their use of DSLs and their tools. One of the challenges herein may be due to the need for using the different DSL versions on the same model so as to benefit the facilities that come with the newer versions. Another possible challenge can be on using and integrating multiple DSLs together for the same problem and interchanging the models created in different DSLs. Also, the tools provided for the DSLs do not necessarily satisfy the modelers, and therefore, modelers may wish to evolve the DSL tools for the domain needs (e.g., adding new analysers, code generators, and simulators), which may cause some challenges for the modelers.

As Figure 6 shows, the most challenging issue herein is evolving the DSL tools with some new domain requirements. This is followed by the

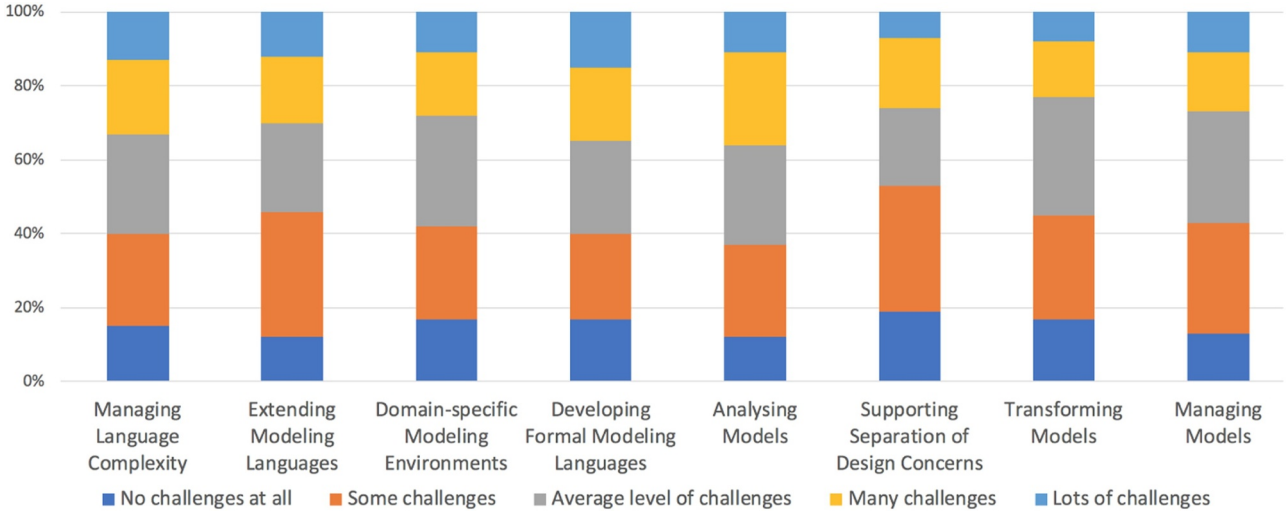

Fig. 3. The level of challenges that the participants face with for each category of modeling challenges

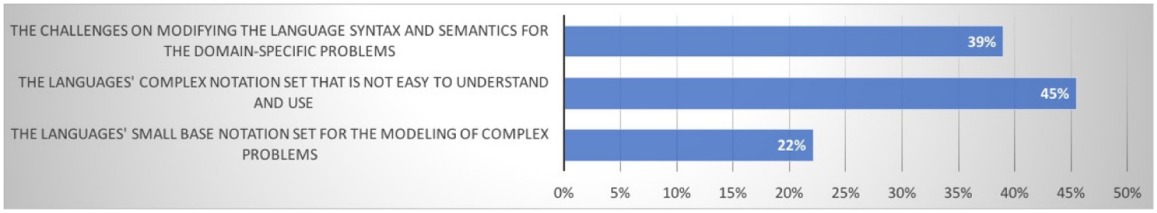

Fig. 4. The type(s) of challenges that practitioners face in the category of "Managing Language Complexity"

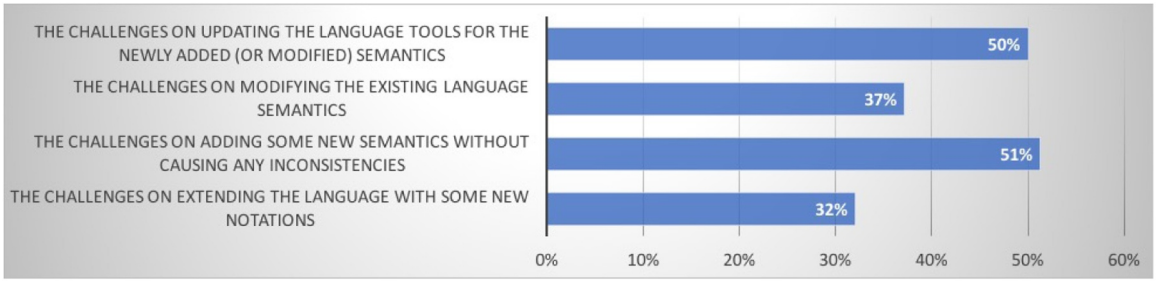

Fig. 5. The type(s) of challenges that practitioners face in the category of "Extending Modeling Languages"

challenge of integrating multiple DSLs together. On the other hand, most participants are not concerned about having to use different versions of the same DSLs on the same model. A few participants stated some other challenges on the domain-specific modeling platforms that are not included in the above given list of the question. These include the challenges on *(i)* the co-evolution of the language and its models and *(ii)* the weak tool support and technical support teams. Also, one participant indicated that the current DSLs focus more on the computer science domains rather than the application domains.

