An analysis of user-interface creation complexity using a model-driven design approach

Análisis de la complejidad de la creación de interfaces de usuario utilizando un enfoque de diseño basado en modelos

Victor H. Mercado[1], Yony F. Ceballos[2], German Sanchez-Torres.[3]

[1]*MS.C. en ingeniería de software, Profesor Auxiliar, Universidad de Antioquia, Grupo de Investigación Ingeniería y Tecnologías de las Organizaciones y de la Sociedad, Medellín-Colombia*

[2]*Ph.D. en ingeniería, Profesor Asociado, Universidad de Antioquia, Grupo de Investigación, Ingeniería y Sociedad, Medellín-Colombia*

[3]*Ph.D. en ingeniería, Profesor Asociado, Universidad del Magdalena, Grupo de investigación en Sistemas y Computación*

Email: yony.ceballos@udea.edu.co

Cite this article as: *V. Mercado, Y. Ceballos, G. Sanchez-Torres "An analysis of user-interface creation complexity using a model-driven design approach", Prospectiva, Vol 17, N° 2, 40-46, 2019*

Recibido: 09/07/2019 / Aceptado: 31/07/2019

http://dx.doi.org/10.15665/rp.v17i2.2078

ABSTRACT

Model-Driven Development (MDD) is a software development approach that facilitates problem comprehension. MDD is carried out based on level of abstraction attained by working with models. When using models to create user-interfaces, development time can be reduced significantly. However, automatic user-interface generation is done with preestablished templates that might not fulfill all the requirements clients. These templates might also be too general and sometimes provide few customization options. In this paper, we review research on the usage of MDD for user-interface development. We also study how automatic user-interface generation can result in limitations as the need might arise for repeatedly modifying model and code.

Keywords: *Model-Driven Development; User-Interface Creation; Software Development; User-Interface Development: Templates.*

RESUMEN

El Desarrollo Dirigido por Modelos (o MDD, por sus siglas en inglés) es un enfoque de desarrollo de software que facilita la comprensión de problemas. El MDD se lleva a cabo según el nivel de abstracción alcanzado al trabajar con modelos. Cuando se usan modelos para crear interfaces de usuario, el tiempo empleado se puede reducir significativamente. Sin embargo, la generación automática de la interfaz de usuario se realiza con plantillas preestablecidas que pueden no ajustarse a todos los requisitos del cliente. Comúnmente, estas plantillas pueden ser demasiado generales y, en ocasiones, ofrecer pocas opciones de personalización. En este documento, se revisa el uso de MDD para el desarrollo de la interfaz de usuario. También se estudia cómo la generación automática de interfaz de usuario puede presentar limitaciones, ya que puede ser necesario modificar el modelo y el código repetidamente.

Palabras clave: *Desarrollo Dirigido por Modelo (MDD); Creación de interfaz de usuario; Desarrollo de software; Desarrollo de interfaz de usuario: Plantillas.*

1. INTRODUCTION

The main goal of software development companies is to fulfill client expectations in less time. However, when it comes to the life-cycle of a software project, expectations can change based on factors such as the accomplishment of the business idea, the reassessment of specific milestones, and the lack of clarity in the scope of the objectives which are set during the initial phase of the project [1]. It is also usual for development teams to employ abstraction models in order to better understand the problems at hand [2].

Model-Driven Development (MDD), also known as Model-Driven Software Development (MDSD), is a software development approach for creating and transforming models based on abstraction, automation, and standardization [3]. These models are turned into running applications through a variety of tools [4]. The main advantages of this methodology are better problem comprehension (thanks to the higher abstraction degree) and faster interface development. However, when these models are transformed into running applications, the resulting interfaces often clash with the corporate image of the client and might not satisfy all of the requirements due to the limitations of the model transformation tool [5].

