User Experience for Model-Driven Engineering: Challenges and Future Directions

Silvia Abrahão * Francis Bordeleau † Betty Cheng ‡ Sahar Kokaly
§ Richard F. Paige ¶ Harald Störrle ‖ and Jon Whittle **
* Department of Computer Systems and Computation, Universitat Politècnica de València, Spain
Email: sabrahao@dsic.upv.es
† CMind, Gatineau, Canada
Email: francis.bordeleau@cmind.io
‡ Department of Computer Science and Engineering, Michigan State University, US
Email: chengb@cse.msu.edu
§ McMaster Centre for Software Certification, McMaster University, Hamilton, Canada
Email: kokalys@mcmaster.ca
¶ Department of Computer Science, University of York, York, United Kingdom
Email: richard.paige@york.ac.uk
‖ QAware GmbH, Aschauer Str. 32, 81549 München, Germany
Email: Harald.Stoerrle@qaware.de
** Faculty of Information Technology, Monash University, Melbourne, Australia
Email: jon.whittle@monash.edu

***Abstract*—Since its infancy, Model Driven Engineering (MDE) research has primarily focused on technical issues. Although it is becoming increasingly common for MDE research papers to evaluate their theoretical and practical solutions, extensive usability studies are still uncommon. We observe a scarcity of User eXperience (UX)-related research in the MDE community, and posit that many existing tools and languages have much room for improvement with respect to UX. Industrial feedback indicates that UX is an important factor in the dissemination and adoption of new technologies, where UX is a key focus area in the software development industry. We consider this a fundamental problem that needs to be addressed in the community if MDE is going to gain widespread use. In this vision paper, we explore how and where UX fits into MDE by considering motivating use cases that revolve around different dimensions of integration: model integration, tool integration, and integration between process and tool support. These use cases help us to illuminate MDE-related UX challenges. Based on the literature and our collective experience in research and industrial collaborations, we propose future directions for addressing these challenges.**

I. Introduction

As computing-based systems continue to increase in volume and complexity, more industrial organizations are considering model-driven engineering (MDE) approaches. The perceived benefit is that MDE enables developers to manage the complexity of software by working at a higher level of abstraction and offers the promise of automatic code generation. Indeed, domains such as automotive have actively been using MDE for the past 10 years, where some companies have developed in-house code generators and others use off-the-shelf code generators. Despite the inroads that MDE has made in industry, a recurring complaint and obstacle for industrial organizations considering MDE is the lack of sufficient tool support. When we delve deeper into this obstacle, it becomes apparent that the criticisms are often centered around poor User eXperience (UX). This vision paper explores how and where UX issues arise in the spectrum of MDE development activities, identifies the corresponding research challenges, and proposes future research directions to address them.

MDE claims several advantages over other approaches to software development, including abstract representations of complex system functionality, complementary views of a given system (e.g., behavioral versus structural), and vertical refinement of high-level system requirements models into design models and eventually down to (automatically-generated) executable code. Much MDE research has focused on individual elements of these capabilities (e.g., new modeling language to represent behavior with timing constraints, model-based testing technique, and semantic mappings for a given modeling language). A notable challenge to a broader use of MDE is the disparate nature of MDE languages, supporting techniques and tools that puts the onus on the MDE developer to identify and "stitch" together the necessary technology and techniques to provide a uniform, cohesive, and seamless integrated experience when progressing from concept to deployed system in a MDE-driven approach.

This paper introduces the concept of User eXperience for MDE, termed *MX* to highlight the challenges and opportunities surrounding UX for MDE-based development. While work has been done over the past decade in UX for software engineering, we posit that MX introduces new dimensions of user experience that are unique to MDE-based development which require complementary investigations. Specifically, at the core of the challenges is the multi-dimensional notion

of integration: model integration, MDE-based tool integration, and integration between the MDE-based process and development environment. These three dimensions of integration, collectively, must be addressed in order for the broader community to adopt MDE-based development on a larger scale. A foundational aspect of MDE is the reliance on the use of multiple models that are used to describe a given system. These models vary according to level of abstraction (e.g., requirements model down to detailed design model) and also viewpoint (e.g., structural versus behavioral models). As such, the first challenge is model integration, where vertical and horizontal model integration (syntactic and semantic) is paramount in order to ensure consistency. Vertical integration relies heavily on traceability from one level of abstraction to the next, while horizontal integration requires maintaining consistency amongst the different views, especially during vertical refinement activities. Second, MDE tool support usually comes from disparate sources and provides targeted capabilities (e.g., model analysis, code generation, test case generation, etc.). It is up to a MDE developer to identify the techniques and corresponding tools to perform the necessary tasks. Finally, a challenge to an MDE developer is to find a single Integrated Development Environment (IDE) for MDE-based development that seamlessly supports progression from concept (possibly in natural language format) to code.