5.7. Q10: The type(s) of challenges that practitioners face in the category of "Developing Formal Modeling Languages"

To enable the precise modeling and formal analysis, the semantics of DSLs need to be defined formally. The semantics of DSLs can be defined formally by means of the translations in any formal specification languages that are based on formal methods [29]. Note here that such DSLs are actually known as the formal modeling languages. The formal models corresponding to the DSL models that satisfy the semantics definitions of the DSLs can can then be formally analysed via the supporting model checkers or theorem provers of the formal specification languages (e.g., the SPIN model checker of the ProMeLa language [30]). However, the meta-modelers may face with many different challenges here in the process of defining the formal semantics of their DSLs. Firstly, it may not necessarily be easy to devise the translation algorithms for translating DSLs' high-level notation sets into the lower-level (and most probably algebraic) notation sets of the formal specification languages. Indeed, the formal specification languages that the meta-modelers use may require a steep learning curve (i.e., difficult to learn and use) [8]. Also, developing a translator tool for automating the translation between the DSL model and formal model in accordance with the formal semantics can be considered as another challenge. Moreover, one may need to verify the correctness of the translations between the DSL and formal models that are performed by the tools.

As Figure 7 shows, nearly half of the participants are concerned about the formal specification languages' steep learning curve, designing the algorithms for translating the DSL models in any formal specification language, and ensuring the correctness of the translation. Integrating the translators with DSLs for automatically translating the models created in DSLs in any formalisms is not challenging for many participants. Some participants further indicated different challenges from those given in the question's answer list. These challenges are that *(i)* the formal modeling languages are not applicable in practice and *(ii)* the existing DSLs that are integrated with model checkers do not visualise the model checking results to a more understandable format, *(iii)* languages' weak tool support, and *(iv)* formalising the natural languages in different domains and disciplines.

5.8. Q12: The type(s) of challenges that practitioners face in the category of "Analysing Models"

Modelers may wish to analyse their software models to check that the software models satisfy the system's functional and non-functional properties. So, if the software models are analysed to be correct, one can then proceed with implementing the design decisions. One of the potential challenges herein is to do with the formal model analysis. As discussed in Section 5.7, the formal modeling languages may provide modelers with translators that can produce formal models in accordance with the language semantics. Then, the formal models can be exhaustively analysed via the supporting model checkers (or theorem provers). However, the formal model analysis herein may be challenging due to several reasons, such as the high-learning curve for the model checking tools and their input languages (i.e., the formal specification language in which the translation is performed), the state-space explosion problem, the lack of visualisation for the analysis results, etc. Simulating the model behaviours is another area of concern, which enables to simulate the execution of the systems over models and observe the system behaviours to detect any anomalies or make predictions about the system properties. Modelers may find the model simulation challenging due to some reasons such as the language tools' simulation support, the simulation techniques supported, visualisation support, etc. Lastly, modelers' use of the model-based testing techniques [31] may be considered challenging, which are essentially concerned with generating test-cases from the system requirements models.

As Figure 8 shows, many of the participants (62%) are concerned about analysing the software models exhaustively using the formal techniques. While model simulation is also quite challenging for the participants, model-based testing is not so. A few participants indicated some further challenges that are not listed. The lack of model analysis tools (and guidelines) that are equivalent to the program analysis tools/guidelines (e.g., SEI CERT[3] and MISRA[4]) and that promote the model

[3] https://tools.netsa.cert.org/
[4] https://www.misra.org.uk/

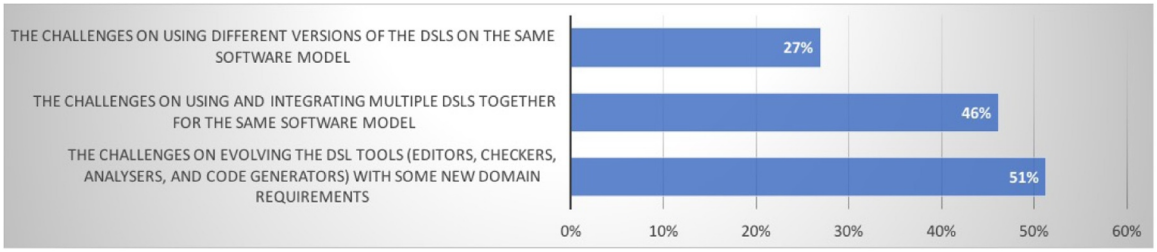

Fig. 6. The type(s) of challenges that the participants face in the category of "Domain-specific Modeling Environments"