In order to solve these issues, it is necessary to carry out modifications to the model by altering application code, which implies a need for the developers to have specific knowledge related to how to transform models [6]. In turn, these transformations can be complicated to implement due to difficulties related to properly understanding the code generated by the model transformation tool. Furthermore, if the interface is modified manually due to some client request, and it is necessary to generate the interface again, the manual changes will be lost. For this reason, every change made to the original model must be traceable [5].

Most of the work in the literature singles out limited support for existing tools as the main reason for the MDD approach not being widely adopted in the industry [7]–[9]. However, that is not the only influential factor. A taxonomy of these factors is described in [10], including:

- Technical factors
- Internal organizational factors
- Factors external to the organizations
- Social factors

In general, technical factors include weaknesses in model transformation tools, weaknesses in the support for domain-specific languages, limitations in code generation, limitations in tool applicability, limitations in complexity, and user-related considerations such as usability and abstraction levels.

Internal organizational factors comprise considerations related to either adapting the MDD paradigm to the existing organizational process or adapting the organizational process to the said paradigm, while considering issues such as sustainability, migration and integration issues, among others. Also included in this category are factors such as organizational culture and whether there are members in the organization with the skills required for adopting this approach.

The impact of governmental standardization, commercial strategies associated with tool adoption costs, and the relationship of said strategies with the business model of the organization all constitute factors external to the organizations.

Finally, the social perception of the community about the quality of the generated code belongs to the social factors category.

In order to carry out the literature review, we analyzed several perspectives on model transformation, and we determined that our main interest for this work was model-based user interface generation.

The lack of flexibility of tools for model-based user interface generation has been addressed in the literature [11], [12]. Main weaknesses of these models include the high degree of similarity among automatically-generated interfaces and the reduced amount of configuration options, which make it harder to fulfill all the project requirements properly.

In this paper, we analyze the strategies that have been proposed to make model transformation tools more flexible, to avoid similarity between automatically-generated user interfaces, and to better satisfy project requirements. Similarly, we are also interested in describing strategies for guaranteeing the traceability of manual changes made to the interfaces after they are generated with a given tool.

2. MATERIALS AND METHODS

Systematic literature reviews (SLRs) are often employed to validate statements about the trends and behavior of a scientific community. Through SLRs, researchers can aggregate experiences resulting from various studies in order to answer a specific research question [13]. Such reviews are based on methodologies designed for determining the final set of work which will be employed to carry out the corresponding study, while guaranteeing objectivity and reproducibility.

Thus, SLRs identify, evaluate, and allow the synthesization of all of the relevant work in order to assess a given research area or scientific phenomenon [14], [15].

The general methodology used in this work is described on the Figure 1.

A. Research questions

Research questions guide the literature review as they provide basic criteria for selecting the main studies and define the relevant pieces of information which are to be extracted, as well as how to synthesize them in order to answer the questions [16]. In this work, we analyzed the following research questions:

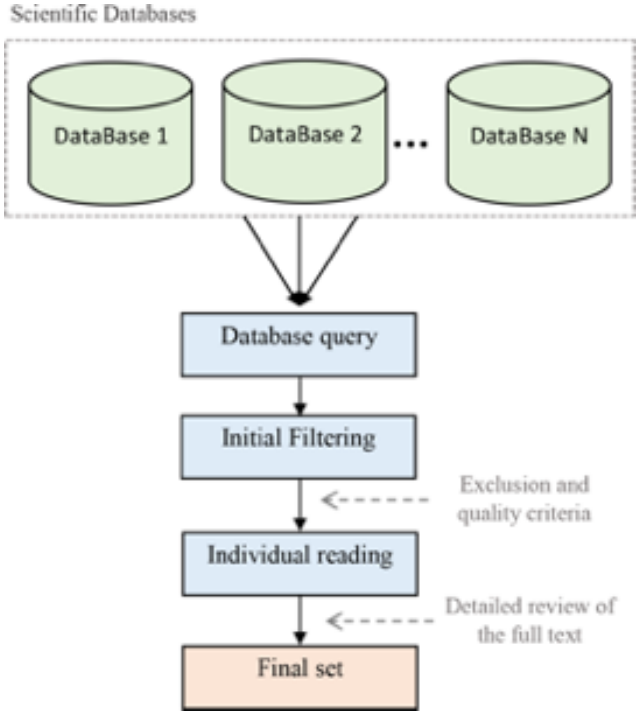
- How can we make the transformation model more flexible in order to prevent the automatically-generated interfaces from being too similar to each other, while fulfilling all project requirements?
- How can we track manual changes made to the models, with the purpose of fulfilling client needs?