The challenges and future directions for research are presented in the context of two use cases that capture state of the practice scenarios with MDE-based development. The remainder of this paper is organized as follows. Section II gives an overview of the origins of UX and defines applicable terminology. Section III discusses UX in the context of software engineering. Section IV gives the state of the practice of UX in MDE. Section V presents the use cases and identifies key challenges for the area of MX. Finally, Section VI identifies promising future research directions to tackle these challenges, as well as calls out communities that should be engaged to collaboratively work on this increasingly important area.

II. History of UX

The area of UX originated in the context of usability of software systems, but the scope of UX has grown beyond a simple notion of usability [26]. Even the term usability has several definitions. In Human-Computer Interaction (HCI), the most widely accepted definition is the one proposed in the ISO/IEC 9241-11 [22]: *"the extent to which a product can be used by specified users to achieve specific goals with effectiveness, efficiency and satisfaction in a specified context of use"*. The intention was to emphasize that usability is an outcome of interaction rather than a property of a product. This standard has been replaced by the ISO/IEC 9241-210 [23], which extends the scope of software products to also consider systems and services.

In Software Engineering (SE), the ISO/IEC 25010[1] [20] recognizes that usability plays a dual role: it is a property of a product affecting its internal and external quality and the outcome of user interaction affecting the product's quality in use. This standard defines software product and computer system quality from two complementary perspectives:

- A product quality model composed of eight characteristics (including usability), which are further subdivided into sub-characteristics (e.g., learnability, user error protection), that relate to static properties of software and dynamic properties of the computer system. Usability is defined as *"the capability of the software product to be understood, learned, used, and attractive to the user, when used under specific conditions"*. This means that it can be measured as "conformance to specification", where usability is defined as a matter of products whose measurable characteristics satisfy a predefined fixed specification. The objective is to predict usability problems that users would experience if they were interacting with the software product.
- A quality in use model composed of five characteristics relating to the outcome of interaction when a product is used in a particular context. This standard defines quality in use as the *"degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use"*.

These different definitions of usability directly affect how it is evaluated, since each method or technique employed in these evaluations may focus on different aspects of the term usability (e.g., learnability of the user interface). The consequence is that there are many individual methods for evaluating usability; they are not well integrated into a single conceptual framework that facilitate their usage by developers who are not trained in the field of HCI.

Usability was operationalized mainly in terms of the user performance (effectiveness and efficiency) given the many problems that were experienced by users of commercial systems. However, as the use of complex systems became widespread, there was an increasing awareness of the importance of the user's subjective reactions and emotional experience. This has led to the focus on User eXperience (UX).

UX is a maturing research area that goes beyond traditional usability. It provides a richer scope where user emotions, affects, motivations, and values are given as much, if not more, attention than ease of use, ease of learning and basic subjective satisfaction [25]. In contrast to task-oriented interactions, UX represents a shift to "experience", focusing also on hedonic qualities[2], and positive emotions and affect (e.g., interested, enthusiastic, irritable) that people experience while interacting with software products or systems.

Despite its importance, there is a lack of agreement on the scope of UX. A formal definition of UX issued by ISO

[1]This standard is a revision of the ISO/IEC 9126-1 [21]. Other parts of ISO/IEC 9126 define metrics for usability and quality in use.

[2]Hedonic quality refers to a product's ability to provide stimulation (e.g., novelty and challenge) and identification (users can express themselves through the product)

9241-210 [23]: *"A person's perceptions and responses that result from the use and/or anticipated use of a product, system or service"* is ambiguous and needs to be refined [24]. The work of Law et al. [25] surveyed the views of 275 researchers and practitioners working on user experience. They found that different definitions of UX exist, but that they share some key characteristics: UX is inherently subjective, context-dependent, dynamic and individual (meaning that each experience is unique) and it is concerned with positive or valuable experiences.

Evaluation of UX not only depends on the various constructs and factors that contribute to the overall experience, but can be heavily influenced by the type of product, the various development phases of the product, the interaction technique, and contextual factors. UX has been evaluated using a variety of methods [36] and metrics [3] .