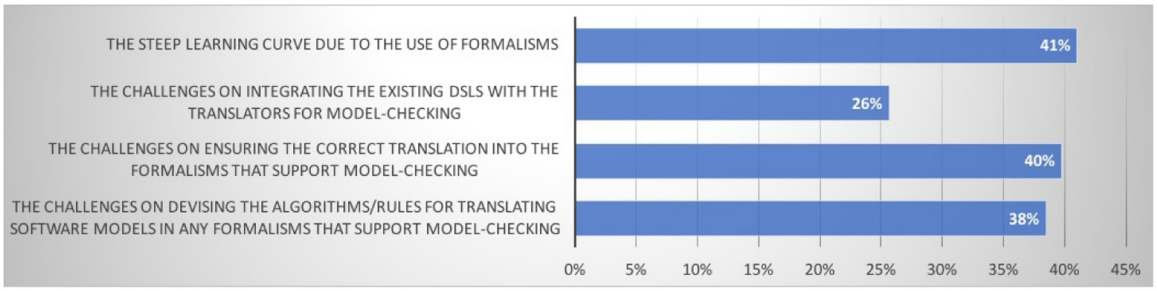

Fig. 7. The type(s) of challenges that practitioners face in the category of "Developing Formal Modeling Languages"

analysis for the functional/non-functional system requirements is one of the concerns herein. Also, some participants face with challenges on the model traceability analysis. Another concern is the lack of languages/tools for performing exhaustive model analysis without having to be involved in any formalisms.

5.9. Q14: The type(s) of challenges that practitioners face in the category of "Supporting Separation of Design Concerns"

The separation of design concerns aids the modelers in decomposing their large and complex models in terms of different viewpoints that each describe the design decisions on a particular issue (e.g., behaviour, concurrency, physical, operational, information, and development). Modelers however may face with several problems herein. Firstly, modelers may be challenged with figuring out how to separate the software architecture specifications of their systems into a set of interrelated viewpoints and deal with each viewpoint and their relationships separately. Also, the existing DSLs may not necessarily be helpful for the multiple-viewpoints modeling due to the reasons such as the lack of support for *(i)* re-using the popular viewpoint frameworks (e.g., Kruchten's 4+1 viewpoint [18]), *(ii)* creating new viewpoints consisting of architectural rules and constraints, *(iii)* and processing the viewpoint models for, e.g., analysis and code generation. Another potential challenge can be on the capability of analysing the different viewpoint models regarding their relationships, e.g., any inconsistencies between the viewpoint models, the completeness of the viewpoint models. Indeed, while modelers sometimes use the DSLs with the multiple-viewpoints modeling support, the DSL tools may not provide the analysis support at an adequate level. Also, modelers may wish to analyse their viewpoint models for their user-defined properties. Lastly, modelers may prefer to use the aspect-oriented software modeling. The aspects herein each represent any functional/non-functional requirements that concern the different components of the system but are unrelated to the components' main functionalities [32]. So, separating the aspects from system components enhances the components' modularity and their analysability. While many modeling languages and software frameworks have been existing that use aspects, it is unclear to what extent modelers are satisfied with those approaches and whether they face with any challenges or not.

As Figure 9 shows, the top challenging issues herein are the ability of separating software architectures into different viewpoint models and analysing the relationships between the viewpoint models (44-46%). While the use of DSLs for the multiple-viewpoint modeling is not so concerning, the aspect-oriented modeling techniques are not considered as challenging by most of the participants. Besides the above discussed challenges on the separation of design concerns, a few participants proposed their own challenges. These challenges are on *(i)* the development viewpoint modeling, *(ii)* modeling software systems at different granularity levels, *(iii)* the lack of support for the multiple users working on the different viewpoint models at the same time, and *(iv)* the successful composition of the different viewpoint models to an entire system model.

5.10. Q16: The type(s) of challenges that practitioners face in the category of "Transforming Models"

Model transformation is essentially modelers' one of the main motivations for specifying the models of their software systems. Indeed, models can be transformed for many different reasons, such as modifying the models to enhance the quality, improving the model understanding and communication, model analysis, and model implementation. Modelers who transform their models may actually face with some challenges that affect the quality of the transformation. So, to learn about those challenges, we consider the model refactoring, model refinement, model abstraction, model synchronisation, consistency of the model transformation, and testing the model transformation. Model refactoring is concerned with improving the software structure models without changing their behaviours. Model refinement is concerned with adding new, further details on the software models. Model abstraction is concerned with reducing the software model complexity without damaging its precision. Sychronisation transformation is concerned with the changes on a model that trigger another change in another model. Modelers may also wish to ensure that the transformation is performed consistently such that the target and source models do not conflict with each other. Lastly, testing the model transformation to validate that the transformation is performed correctly is another challenge here, which actually concerns the meta-modelers who are supposed to develop the translators for the DSLs.