B. Exclusion criteria

Exclusion criteria are intended to guarantee that non-relevant work is excluded from the study. For this reason, we adopted the following exclusion criteria:

- EC1: Duplicated or similar work among the platforms.
- EC2: Full-text unavailable.
- EC3: The documents are not related to architecture-based tests.
- EC4: The work does not explicitly discuss the MDD or MDSD approaches.
- EC5: Experiences or polls are reported.
- EC6: No work is carried out on user-interface design.

Figure 1. Methodology used for literature review.



C. Quality criteria

The quality of the papers can be evaluated based on the following criteria:

- QC1: The reported solutions for addressing the issue are built coherently, on solid knowledge bases.
- QC2: The examples used to describe the issue are clear and appropriate.
- QC3: The reported solution can be applied regardless of the technological platform where it is implemented.
- EC4: The problem that leads to the need for developing user-interfaces through model-based automatic code generation is properly described.
- EC5: The reported solution has been tested and verified, excluding papers with partial results.

The scale employed to evaluate the papers is described in Table 1.

Table 1. Evaluation scale for criteria fulfillment

Description	Value
The criterion is completely fulfilled	3
The criterion is partially fulfilled	2
The criterion is not fulfilled	1

3. RESULTS

A. Database query

We queried several scientific databases to obtain the academic work reported in each of them. Specifically, we performed queries on Scopus, Web of Science – WoS, and GoogleScholar.

The search equation includes the terms "model-driven development" and "model-driven software development", as follows:

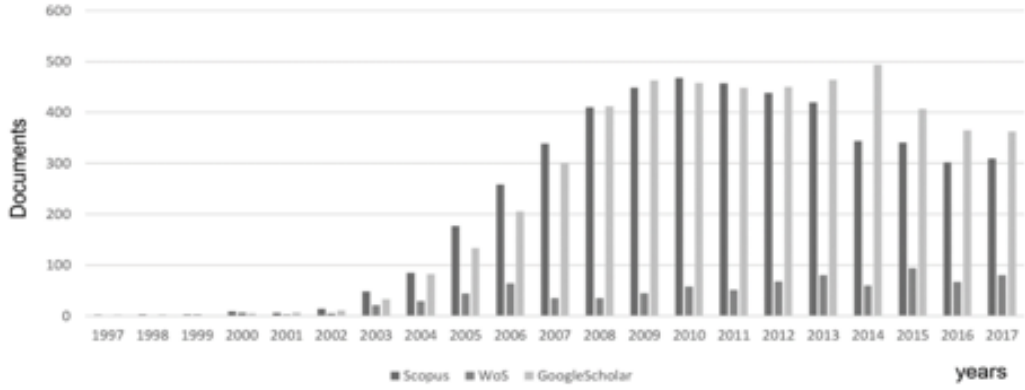
- Theme: (Model-driven development OR model-driven software development)
- Indexes: SCI-EXPANDED, SSCI, A&HCI, ESCI
- Time range: Every year.

The results, sorted by year for every database, are shown in Figure 2.

The results show that in the last two decades, academic interest for research on MDD and MDSD reached its peak towards the end of the 2000-2010 decade (see Figure 2). In the Scopus database, most of the work is comprised of conference proceedings, with 68.8%, while journal articles represent 22.9% of the reported work (see Figure 3).

By unifying the results, we produce a first set of work related to our research questions.

Figure 2. Work reported by year for MDD and MDSD in the Scopus, WoS, and GoogleScholar scientific databases.