III. UX in Software Engineering

The SE community has recognized the importance of usability. Efforts have focused on explaining the implications of usability for requirements gathering, software architecture design, software design, and the selection of software components (e.g., [2], [7]). Several research efforts have considered integrating usability and user-centered design in specific software development approaches such as agile (e.g., [10]) and MDE (e.g., [14], [4]). While the gap between HCI and SE with regard to usability has somewhat been narrowed, it may be widened again due to the emergence of UX. Usability evaluation methods and metrics are relatively more mature. In contrast, UX design and evaluation methods are still taking shape. It is conceivable that feeding outcomes of UX evaluation back to the software development cycle to instigate the required changes can be even more challenging than doing so for usability evaluation. We observe that most of the initiatives that deal with the integration of UX in software development are focused on agile methods. Systematic reviews identified recurring themes and patterns of the most common activities and artifacts used by teams when integrating UX in agile processes [13] and some maturity models for guiding companies with this integration start to appear [29], [31].

Outside of the MDE community, there is a long history of considering the role of the user – and how the user will interact – in tool development. Indeed, some argue that even Heidegger recognized this, making a distinction between tools (of the non-software kind) that were 'ready to hand' versus 'present at hand' [8]. Heidigger's observation was that tools should be so intuitive that they fade into the background for the user – i.e., be ready to hand. In this case, the tool is completely natural to use (the classic example is that of a hammer). In contrast, many tools (especially in software development) – those which are present at hand – are instead difficult to use to the point that the user has to focus on the tool rather than the problem s/he is trying to solve.

There is a distinguished body of work looking at how best to take users' existing practices into account when designing software tools. These include consideration of the social and organisational context of a new tool [17], designing tools where there is a cognitive fit between users' mental models and tool representations [16], acknowledging that users have cognitive biases that will affect tool adoption [30] and being aware that business considerations usually out-trump technical elegance of a tool [9]. The fields of participatory design and co-design [32] have tackled these challenges head-on by including users and other stakeholders as equal partners from the very beginning of the design process. By contrast, in the MDE community, tools are still typically designed and implemented by technicians and users are asked for feedback only once the tool is built, if at all. This has led to re-design efforts that are very costly. A concrete example that illustrates this is the case of the MetaSketch tool. As described in [19], the original version of the tool allows software language engineers to define metamodel languages and supports other users (e.g., designers, developers, and other practitioners) in creating models using a similar interface for metamodel definition and modeling. This design does not comprehensively support users of different levels of expertise and varied work styles: whatever kinds of models or metamodels are used, the interface is the same. The tool was redesigned and separated in many parts to support different uses and to engage all potential user types.

IV. UX in MDE: State of the practice

In this section, we give a brief overview of the state of industrial practice in MDE, and use this to draw out some typical pain-points for users of MDE with respect to UX.

As a result of discussions with numerous stakeholders and collaborators in industry, we have developed two generic MDE scenarios, which we will use to help distill some of the common tasks carried out when using MDE, and which will in turn be used to help derive MX challenges.

The first is a software engineering scenario for producing aerospace control software. The scenario requires collaborative modeling capabilities to support a team of software engineers. The use of MDE starts after a significant number of requirements (safety, functional, non-functional) and constraints have been derived or imposed by the system engineering process or the embedding system. These requirements are usually captured in a non-MDE tool such as DOORS, Excel or Word. The requirements (and previous experience) are used to drive customisation of UML for the particular project at hand; the customisation is done via UML profiles, supported by a suitable profile-aware UML tool such as Papyrus. Then, iteratively, models and constraints on models are produced; in some cases, the constraints are supplemented with "fixes" that will run update transformations on the models when a constraint is violated. Finally, a code generator is applied to produce Ada code from the models. In rare but not totally unknown cases, the code generator must be extended to support new customisations or new domain constraints, e.g., forms of timing analysis.

The second scenario is from the automotive domain, where the most common modeling languages used are UML and SysML. Rhapsody is sometimes used as a tool to create

UML/SysML models to represent the structure of the software architecture at a high level as well as to describe the behaviour of the system via state machines. While Infotainment subsystems rely heavily on UML, other subsystems, such as braking, powertrain, door and window control, which involve lots of I/O and control design, rely more on Stateflow/Simulink models. In some cases, DOORS is used as a tool for capturing early requirements, and then a so-called "bridge" (a model transformation) is built in-house to go from DOORS to Rhapsody for example. Most of the specified models are used for code generation. However, some UML/SysML models are also used to create models in Simulink. SysML in particular is used for analysis. Stateflow/Simulink models are used both for analysis (e.g., throughput analysis) and code generation.

From these scenarios we identify some common tasks that engineers carry out when applying MDE:

- editing models (collaboratively, individually)
- customising modeling languages (via profiles, via annotations, via creating a DSL)
- analysing models (verification, validation)
- managing models (transformation, merging, comparison)

The use of standard modeling languages (and customizations of modeling languages) is a cross-cutting concern.