As Figure 10 shows, the top-concerns are to do with ensuring the consistent transformation from the source model into the target model and keeping the source and target models synchronised. While the model abstraction is also found quite challenging, the other possible challenges (i.e., model refactoring, model refinement, and testing the model

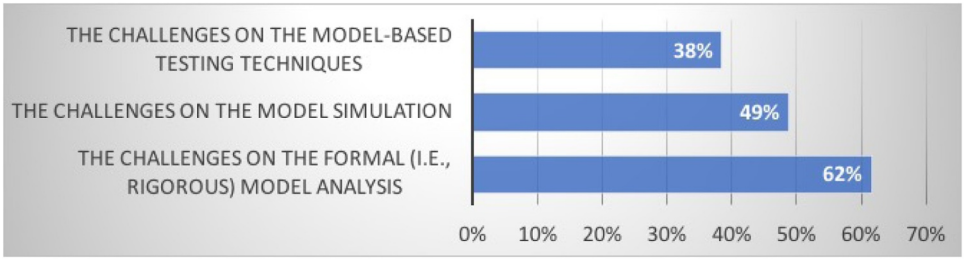

Fig. 8. The type(s) of challenges that the participants face in the category of "Analysing Models"

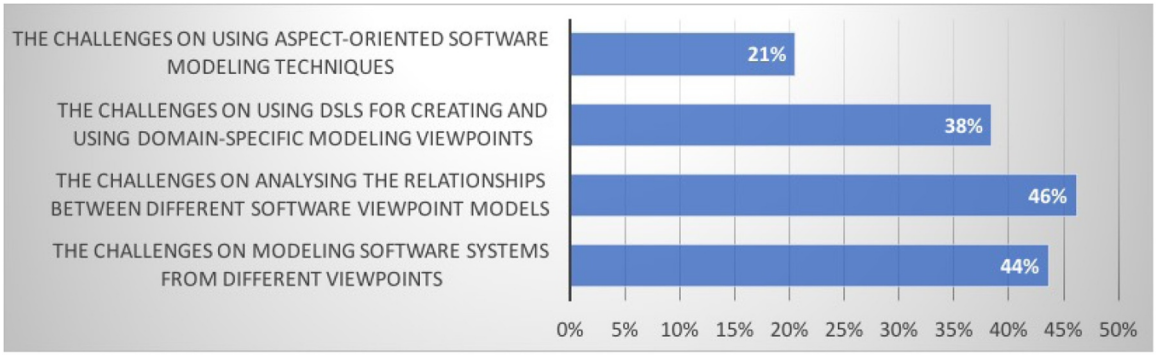

Fig. 9. The type(s) of challenges that the participants face in the category of "Supporting Separation of Design Concerns"

transformation) are relatively less concerning for the participants. A few participants indicated that they face with other challenges on *(i)* keeping the code and model synchronised even after modifying either of them depending on the results obtained via the model/program analysis tools and *(ii)* the lack of tool support for model transformation.

5.11. Q18: The type(s) of challenges that practitioners face in the category of "Managing Models"

To perform model-driven software development, many activities need to be performed such as specifying the models with DSLs, and analysing, versioning, tracking, transforming, and realising the models. Modelers may face with challenges on the management of the above modeling activities. Realising models is concerned with implementing the software models in the same way as specified. Model versioning is concerned with creating and managing model versions (e.g., comparing, tracking, and merging different versions). Maintaining the model repository is concerned with keeping different versions of the models in an internal/external repository center. Managing the model dependency relationships is concerned with relating different models to each other. Lastly, using the modeling tools that allow for extending the language meta-models for model manipulations (e.g., model transformation, model analysis, model documentation) may sometimes be challenging for the language users.

As Figure 11 shows, participants' top-concern is the model versioning. Managing the dependency relationships between different models is also challenging for nearly half of the participants. On the other hand, the model realisation is the least challenging issue for the participants. Some participants indicated further challenges that are not given in the question's answer list. These include the challenges on *(i)* using the distributed version control systems for models, *(ii)* merging the parts from different model versions into a new model version, and *(iii)* the lack of tool support for reviewing models.

5.12. Q19: List below the challenges that you face in "Other Modeling Categories" on which you would consider collaborations with software modeling researchers

Some of the participants (24%) indicated some other challenges that they face with on software modeling and are not included in the categorisation of France et al. These are to do with *(i)* the software product-line engineering, *(ii)* automating the model migration, *(iii)* the multiple-user modeling with interdisciplinary teams, *(iv)* maximising the usability

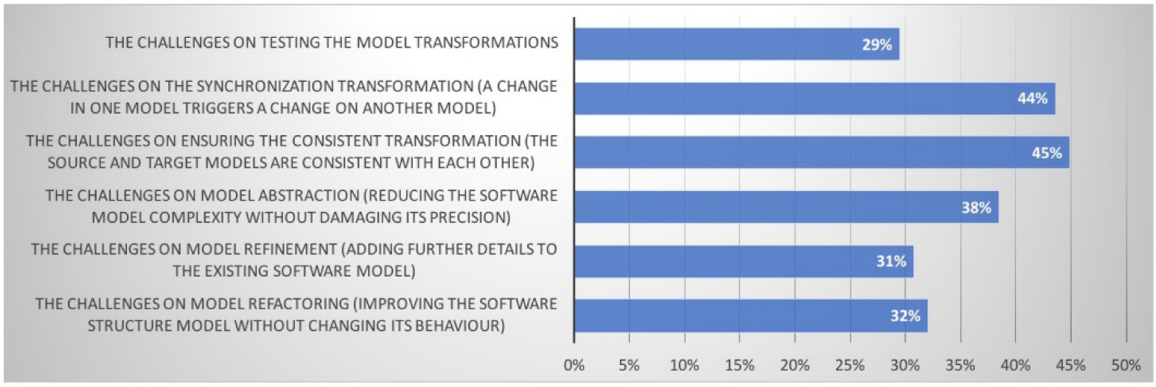

Fig. 10. The type(s) of challenges that the participants face in the category of "Transforming Models"