B. Initial filtering

We removed duplicated documents by verifying the titles and authors of every paper obtained from the databases. Deduplication resulted in a set 850 unique papers, and in this phase, we applied exclusion criteria on the set of documents obtained from the databases. The first filter was to search for documents whose title and abstract addressed the issues we detected regarding model transformation in MDD and MDSD. Next, we formulated the research question, which helped us further reduce the amount of documents, since it is a very specific problem which has not been addressed by many authors. The selected articles were those whose main idea was to generate user-interfaces through MDD and MDSD.

Also selected were those that highlighted the importance of tracing changes made to models. In both cases, we prioritized articles in which the authors also proposed concrete solutions to the issues they highlighted. We excluded papers that addressed user-interface creation superficially, or that did not address it from the perspective proposed in the research question. We also excluded work in which the issue was left as an open question and no specific solution was proposed.

C. Individual reading

We obtained a final set of 6 documents related to the problem of MDD and MDSD-based user-interface construction. We did a full revision of each of the six documents which fulfilled exclusion criteria. The final set is reported in Table 2.

D. Final set analysis

The results of applying quality criteria are reported in Table 3.

Aquino [11] proposes the generation of user-interfaces based on templates, thus offering alternatives for solving the issue of similarity between automatically-generated interfaces and providing the option to add different functionalities to each interface. For this purpose, Aquino introduces the concept of transformation profile. A transformation profile has two components. The first is the transformation template, a set of parameters in which the graphical and stylistic elements of the interface are established. The second component is the correspondence model, which comprises the relationship between models, from the base model to the target model, which includes the customized elements employed to add functionality to the interfaces.

Figure 3. Distribution by type-of-product of the reported works using the terms MDD – model-drive development and MDSD - Model-Driven Software Development.

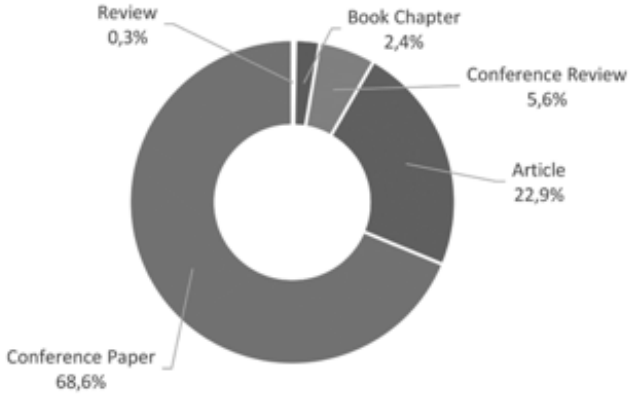


Table 2. Information related to the final set of works

ID	Title	Type	Source
[11]	Adding Flexibility in the Model-Driven Engineering of User Interfaces	Conference	ACM
[12]	Feature-Oriented refinement of Models, Metamodels and Model Transformations	Conference	ACM
[17]	Maintaining Invariant Traceability through Bidirectional Transformations	Conference	ACM
[18]	Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges	Article	ACM
[19]	Diseño de reglas de adaptación y transformación para interfaces de usuario	Article	Redalyc
[5]	Generative Pattern-Based Design of User Interfaces	Article	ACM

Trujillo et al. [12] note that, in order to properly track modifications in model transformations, it is necessary to account for three basic concepts: models, metamodels, and model transformations. Models are the representation of the problem which is to be solved. Metamodels are descriptions of the features of the model. Model transformations are processes whose input is a model and whose output is the model with the required modifications for customizing the solution and its functionality. One of the proposed tools for model transformation is MOFScript (which can be downloaded as an Eclipse ® plugin) [20]. Transformations carried out using such tools are based on rules. The term model refinement is also included, which allows for a model to be extended by adding elements, in order to facilitate customization for different purposes. This feature allows base models to be constructed, which are generalizations that possess common elements and can be used later for obtaining modified/expanded models.