Our observations are consistent with those from a key reference in the field [27], which surveys four large industrial MDE projects carried out between 2006-2010. While the study did not focus on UX concerns specifically, it did elicit the following set of key MDE tasks carried out by the engineers:

- *model engineering*, i.e., constructing models, building domain-specific languages, carrying out model management - e.g., model transformation);
- *verification and validation*, i.e., testing models, carrying out performance analysis, model simulation;
- *run-time configuration and management* of systems using models, once systems have been deployed (so-called *models at runtime*);
- *modeling platforms*, i.e., common architectures, standards and repositories that underpin the three tasks above.

From this study, and from our experiences working with industry on the scenarios described above, several pain-points pertaining to UX have emerged, including:

- *learning:* learning to use MDE was determined to be neither easy nor hard, but significant engineer time needed to be put into learning MDE basics (e.g., editing models).
- *training:* training personnel, and finding trained personnel, is difficult. In the aerospace scenario described above, which was part of a large collaborative project, trained personnel were concentrated in one or two organisations (out of around 15 partners).
- *lack of domain expert focus:* MDE tools were largely oriented to engineers, rather than considering diverse types of users (e.g., domain experts);
- *complex languages:* the standard modeling languages that were used - specifically UML - were considered to be very complex.

With this in mind, we identify a set of MX challenges next.

V. UX in MDE: Challenges

From an industrial perspective, the main goals for using MDE are to increase productivity (i.e. reduce development time and cost), increase system quality, and increase overall business and technical agility. There exist both UX *and* MX challenges for achieving these goals, where we identify five of the main challenges that must be addressed by MDE languages and their corresponding tools.

A. User Model Integration

As illustrated in the scenarios, current MDE practices are focused on functionalities and tasks. In order to achieve UX, we should move MDE from a technology-driven approach to a user-driven approach. This means that in order to properly incorporate UX in MDE it is essential to identify "who the users are" (e.g., developers, customers, suppliers, end-users), and what their activities and concerns are (see [34] for a starting point). This can be challenging for MDE as there are many different potential users (e.g., domain modeler, transformation user, metamodeler/language designer). A key aspect at this stage is to establish a User Model for describing a typical user. This kind of model can be used to set general parameters (e.g., age, gender, qualifications), preferences (e.g, preferred modeling abstractions and representations, level of automation), level of expertise (e.g., novice, expert), etc. The Persona technique [12], [6] is currently a well-known approach for the modeling of users. Next, the business goals and the interests, motivations and values of the users need to be identified and understood. The software development team should then have processes for tailoring UX design to these goals, interests and motivations. Once the mechanics have been applied, ongoing UX measurement, monitoring and improvement are essential.

B. Processes for Tailoring UX

Three processes are relevant for tailoring UX for MDE: (1) modeling UX; (2) design for UX; (3) evaluation and improvement. The first activity is concerned with the need to understand, scope and define the concept of MX. For this purpose, we should understand what UX means for MDE.

- What is the *unit of analysis* for UX (social or individual)? Is it a single aspect of an individual end-user interaction with the language/tool or several aspects of multiple users' interactions (co-experience) are relevant as well?
- Many *dimensions* are involved (functionality, usability, emotion, value, pleasure, beauty, social, hedonic quality, etc.). There are studies [35] that show that some of them hold for experiences in leisure domains (e.g., gaming) and others for experiences in work domains, but what is the relevance of these dimensions for MDE?
- What are the *contextual factors* (e.g., technology, physical environment, earlier experiences, task context) that affect UX of MDE languages and tools?

The second activity is concerned with the design of UX in MDE – that is, how can we consider UX and usability within scenarios that apply MDE. In practical terms in MDE we are always using a tool of some kind to create and manipulate models (this includes digital tools as well as pen and paper). Every tool comes with an inherent load factor; for pen and paper this is small, but for Papyrus or Eclipse, it is larger. We need to be able to assess a tool's load factor on users to ensure that it is acceptable. Techniques for achieving this can be based on participatory design, focus groups, design science, etc. This focuses on the *process*; we still need better approaches to think about UX and usability of UML, Simulink, or DSLs that we create. A potential starting point would be the Physics of Notations [28] (which aspires to provide a theory for assessing and designing effective visual notations), or the Cognitive Dimensions Framework [16], which aims to consider both notation and tooling. Creating or customizing a language taking into account existing knowledge about usability and cognitive effectiveness is a significant challenge.

The third activity is concerned with the evaluation of MX. There are several specific challenges related to this evaluation:

- When should UX be evaluated? During the user interaction, before interaction or after interaction? The three moments are relevant - industry is typically interested in long-term user experiences, as temporary feelings are less important than the overall product user experience when people evaluate products.
- How can we evaluate UX in early stages of modeling language or tool development? For developers, it is essential to evaluate UX already in the early stages of the language or tool development, so methods for evaluating UX of anticipated use without the actual language or tool will be very valuable.
- How can we operationalize and evaluate UX in MDE against measurements, e.g., measurements pertaining to cognitive effectiveness?