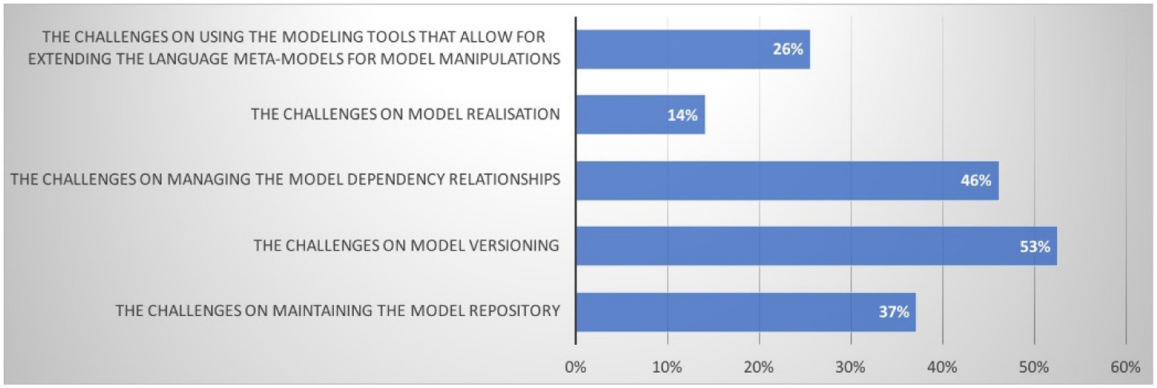

Fig. 11. The type(s) of challenges that the participants face in the category of "Managing Models"

of the language and its toolset, *(v)* managing multi-level of precisions to enable multiple users from different backgrounds to work on the same model, *(vi)* reusing models via the use of model templates, *(vii)* automating the modeling process, *(viii)* tool support for creating OMG meta-models and working with them, *(ix)* achieving simplicity in modeling and meta-modeling, *(x)* model execution and debugging, *(xi)* languages' tool complexity that require huge effort to learn and use, *(xii)* hybrid modeling with textual and visual notation sets together, and *(xiii)* a standard for exchanging models between different tools and technologies.

6. Discussions

6.1. Lessons Learned

While practitioners in industry face with several challenges on software modeling, the existing survey studies in the literature are not so helpful in terms of understanding the software modeling challenges without focusing on any particular domains in a way that considers various possible challenges in general. So, our survey discussed in this paper focuses on France et al.'s extensive study on the software modeling challenges and let us learn many lessons about practitioners' experience with those challenges identified by France et al. Firstly, we learned that the separation of design concerns is the least challenging aspect of software modeling for the practitioners, while analysing models is the top challenging issue. We also learned some crucial lessons about each category of software modeling challenges. Practitioners indicated some concrete challenges on the complexity of the languages. Indeed, while modelers are concerned about the languages' steep learning curve, meta-modelers are concerned about modifying the definitions of the languages (i.e., syntax and semantics). While the extensible languages are supposed to be highly useful for adapting domain-specific concepts and tools, modelers have been observed to face with challenges on introducing a new language semantics without causing any conflicts with existing language semantics and updating the language tools accordingly. Concerning the domain-specific modeling environments, modelers' main concern is to do with the DSLs' tool support for integrating new plug-in tools that can perform the required tasks by the modelers. The formal modeling languages that enable the precise and formally analysable models have also been found to be challenging due to their steep learning curve. While some formal languages are supported with the exhaustive model checkers (or theorem provers) for proving the model correctness, modelers are unfamiliar to such analysis tools that accept formal models. Another lesson learned here is that meta-modelers who develop DSLs find it challenging to develop and test the translator tools that may translate the DSL models in any formal languages to benefit the supporting model checkers of the formal languages. While separating software models into different viewpoints is highly important for the modular and understandable model specifications, modelers expressed their concerns on the ability of decomposing any software models into different viewpoints depending on their domain requirements and analysing the relationships between different viewpoint models to ensure their consistencies. Concerning the modelers' experience on model transformation, their top challenges are on ensuring the consistency between the source and target models and enabling the synchronisations between different models. Lastly, we also learned that modelers find it challenging to version their software models so as to perform such facilities as comparing, tracking, and merging different model versions.