Table 3. Quality criteria for papers in the final set

		Quality Criteria				
		QC1	QC2	QC3	QC4	QC5
Literature	[11]	3	3	3	3	2
	[12]	3	3	3	2	2
	[17]	3	3	1	2	3
	[18]	3	3	3	3	3
	[19]	3	2	3	3	3
	[5]	3	2	1	2	3

Yu et al. [17] address the issue of how to modify a model so that it is adjusted to specific needs. However, the authors center on the importance of having traceability every time such modifications are made. The developers can make such changes in parallel both for the code and for the model. In order to guarantee traceability in the changes, every time that a change is made in code, it should be made in the model too. Likewise, when changes are made in the model, such changes should also be observed in the code. These transformations are deemed bidirectional transformations. However, it is possible that there is not 100% correspondence between the changes made to the model or the code. The authors also show the consequences of having no correspondence between the generated templates and the modifications carried out by the user. In response, a two-layer-framework is proposed, which combines changes through bidirectional transformation. The first layer, called model–code, synchronizes structural changes between the model and the template using EMF (Eclipse Modeling Framework) [21], while the second layer, code – code, synchronizes functionality changes between the code of the template and the code of the model. The framework raises warnings when there are significant differences between code and model. The first step for creating said models is automatic code generation. The developer then modifies the code in order to customize and add the required functionalities. Such changes modify the model, so the original template must be re-generated.

Vanderdonckt [18] focuses on MDD-based user-interface generation and how such interfaces can be adapted to the changes when the user requires it. This paper describes MDD concepts in a concise way. It also describes the different levels which compose MDD. The first level is the task and domain of the model, in which the final user task is specified, setting the basis for the model. The second level is the abstract user-interface, in which the interfaces are maintained independent of the employed technological platform. In the third level, we can find the concrete user-interfaces, in which the technological platform is already defined. Lastly, there is the final user-interface, in which the interface is produced based on the previous levels. Furthermore, the concept of the model is defined, which is different than the way it is done in ID2, as the author focuses on the different kinds of models that can be used for developing an interface.

López-Jaquero and Montero [19] propose a different approach regarding the role MDD can play in interface design. This is because, in addition to the previously mentioned issues, they note that interfaces are sometimes made for users with limited technical skills, resulting in a need for them to be intuitive. There is, too, another factor: the variety of devices used to access software applications (tablets, smartphones, etc). When working with such models, it is necessary to begin with an abstract model; however, as development advances, said model is subject to several transformations, turning it into a concrete model and then into the final version. The transformations that the abstract model is subject to must follow a set of rules, so López-Jaquero and Montero define the concept of "adaptation rules". These rules are comprised of a context in which they are valid, the data which will be processed by the rules, and the transformation resulting from applying the rules. Finally, the authors introduce T:XML as a rule which allows the specification of adaptation rules.

Vanderdonckt and Simarro [5] present a new method for user-interface development based on generative patterns. This method is comprised of 4 axes that are based on pattern administration and the use of a design pattern called Markup Language (PLML), which was born in the CHI'2003 [22]. This pattern came up from the analysis carried out on existing interface patterns. The authors attempt to solve the problems that these patterns encountered when carrying out model transformation, such as interface ambiguity and incoheren-

ce. Aware of these issues, the authors propose a novel pattern in which these problems are reduced by establishing rules for the usage of the new pattern. An important axis which makes up this method is an application called IDEALXML, a software developed in Java for supporting pattern usage.

4. DISCUSSION

It is essential to model software systems when attempting to create software platforms for complex systems. Doing so allows the analyst to properly capture the most relevant aspects of the system and to take them to the required level of abstraction.

MDD attempts to improve productivity using automation of repetitive tasks. For this reason, Aquino [11] presented tools which allowed developers to quickly generate user-interface. However, those interfaces might end up looking too similar or may not completely satisfy Project needs, so Aquino introduced the transformation profile concept, but the development of the model gets too technical and the conceptual part is relegated excessively. In the end, Aquino declares the main feature of his method to be the use of templates in order to separate some special features and perform transformations. This, according to the author, constitutes an advantage when compared to other studies on MDD.