C. Empirical Studies of MX

Although there are some empirical studies that deal with users of and usability in MDE (e.g., [1], [19], [4], [34]), more studies are needed to build a body of knowledge about MX. The absence of empirical research hampers theoretical advancement on the understanding of MX. Empirical studies can be used, for instance, to evaluate the suitability of existing UX models proposed by the HCI community (e.g., [18]) in the MDE context. Empirical studies are also important to understand how modelers actually work. In particular, it is important to carry out studies to understand positive and negative experiences with modeling languages, tools and processes. Positive experiences are appropriate as understanding their determinants and underlying mechanisms can help design products that elicit these experiences, while negative experiences may inform designers and tool vendors about potential pitfalls in their product's UX. We believe that qualitative and quantitative studies of modeling work practices can help us understand how to create modeling languages and tools that better support specific user needs and work styles.

D. Customization and Domain Specific Modeling Support

A general-purpose tool is never really fit for purpose; one size does not fit all. To increase productivity, it is essential that MDE tools are customized for the specifics of the domain/context in which they are used. For this reason, MDE tools need to provide first-class support for customization and domain specific modeling. This involves three aspects: support for tool simplification to allow adapting the tool environment to only provide the set of concepts and capabilities that are relevant to the users in their development context; support for workflow customization to allow adapting the tool workflows for the specifics of the development context; and support for visually representing domain concepts in a way that is meaningful to the different users, whether they are designers developing the models or people to who the models are presented. A box in which «Cat» is written may be understood as a "cat" for a software engineer, but it is still a"box" for most people. Since communication is a key aspect of complex system development, the use of specific shapes or icons to represent domain concepts is considered to be essential by many stakeholders. Modeling must make communication with the different stakeholders easier, not more difficult.

Support for customization and domain specific modeling is an aspect that essentially all commercial UML modeling tools have failed to address. UML tools typically present the whole UML language to the user and they provide very little capabilities to reduce the set of concepts and diagrams to the subset the user needs. These general purpose tools have forced people to adapt their way-of-working to the constraints imposed by the tools. Such approach makes sense from a tool vendor perspective (to avoid having to support multiple modeling environments), but it results in an overall tool environment that is much more complex then it needs to be. While most people associate this problem with UML, it is not a problem with UML itself, but with the UML tools.

While it is broadly agreed that Domain Specific Languages (DSLs) provide a much simpler and better adapted environment, the use of DSLs also comes with a set of issues. France and Rumpe [15] refer to the "DSL-Babel" challenge where the surge in new DSLs poses significant challenges relating to communication, interoperability, and training.

E. Interoperability

Interoperability has a significant impact on MX. There are many dimensions to interoperability, including: *model integration* (i.e., vertically, across different levels of abstraction; and horizontally, where we must integrate models at the same level of abstraction but from different views or by different engineers); *tool integration* (e.g., integrating an MDE tool like EMF with a non-MDE tool like DOORS, or combining different MDE tools through middleware or via shared repositories); *process integration* (e.g., combining support for model transformation and test-driven development, perhaps through an

IDE); and *integration through collaboration*, i.e., support for integration among developers. Each form of interoperability substantially impacts UX: in many cases today interoperability is not seamless and hidden from the engineer, but requires significant context switching overhead (e.g., using MDE tools with a change management tool like Jira).

VI. UX in MDE: Future Directions

In this section we identify several opportunities for future research and development in MX, organized around some of the key challenges described in the previous section.

A. Support for Model and Tool Integration

As computing-based systems increase in complexity and become more pervasive, more demands are placed on the modeling community, including the development of new DSLs, new dialects or profiles for existing languages, and new modeling concerns, such as run-time monitoring for autonomous systems [5]. With this growth in language development comes the extra burden of how to integrate models in order to maintain consistency throughout the development process and beyond (e.g., testing, run-time activities, etc.). Vertical integration refers to the refinement relationship(s) when moving from abstract to concrete representations and vice versa (in the case of code refactoring and reverse engineering). Horizontal integration refers to the integration of complementary views/perspectives of different modeling languages that together need to consistently describe the same target system. These views may be represented by general purpose languages (e.g., UML and SysML), domain-specific languages (e.g., Simulink), or a combination thereof.

In order to provide useful tool support, the languages and the integration between languages must be well-defined, including traceability as the languages evolve. Both domain-specific and general purpose languages should be amenable to automated processing (e.g., analysis, synthesis, composition, etc.) with these objectives in mind. Also, when different tools are used for modeling, e.g. the combination of a UML/SysML to describe the overall system and software architecture and MathWorks Simulink to describe the signal processing aspect, model integration also requires the integration of tools.