Besides the lessons learned about the survey results, we also learned some important lessons from the feedback received about the survey questions that can be considered for any future surveys. Firstly, the survey's profiling questions aim to learn basically the participants' work countries and work industries. To maximise the anonymity of the participants, we did not include any other profiling questions to learn, e.g., the participants' role in modeling (modeler or meta-modeler), the academic degrees and majors, the years of experiences on software modeling, and the types of software projects involved. Indeed, given our experiences from the previous surveys, many practitioners may not wish to share specific information about their profiles. However, the lack of the detailed profile information essentially prevented us from determining any correlations between the participants' profiles and the software modeling challenges. Another lesson is to do with the questions that aim to learn the types of software modeling challenges in each category considered in the survey. We provided a set of challenges for each of those questions and expected the participants to choose among those pre-defined list of challenges or state their own challenges that are not included in the pre-defined list. However, we actually did not provide any questions to learn the participants' motivations behind the challenges that they choose. Therefore, we were not able to learn what make the participants face with those challenges or any software modeling languages/techniques that cause the challenges.

6.2. Threats to Validity

6.2.1. Threats to Internal Validity

The internal validity is intended for ensuring that no any survey results are affected by the unknown variables that unexpectedly affect the survey results and cause biases. To avoid any threats against the internal validity, we shared the survey among the potential participants who are expected to have some experiences on software development and preferably software modeling. Note that we are also interested in the participants who are involved in software development but barely describe software models due to some challenges. So, this minimised the chance of receiving answers from the participants who are likely to misunderstand the questions and provide biased answers due to the lack of knowledge on software development and modeling.

Also, the unknown variables may be introduced due to the non-random selection of the participants. While we did not have the chance

of reaching every single practitioner randomly, we tried to mimic the random selection via social platforms. Indeed, we sent messages to several mailing lists that focus on software modeling & development (e.g., eclipse mailing lists, AADL mailing list, Netbeans mailing list, the mailing lists of netbeans, IEEE architecture description mailing list), shared the survey link in many the linkedin groups on software modeling & development and our twitter accounts, and sent posts to the popular software development forums. So, the participants who use the social platforms that we interacted have each got an equal chance of accessing the survey randomly.

To strengthen the internal validity of the survey results, we selected the participants from diverse profiles represented with the country of work and work industry. By doing so, any biases due to being restricted with a particular set of profiles are avoided. Indeed, the survey includes participants from 18 different countries and several different work industries including automative, consumer electronics, energy, finance, IT, retail, government, and healthcare. Note however that we did not include any other profiling questions in our survey for learning, e.g., the participants' age, academic degrees & subjects, project types, and job positions. This is essentially because we aimed to maximise the simplicity of the survey and attract more participants. Indeed, our past experiences on the survey design and execution and the feedback received via the pilot study show that many participants do not wish to give any personalised information or at least lose time answering profiling questions. However, omitting such kinds of profiling questions unfortunately prevented us from discussing the internal validity of the survey results in a more precise way, as it is not currently easy for us to determine the factors that may affect the participants' answers and establish some worthwhile correlations.

Lastly, to avoid any experimenter biases, the collected data of the survey has been analysed by each author of the paper separately. The authors compared their analysis for each survey question, and, if any discrepancies detected, the authors conducted a discussion session to reach a consensus.

6.2.2. Threats to External Validity

The external validity is intended for ensuring that the survey results can be generalised to the entire population. To maximise the external validity, we did our best to reach as many different profiles of the population as possible. So, we succeeded to reach 80 different participants from 18 different countries who work in 18 different industries.

6.2.3. Threats to Construct Validity

The construct validity is for ensuring that the survey results are satisfactory in terms of answering the research questions of the survey. To maximise the construct validity of the survey results, we determined the set of survey questions that are associated with each research question as shown in Table 1. We further divided the survey questions into sections with regard to the categories of modeling challenges considered in this study. To avoid any biases here, we firstly asked in each section the level of challenges that the participants face with, which is essentially for satisfying the first research question. If the participants have no challenges in that category, we directed the participants to the next section without answering the questions of the current section. Otherwise, the participants are prompt to state their concrete challenges in that category, which is intended for the second research question. The answers of the participants for each survey question have been analysed statistically via the MS Excel office application.

The construct validity can be threatened when the constructs (i.e., the software modeling challenges) are measured with a single exemplar only, which is known as the mono-operation bias. To avoid the mono-operation bias, we intended to spread the survey among many platforms that are internationally recognised and accessed by the participants from diverse countries and industries. So, we reached 80 participants from 18 different countries and 18 different work industries.

Another potential threat results from the mono-method bias, which is concerned with using a single method to measure the construct. In this survey, we focused on France et al.'s 8 categories of software modeling challenges to understand the modeling challenges that practitioners face with. While France et al.'s study is well-regarded by the software engineering community and highly cited in thousands of papers, one may consider France et al.'s research quite old that was conducted 13 years ago and may therefore miss some newly emerging categories of modeling challenges which the industry face with. To reduce the bias here, we allowed the participants to type any other categories of challenges that the participants face with in industry via the last question of the survey.