In the work of Trujillo et al. [12], the content is focused on the importance of tracking changes made on the model, while the issue of generating interfaces is not addressed. Despite this, we consider the article to be relevant, as it can be presented as a complement to the other papers we selected, due to the fact that in only a few of them the authors specify what happens when it is necessary to step back from a previous model. It is a concise proposal which involves the reader through an example that evolves with the development of the idea in order to introduce all the relevant concepts.

Yu et al. [17] highlight the importance of parallelly modifying both the automatically-generated code and the corresponding models. This is done to make interfaces more flexible and to facilitate any required customization. The authors focus on how, when building some features, it may be necessary to backtrack into a previous version of the model. For this reason, it is necessary to trace changes, because templates and code can diverge too much from the original model. In contrast with [12], the authors point out that the need for tracing changes arises from the parallel modifications made to both the code and the model. This kind of change can have serious consequences if the designed interface is too complex, which can lead to questioning its utility, because in other proposals it is not necessary to modify the code in such a direct and specific way.

In the work of Vanderdonckt [18], the proposed approach can seem general at first as the author introduces many concepts and explains them in a simple manner. There is no focus on a single tool or method. However, the author presents a wide array of options which he does not delve into, instead showing only the most basic elements. We also observe some elements mentioned in the other articles reviewed in this document, such as baselines, assignation rules, etc., which allow readers to get familiar with the terminology and to do comparisons in this area. For this reason, we deem this paper to be the complete one and the most appropriate for inexperienced readers who wish to know about the issue without delving too deep into the literature.

After reviewing the final set of papers, we establish that each of the proposals is a contribution to some specific features, while ignoring others which might also be significant. For this reason, we raise a key question: what is the level of maturity of MDD-based automatic user-interface generation? In the literature, authors reiterate that it is important to employ MDD, and they highlight the associated risks or difficulties. However, they provide no definitive solution. The central theme described in [19] by López-Jaquero and Montero is an ideal complement to [11], [12], and [17], as it centers around the design of adaptation rules. The more general approach to this theme is reutilization, which is one of the most valuable features as it centers directly around improving quality.

Finally, in the work of Vanderdonckt and Simarro [5], a pattern-based methodology is proposed for carrying out interface transformation. This is done by using an application consisting of a sizable amount of code, which corresponds to the patterns stored in it. We consider this work to be very complete, as the authors specify all the components employed for building the application.

5. CONCLUSIONS

Systems modeling is a powerful tool for the comprehension and design of solutions for a given problem. The usefulness of this tool lies in its capacity to manage complex systems by decomposing them into smaller, easier to tackle subsystems. Being able to obtain user-interfaces directly from such models not only implies increased productivity, but increased software product quality, as the model should cover all the requirements of the client.

Model-based automatic code generation is a quick way to obtain user-interfaces. There are many advantages to this approach, such as the inclusion of elements within model generation templates and the automation of repetitive tasks. However, as shown in this review, it is very hard to fully remove human intervention from the development process.

During the lifecycle of a software project, the need to make new changes will arise constantly in order to fulfill client needs. In many cases, such changes must be reversible, because the specific need of the client might also change, the proposed solution is not viable, etc. In any case, it is essential to be able to reverse changes, thus making it necessary for all changes made during development to be traceable.

There remain many challenges to fully automated model-based user interface generation. However, the proposed solutions (some of them already functional, others still in development) are relevant. The MDD concept is very attractive thanks to the advantages it provides. By solving these challenges, projects might be developed much faster without a loss of quality.

Finally, we can say that there are many proposed solutions for the problems that arise from model transformations, but there is no standard which solves such problems in a platform-independent way. From analyzing the selected papers, we saw that the authors propose different solutions, but none of them is definitive. It is up to developers to decide which solution is better adapted for their specific development environment and requirements. For this reason, it is always necessary to analyze the available options, and sometimes the best solution might be a combination of several such proposals.