The GEMOC Initiative is an international group of MDE researchers are taking a multi-pronged approach comprising language engineering, tool/framework development, and processes to enable the *globalization of modeling languages* [11]. And the long-running international conference on Software Language Engineering (SLE) also includes research in this direction.[3] These examples are indicative of the MDE community's awareness of the technical challenges, but none of these are explicitly addressing MX.

B. Support for Domain Specific Modeling

Domain-specific tooling is critical for the broad adoption and success of MDE. While important progress has been made over the last years to better support DSLs, more research and development is required to reach the required level of MX.

- **Domain-specific vs general-purpose tools** .
  Domain-specific tools offer better UX then general purpose tools. However, it is important to distinguish between *language* and *tool*. In general, people tend to establish some type of equivalence between a language and the tools that are supporting it. For example, UML is often put in opposition with DSLs based on the argument that UML is too big and complex. However, the reason for this oversimplification is not UML itself, but the tools supporting it. UML is a general purpose language that can be supported by both general-purpose and domain-specific tools. As discussed in the previous section, the existing commercial UML tools have mainly focused on providing support for the complete UML language, and not on providing first-class support for customization and DSL. However, Papyrus, also based on UML, provides advanced support for customization and DSLs and can be viewed as a DSL workbench. Many companies have successfully used Papyrus to develop their own DSLs. The same reasoning applies to EMF, which is a general-purpose language for which the Eclipse Sirius DSL workbench has been developed.
  In order to support domain specific modeling, a tool must: 1) Be based on a language that provides the required mechanisms for the definition of domain specific languages, and 2) Provide tool customization capabilities that allow adapting the menus, pallets, and workflows to the specifics of the domain.
- **Domain-specific environments to support integration.**
  A main challenge industry faces is that the desire to optimize the productivity of the developers/engineers requires that the development of complex systems uses different domain specific environments/tools for different "aspects" of the system. In order to ensure overall consistency and enable system maintenance and evolution, the models produced for these different aspects must be integrated together. In this context, one way to reduce the complexity and cost of integrating different DSLs is to base different DSLs on the same underlying language (or meta-model). While it is in general not possible or practical to base all aspects on the same underlying language, it is highly desirable to consolidate as much as possible. For example, UML can be used as an underlying language to define several DSLs for different software and system aspects.

C. Support for User Model and Process Integration

- **Support for Collaborative Modeling.**
  MX can be improved significantly by supporting collaboration. Comparing modeling IDEs to programming IDEs, we see that model based tools lag in capabilities such as diff/merge and code reviews: model diff/merge is still considered unreliable or hard to use, and there is essentially no real support for model reviews.

[3] http://www.sleconf.org/

- **Moving from a technology/functionality-driven approach to a user-driven approach.**
  As model-based tool developers, we often ignore the fact that the majority of users who will be using our tools are in fact not trained as MDE engineers. Most users are trained as software engineers, and others, are domain experts who may have limited software engineering knowledge. In order to overcome this gap, and make our tools easier to adopt by non-MDE experts, we need to consider various aspects: We need to understand the context in which MDE languages and tools are used. This should at least consider the users, the tasks, and the work environment. For users, we should understand their background, domain and expertise and try to provide tools that help "bridge" the gap between what they know and what is expected for them to know. We need to understand and classify the tasks that these different types of users of MDE are trying to accomplish (i.e., perform task modelling). We also need to understand the work environment, which includes company policies, regulations, etc. Finally, we should consider an incremental evaluation of MX rather than a post facto evaluation.

D. Cross-cutting Areas to be Addressed

1) Evaluation Methods and Metrics for MX: In order to make concrete progress in this area, we need to define methods and metrics for evaluating MX with a sound theoretical basis. These could be based on methods and metrics from UX in general, but more needs to be done on adapting these approaches to MDE artifacts, concepts and tools.

2) A Theory for MX:: In mature sciences, empirical theories help gain and accumulate knowledge. If we want to understand MX, we need to build an explanatory framework with a number of factors that explain UX for MDE. The theory should contain hypotheses that can be tested empirically in order to provide evidence about why a certain phenomena occur and how we can predict it. The theory will be useful to understand how the different factors affecting MX (e.g., type of task, task context, unit of analysis, UX dimension, MDE technology, work environment, earlier experiences, etc.) relate to each other. From a more practical viewpoint, a theory for MX can give us input for decision-making regarding choices of technology and resource management. Existing theories from the SE field (e.g., theory for explaining the effect of UML-based development [33]) or other disciplines such as Behavioral Sciences and Cognitive Psychology (e.g., theory of cognitive fit) can be adapted to MDE.