7. Conclusion

In this paper, a survey has been conducted among 80 different participants from 18 different countries with the goal of understanding the challenges that practitioners face with on software modeling. The survey focuses on eight different categories of software modeling challenges and aims to determine which categories are more challenging for practitioners and the concrete challenges faced with in each category. The challenge categories considered are *(i)* managing the language complexity, *(ii)* extending modeling languages, *(iii)* domain-specific modeling environments, *(iv)* developing formal modeling languages, *(v)* analysing models, *(vi)* separation of concerns, *(vii)* transforming models, and *(viii)* managing models. As the survey results indicate, practitioners face with many concrete challenges in each category that reveal several interesting issues from modelers' and meta-modelers' point of views that have remained unclear so far. Separation of concerns has been observed to be the least challenging category of software modeling for the practitioners, while analysing models has been observed to be practitioners' top challenge. Some of the concrete challenges for different categories are listed as follows: *(ix)* languages' notational complexities that are difficult to learn and use, *(x)* changing the definition of an existing language for new requirements, *(xi)* extending the language semantics without inconsistencies and updating the language tools in accordance with the new semantics, *(xii)* evolving the DSL tools with the domain-specific requirements and integrating multiple languages together, *(xiii)* developing and testing the translators that translate the DSL models into formal models in any formal verification language for formal analysis, *(xiv)* using the model checker and theorem prover tools to perform formal analysis on the formal models, *(xv)* decomposing models into different viewpoint models that each deal with a particular concern and ensuring the consistencies between different viewpoint models, *(xvi)* ensuring the correct and consistent transformation between the source and target models, and *(xvii)* managing different versions of the models (e.g., comparing, tracking, and merging versions).

The results of this survey are expected to be highly useful for the language developers to understand the most challenging issues about software modeling and develop their languages (or evolve the existing languages) in a way that bridges those gaps determined. Moreover, the tool vendors may use the survey results to improve their modeling tools for better addressing the needs of practitioners. The survey results are also expected to trigger similar researches on other fields of software engineering, such as mobile software development, and embedded software development. Lastly, the results will be extremely useful for the academia, who may wish to conduct researches on the relevant topics with the software modeling challenges determined in our study and collaborate with practitioners to propose innovative approaches that address those challenges.

In the near future, we plan to validate the survey results via some case-studies. To this end, we will consider the XIVT[5] and PANORAMA[6] projects, which are labeled by the European Union's EUREKA Cluster

[5] XIVT web-site: https://itea3.org/project/xivt.html

[6] PANORAMA web-site: https://itea3.org/project/panorama.html

programme ITEA (Information Technology for European Advancement). We will design and execute case-studies for different industries in which the projects' industrial partners work, such as Arcelik for consumer electronics, Turkcell for telecommunications, and AVL for automotive. Each case-study will focus on a real problem that can be solved with a software development in one of the industries considered. The case-studies will be designed by following the well-accepted principles (e.g., [33]) and prompt the practitioners to experience each category of software modeling addressed in the survey so as to observe the concrete challenges that the industry faces with. Each case-study is to be conducted over a pre-determined group of practitioners who work in the corresponding company, and we will collect the data with our observations and a survey that is to be conducted on those practitioners after they perform the case-study.

Disclosure of conflicts of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Seidewitz, What models mean, IEEE Software 20 (5) (2003) 26–32, https://doi.org/10.1109/MS.2003.1231147.

[2] B. Selic, The pragmatics of model-driven development, IEEE Softw. 20 (5) (2003) 19–25, https://doi.org/10.1109/MS.2003.1231146.

[3] C. Atkinson, T. Kühne, Model-driven development: A metamodeling foundation, IEEE Softw. 20 (5) (2003) 36–41, https://doi.org/10.1109/MS.2003.1231149.

[4] J. Rumbaugh, I. Jacobson, G. Booch, Unified Modeling Language Reference Manual, The (2Nd Edition), Pearson Higher Education, 2004.

[5] M. Ozkaya, Analysing uml-based software modelling languages, Journal of Aeronautics and Space Technologies 11 (2) (2018) 119–134. http://www.rast.org.tr/JAST/index.php/JAST/article/view/326

[6] M. Ozkaya, The analysis of architectural languages for the needs of practitioners, Softw., Pract. Exper. 48 (5) (2018) 985–1018, https://doi.org/10.1002/spe.2561.

[7] M. Ozkaya, Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling? Information & Software Technology 95 (2018) 15–33, https://doi.org/10.1016/j.infsof.2017.10.008.

[8] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What industry needs from architectural languages: A survey, IEEE Transactions on Software Engineering 99 (2012), https://doi.org/10.1109/TSE.2012.74.

[9] D. Akdur, V. Garousi, O. Demirörs, A survey on modeling and model-driven engineering practices in the embedded software industry, Journal of Systems Architecture - Embedded Systems Design 91 (2018) 62–82, https://doi.org/10.1016/j.sysarc.2018.09.007.

[10] R. France, B. Rumpe, Model-driven development of complex software: A research roadmap, 2007 Future of Software Engineering, FOSE '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 37–54, https://doi.org/10.1109/FOSE.2007.14.

[11] M. Ozkaya, What is software architecture to practitioners: A survey, in: S. Hammoudi, L.F. Pires, B. Selic, P. Desfray (Eds.), MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016. SciTePress, 2016, pp. 677–686, , https://doi.org/10.5220/0005826006770686.