6. REFERENCES

[1] D. Carrizo and J. Rojas, "Metodologías , técnicas y herramientas en ingeniería de requisitos : un mapeo sistemático Methodologies , techniques and tools in requirements," vol. 26, pp. 473–485, 2018.

[2] S. P. Jácome-guerrero and E. Salazar-jácome, "Software development environments and tools in MDE," pp. 1–15, 2018.

[3] J. Quintero, "Marco de Referencia para la Evaluación de Herramientas Basadas en MDA," in *Memorias del X Workshop IDEAS*, 2007, no. c, pp. 1–14.

[4] J. Gamalielsson, B. Lundell, and A. Mattsson, "Open Source Software for Model Driven Development: A Case Study," *Open Source Syst. Grounding …*, no. Ebert 2008, pp. 348–367, 2011.

[5] J. Vanderdonckt and F. M. Simarro, "Generative pattern-based design of user interfaces," *Proc. 1st Int. Work. Pattern-Driven Eng. Interact. Comput. Syst. - PEICS '10*, pp. 12–19, 2010.

[6] J. S. Cuadrado and J. De Lara, "AnATLyzer : An Advanced IDE for ATL Model Transformations," pp. 16–19.

[7] G. Hinkel, T. Goldschmidt, E. Burger, and R. Reussner, "Using internal domain-specific languages to inherit tool support and modularity for model transformations," *Softw. Syst. Model.*, vol. 18, no. 1, pp. 129–155, Feb. 2019.

[8] P. Mohagheghi, M. A. Fernandez, J. A. Martell, M. Fritzsche, and W. Gilani, "MDE Adoption in Industry: Challenges and Success Criteria," in *Models in Software Engineering*, 2009, pp. 54–59.

[9] M. Staron, "Adopting Model Driven Software Development in Industry – A Case Study at Two Companies," in *Model Driven Engineering Languages and Systems*, 2006, pp. 57–72.

[10] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?," in *Model-Driven Engineering Languages and Systems*, 2013, pp. 1–17.

[11] N. Aquino, "Adding flexibility in the model-driven engineering of user interfaces," in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems - EICS '09*, 2009, p. 329.

[12] S. Trujillo *et al.*, "Feature-oriented refinement of models, metamodels and model transformations," *ACM Int. Conf. Proceeding Ser.*, pp. 87–94, 2009.

[13] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," 2006, pp. 1051–1052.

[14] "Systematic reviews and meta-analytic techniques - ScienceDirect." .

[15] L. Askie and M. Offringa, "Systematic reviews and meta-analysis," *Semin. Fetal Neonatal Med.*, vol. 20, no. 6, pp. 403–409, Dec. 2015.

[16] B. Uzun and B. Tekinerdogan, "Model-driven architecture based testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 102, pp. 30–48, Oct. 2018.

[17] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, "Maintaining invariant traceability through bidirectional transformations," in *Proceedings - International Conference on Software Engineering*, 2012, pp. 540–550.

[18] J. Vanderdonckt, "Model-Driven Engineering of User Interfaces : Promises , Successes , Failures , and Challenges," in *Proceedings of the National Conference on Human-Computer Interaction*, 2008, pp. 1–10.

[19] V. López-Jaquero and F. Montero, "Diseño de reglas de adaptación y transformación para interfaces de usuario.," *RASI*, vol. 7, no. 1, pp. 53–58, 2010.

[20] C. Blanco, I. G. R. de Guzmán, E. Fernández-Medina, and J. Trujillo, "An MDA approach for developing secure OLAP applications: Metamodels and transformations," *Comput. Sci. Inf. Syst.*, vol. 12, no. 2, pp. 541–565, 2015.

[21] A. Rodrigues Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Comput. Lang. Syst. Struct.*, vol. 43, pp. 139–155, 2015.

[22] S. Fincher *et al.*, "Perspectives on HCI patterns," in *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*, 2003, p. 1044.