3) Engaging Other Disciplines: Given the range of issues covered by MX, it is strategic for the MDE community to engage researchers and stakeholders from other disciplines to collaboratively address the numerous challenges including:

- cognitive behavioral scientists.
- graphical design and visualization technology experts.
- game designers (there is a wealth of work in educational games that enables kids to focus on the "game" and not become overwhelmed with the intricacies of a given tool.)
- domain experts to work on the respective DSLs, tools and their integration with a domain-specific process.

4) Training and Support: A significant cross-cutting MX issue is training: we currently rely heavily on weighty books on our desks, and web articles, to help train users in MDE; this offers a weaker UX than training approaches for programming that are based on StackOverflow and YouTube videos. People need to be able to quickly obtain answers to their questions and understand how to best use the tools (based on community experience and feedback). Conventional passive documentation is no longer sufficient or viable. In industry, it is common practice for users who have questions about C++ or CDT to go on StackOverflow and quickly obtain an answer (or many answers). When users have a problem with (commercial or open source) MDE tools, there is no obvious resource to consult for answers. We need a bigger and stronger user community modeled on successes like StackOverflow.

Consider also the success factors for widely used modeling tools such as those from Mathworks: they are used early and often in university engineering programs. Students learn to use the tools early in their education program and they continue to use it when they get into the work place. Also, they learned to live with (or adapt to) the limitations of the tools from the beginning. Similar experiences with MDE tools – i.e., early and often usage in university – may help mitigate the common pain points related to MX.

VII. Conclusions

The MDE research community has claimed many successes – we have developed powerful tools, and some of them have been used to solve significant industrial problems. Can we yet claim that our tools are *useful*? Given that we have not broadly considered usability during the development of our tools, and when evaluating their use in practice, it is difficult to provide supporting evidence for such claims. Such evidence is particularly important if we are aiming to provide holistic or systemic support. In a nutshell, if we are claiming that our MDE solutions are useful, we should not false advertise them.

One approach to improving MX would be to evaluate the usability of our existing MDE tools, and use that process to trigger improvements, e.g., to interfaces. Can we consider usability and UX for MDE *up front*, before the tool or metamodel has been built? What is the added value of doing so? Can we iteratively and incrementally evaluate usability while an MDE tool is being constructed, or being applied in an engineering context?

These, and the other challenges identified in Section VI, will hopefully set the stage for a program of research on UX in MDE which will benefit both tool builders and users of MDE – and the software engineering community as a whole.

VIII. Acknowledgments

Thanks to Bran Selic, Jordi Cabot and Dimitris Kolovos who provided valuable input for the "UX for MDE" panel at MODELS'16.

References

[1] S. Abrahão, E. Iborra, and J. Vanderdonckt. Usability evaluation of user interfaces generated with a model-driven architecture tool. In *"Maturing Usability: Quality in Software, Interaction and Value"*, pages 3–32. Springer Berlin Heidelberg, 2007.

[2] S. Abrahão, N. Juristo, E. L.-C. Law, and S. Jan. Interplay between usability and software development. *Journal of Systems and Software*, 83(11):2015–2018, 2010.

[3] W. Albert and T. Tullis. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics, Second Edition*. Morgan Kaufmann, USA, 2013.

[4] D. Albuquerque, B. Cafeo, A. Garcia, S. Barbosa, S. Abrahão, and A. Ribeiro. Quantifying usability of domain-specific languages: An empirical study on software maintenance. *Journal of Systems and Software*, 101:245–259, 2015.

[5] N. Bencomo, R. B. France, B. H. C. Cheng, and U. Aßmann, editors. *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, November 27 - December 2, 2011]*, volume 8378 of *Lecture Notes in Computer Science*. Springer, 2014.

[6] Buxton, Bill and Greenberg, Saul and Carpendale, Sheelagh and Marquardt, Nicolai. *Sketching User Experiences: The Workbook*. Morgan Kaufmann, 2012.

[7] L. Carvajal, A. M. Moreno, M. I. S. Segura, and A. Seffah. Quantifying usability of domain-specific languages: An empirical study on software maintenance. *IEEE Trans. Software Eng.*, 39(11):1582–1596, 2013.

[8] M. Chalmers. A historical view of context. *Computer Supported Cooperative Work*, 13(3):223–247, 2004.

[9] T. Clark and P. Muller. Exploiting model driven technology: a tale of two startups. *Software and System Modeling*, 11(4):481–493, 2012.

[10] G. Cockton, M. Larusdottir, P. Gregory, and A. Cajander, editors. *Integrating User-Centred Design in Agile Development*. Springer, 2016.