[12] R. Popping, Analyzing open-ended questions by means of text analysis procedures, Bulletin of Sociological Methodology/Bulletin de Méthodologie Sociologique 128 (1) (2015) 23–39, https://doi.org/10.1177/0759106315597389.

[13] A. Forward, T.C. Lethbridge, Perceptions of Software Modeling: A Survey of Software Practitioners, Technical Report, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada K1N 6N5, 2008.

[14] G. Liebel, N. Marko, M. Tichy, A. Leitner, J. Hansson, Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice, Software & Systems Modeling 17 (1) (2018) 91–113, https://doi.org/10.1007/s10270-016-0523-3.

[15] P. Mohagheghi, V. Dehlen, Where is the proof? - a review of experiences from applying MDE in industry, in: I. Schieferdecker, A. Hartman (Eds.), Model Driven Architecture – Foundations and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 432–443.

[16] T. Berger, R. Rublack, D. Nair, J.M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, ACM, New York, NY, USA, 2013, pp. 7:1–7:8, https://doi.org/10.1145/2430502.2430513.

[17] C.F.J. Lange, M.R.V. Chaudron, J. Muskens, In practice: UML software architecture and design description, IEEE Software 23 (2) (2006) 40–46, https://doi.org/10.1109/MS.2006.50.

[18] P. Kruchten, The 4+1 view model of architecture, IEEE Software 12 (6) (1995) 42–50, https://doi.org/10.1109/52.469759.

[19] F. Tomassetti, M. Torchiano, A. Tiso, F. Ricca, G. Reggio, Maturity of software modelling and model driven engineering: A survey in the italian industry, 16th International Conference on Evaluation Assessment in Software Engineering (EASE 2012), (2012), pp. 91–100, https://doi.org/10.1049/ic.2012.0012.

[20] A. Wortmann, O. Barais, B. Combemale, M. Wimmer, Modeling Languages in Industry 4.0: An Extended Systematic Mapping Study, Software and Systems Modeling (2019) 1–28, https://doi.org/10.1007/s10270-019-00757-6.

[21] T. Kosar, S. Bohra, M. Mernik, Domain-specific languages: A systematic mapping study, Information & Software Technology 71 (2016) 77–91, https://doi.org/10.1016/j.infsof.2015.11.001.

[22] G. Czech, M. Moser, J. Pichler, Best practices for domain-specific modeling. a systematic mapping study, 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), (2018), pp. 137–145, https://doi.org/10.1109/SEAA.2018.00031.

[23] D. Méndez-Acuña, J.A. Galindo Duarte, T. Degueule, B. Combemale, B. Baudry, Leveraging Software Product Lines Engineering in the Development of External DSLs: A Systematic Literature Review, Computer Languages, Systems and Structures (2016), https://doi.org/10.1016/j.cl.2016.09.004.

[24] B. Lelandais, M.-P. Oudot, B. Combemale, Applying model-driven engineering to high-performance computing: Experience report, lessons learned, and remaining challenges, Journal of Computer Languages 55 (2019) 100919, https://doi.org/10.1016/j.cola.2019.100919.

[25] H. Störrle, How are conceptual models used in industrial software development?: A descriptive survey, Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE'17, ACM, New York, NY, USA, 2017, pp. 160–169, https://doi.org/10.1145/3084226.3084256.

[26] J. Whittle, J.E. Hutchinson, M. Rouncefield, H. Burden, R. Heldal, A taxonomy of tool-related issues affecting the adoption of model-driven engineering, Software and Systems Modeling 16 (2) (2017) 313–331, https://doi.org/10.1007/s10270-015-0487-8.

[27] D. Ameller, X. Franch, C. Gmez, S. Martnez-Fernndez, J. Araujo, S. Biffl, J. Cabot, V. Cortellessa, D. Mndez, A. Moreira, H. Muccini, A. Vallecillo, M. Wimmer, V. Amaral, W. Bhm, H. Bruneliere, L. Burgueo, M. Goulo, S. Teufl, L. Berardinelli, Dealing with non-functional requirements in model-driven development: A survey, IEEE Transactions on Software Engineering (2019), https://doi.org/10.1109/TSE.2019.2904476. 1–1

[28] P.H. Feiler, B.A. Lewis, S. Vestal, The SAE architecture analysis & design language (AADL): A standard for engineering performance critical systems, IEEE Intl Symp. on Intell. Control, (2006), pp. 1206–1211, https://doi.org/10.1109/CACSD.2006.285483. //aadl.info

[29] E.M. Clarke, J.M. Wing, Formal methods: State of the art and future directions, ACM Comput. Surv. 28 (4) (1996) 626–643, https://doi.org/10.1145/242223.242257.

[30] G.J. Holzmann, The SPIN Model Checker - primer and reference manual, Addison-Wesley, 2004.

[31] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz, Model-based testing in practice, Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002), (1999), pp. 285–294, https://doi.org/10.1145/302405.302640.

[32] R. Filman, T. Elrad, S. Clarke, M. Akşit, Aspect-oriented Software Development, first, Addison-Wesley Professional, 2004.

[33] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2) (2009) 131–164, https://doi.org/10.1007/s10664-008-9102-8.