[11] B. Combemale, B. H. Cheng, R. B. France, J.-M. Jezequel, and B. Rumpe. *Globalizing Domain-Specific Languages*, volume 9400 of *LNCS, Programming and Software Engineering*. Springer International Publishing, 2015.

[12] Cooper, Alan. *The inmates are running the asylum*. SAMS, 1999.

[13] T. S. da Silva, A. Martin, F. Maurer, and M. Silveira. User-centered design and agile methods: A systematic review. In *Proceedings of the 2011 Agile Conference, Salt Lake City, USA*, pages 77–86. IEEE Computer Society, 2011.

[14] A. Fernandez, S. Abrahão, E. Insfrán, and M. Matera. Usability Inspection in Model-Driven Web Development: Empirical Validation in WebML. In *"16th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2013), Miami, USA"*, pages 740–756. Springer Berlin Heidelberg, 2013.

[15] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[16] T. Green and M. Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7(2):131 – 174, 1996.

[17] J. Grudin. Why CSCW Applications Fail: Problems in the Design and Evaluationof Organizational Interfaces. In *Proceedings of the 1988 ACM Conference on Computer-supported Cooperative Work*, CSCW '88, pages 85–93, New York, NY, USA, 1988. ACM.

[18] M. Hassenzahl, S. Diefenbach, and G. Anja. Needs, affect, and interactive products – facets of user experience. *Interacting with Computers*, 22(5):353–362, 2010.

[19] K.-H. Huangand, N. J. Nunes, L. Nobrega, L. Constantine, and M. Chen. Hammering models: designing usable modeling tools. In *INTERACT'11 Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part III, Lisbon Portugal*, pages 537–554. Springer-Verlag, Berlin Heidelberg, 2011.

[20] ISO 25010:2011. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. ISO, Geneva, Switzerland.

[21] ISO 9126-1:2001. *Software engineering - Product quality - Part 1: Quality model*. ISO, Geneva, Switzerland.

[22] ISO 9241-11:1998. *Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability*. ISO, Geneva, Switzerland.

[23] ISO 9241-210:2010. *Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems*. ISO, Geneva, Switzerland.

[24] E. Law and S. Abrahão. Interplay between user experience (ux) evaluation and system development. *International Journal of Human-Computer Studies*, 72(6):523–525, 2014.

[25] E. L.-C. Law, V. Roto, M. Hassenzahl, A. P. Vermeeren, and K. Joke. Understanding, scoping and defining user experience: A survey approach. In *ACM SIGCHI conference on Human Factors in Computing Systems*, pages 719–728. ACM, 2009.

[26] H. Marc and T. Tractinsky. User experience - a research agenda. *Behaviour and Information Technology*, 25(2):91–97, 2006.

[27] P. Mohagheghi, W. Gilani, A. Stefanescu, and M. A. Fernandez. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering*, 18(1):89–116, 2013.

[28] D. L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.

[29] A. L. Peres, T. S. Da Silva, F. S. Silva, F. F. Soares, C. R. M. De Carvalho, and M. S. R. D. Lemos. Agileux model: Towards a reference model on integrating ux in developing software using agile methodologies. In *Proceedings of the 2014 Agile Conference, Orlando, USA*, pages 61–63. IEEE Computer Society, 2014.

[30] P. Ralph. *Toward a Theory of Debiasing Software Development*, pages 92–105. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[31] D. Salah, R. F. Paige, and C. P. A. A maturity model for integrating agile processes and user centred design. In *16th International Conference on Software Process Improvement and Capability Determination (SPICE 2016), Dublin, Ireland*, pages 109–122. Springer, 2014.

[32] E. B.-N. Sanders and P. J. Stappers. Co-creation and the new landscapes of design. *CoDesign*, 4(1):5–18, 2008.

[33] D. I. K. Sjøberg, T. Dybå, B. C. D. Anda, and J. E. Hannay. *Building Theories in Software Engineering*, pages 312–336. Springer London, London, 2008.

[34] H. Störrle. How are Conceptual Models used in Industrial Software Development? A Descriptive Survey. In E. Mendes, K. Petersen, and S. Counsell, editors, *Proc. 21st Intl. Conf. on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 2017.

[35] A. N. Tuch, P. Van Schaik, and K. Hornbæk. Leisure and work, good and bad: The role of activity domain and valence in modeling user experience. *ACM Trans. on Computer-Human Interaction*, 23(6):35:1–35:32, 2016.

[36] A. P. Vermeeren, E. L.-C. Law, V. Roto, M. Obrist, J. Hoonhout, and V.-V.-M. Kaisa. User experience evaluation methods: Current state and development needs. In *6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, Reykjavik, Iceland*, pages 521–530. ACM, 2010.