Evaluating the Benefits of Using Domain-Specific Modeling Languages - an Experience Report

Timo Wegeler
Fraunhofer Institute for Open Communication Systems FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
timo.wegeler@fokus.fraunhofer.de

Friederike Gutzeit
Fraunhofer Institute for Open Communication Systems FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
friederike.gutzeit@fokus.fraunhofer.de

Aurèle Destailleur
Fraunhofer Institute for Open Communication Systems FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
aurele.destailleur@fokus.fraunhofer.de

Bernhard Dock
Klopotek & Partner GmbH
Schlüterstraße 39
10629 Berlin, Germany
B.Dock@klopotek.de

ABSTRACT

There are many tools available for the creation of domain specific languages (DSLs) but the question remains how to identify appropriate use cases for the application of domain specific modeling and language design, and how to measure success. We report on our observations after three years of accompanying several real-life industrial DSL design projects and on our experiments with applying qualitative and quantitative evaluation criteria. We suggest an evaluation methodology spanning the entire DSL life cycle. It consists of an assessment of motivation, qualitative interviews, a validation of DSL design, quantifying benefits and a comparison of impacted workflows before and after adoption. We conclude with a discussion of inherent limitations.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.3.2 [**Programming Languages**]: Language Classifications—*Very high-level languages*

General Terms

Measurement, Languages

Keywords

Domain Specific Languages, Domain Specific Modeling, Metrics, Assessment, Evaluation

1. INTRODUCTION

Domain specific graphical or textual modeling languages are useful tools if applied to the right class of problems: Visual representations are especially appropriate where non-linear information flows need to be expressed, be it interactions, relations or state changes in a running system. In these areas, domain-specific concrete syntax also plays a key role and allows for easy recognition of important abstractions. However, often the cost of building a graphical modeling tool from scratch is too high and possibilities – or the cost – of tweaking existing tools are unknown. Therefore, instead of implementing a graphical modeling solution that would provide benefits to the modeling process, tools at hand are used. This may range from textual representations over textual DSLs to handmade diagrams on paper or using drawing tools which offer enough freedom. Using non-formal representations often helps to gain a common understanding of the problems at hand, but still lack the generative part at the end.

Once the initial cost of using frameworks for the definition of graphical or textual DSLs is overcome, the benefits of using a domain-specific modeling (DSM) solution can unfold. These benefits are often stated as "improving productivity by automating of common tasks", "raising the level of abstraction" and allowing domain experts to be involved in the development without requiring software development experience by using an expressive representation that allows for code generation [9]. In order to gain a better understanding of when it is worth to invest the initial effort to learn the usage of a DSL framework, our goal was to define criteria that help to identify successful applications of the DSM principle.

One of the main challenges for the creation of models or domain specific languages for use in software engineering is to find the right abstractions. Depending on the type of

model or DSL, there are a number of factors to be considered. For example, in the case of creating a DSL to be used by domain experts with little or no programming experience, these abstractions have to be mapped to concepts known to the user. When designing such a DSL, special attention has to be paid to the concrete syntax, be it of graphical or textual nature. However, it is difficult to measure the quality of a DSL regarding an appropriate syntax and the right abstractions, since it depends on each case, or more specifically, on each individual person and the concepts he is familar with. Therefore, the challenge for creating DSLs not only is a technical one, but also involves aspects of human cognition and other complex issues that are related to the difficulties of human social interaction and of the natural language used for communication.

For DSLs used in a more technical manner[1], the design problem of finding the right abstractions is more an issue of efficiency: Such a DSL is expected to increase productivity by reducing the lines of code that have to be written manually, or to enable or ease other aspects of software development like formal verification or test generation. This is a no less complex issue, and guidance for how to create such DSLs of high quality is also rare.

Our research is part of the BIZWARE[2] project where a systematic and standardized process of model-based software construction and operation using DSLs was researched (running 09/2010-08/2013).

2. RELATED WORK

The question of evaluating DSL solutions is gaining more and more attention in the scientific community. Due to the complexity of the problems addressed with DSM solutions, many authors use methods from empirical software engineering and deal with aspects of human cognition.

Khalaoui et al. have examined the success factors for domain specific modeling activities and compiled a list of qualitative criteria with positive and negative impacts [13]. In order to evaluate a DSM solution, Mohagheghi et al. [15] suggest using both quantitative and qualitative criteria, taking into account the stakeholder's interests.

Using qualitative research via interviews and online surveys, an empirical study about the acceptance of model-driven engineering in four industrial cases [16] indicates that the main motivation for the adoption of model driven engineering is better communication between stakeholders, consistency among development artifacts and higher productivity, while quality of code and decreased development time are not prioritized goals. Design patterns for the use of DSLs have been analyzed by Spinellis [19].

Rodriguez et al. suggest an evaluation management process for any kind of software artifacts, demonstrated with UML class diagrams [18]. They introduce checklists for the syntactic, semantic and pragmatic quality of an artifact. An empirical study with computer science students measured understandibility time to compare different solutions of a modeling example in UML [7]. They suggest that relationships with different semantic strengths affect the time needed to understand a model. Lahtinen et al. propose a wizard that guides the creation of a valid model using a task list [14] to help new users which are unaware of the underlying metamodel.

There are also approaches that treat DSLs as a UI and apply UI-related usability tests. An empirical validation considering the cognitive complexity of OCL expressions [17] hints that a cognitive complexity model can be utilized to measure the impact of a certain feature of OCL on comprehensibility. Metrics for the understandability of ER diagrams are defined and empirically validated by Genero at al. [8]. They argue that even a small set of simple structural metrics can indicate their understandability. Wu et al. [25] measure effort for creating software applications using DSML's and propose metrics.

The question whether a DSL is a modeling or a programming language has been examined by Sun et al. [22], delivering criteria to classify a language. Jackson and Sztipanovits have found differences in metamodel– and grammar-based DSLs and examined a bottom-up metamodeling approach where metamodels are transformed into constraints to ensure mathematical precision [11]. Wu et al. present a framework for generating DSL unit test engines [24]. A scenario-based approach for system validation is presented by Carioni et al. [3], using a UML profile for the domain specific modelling of embedded systems.

Gailly and Poels examined the domain-specific quality of profiled UML diagram variants and demonstrate that domain ontologies help the evaluation process [6]. Natural Language complexity is measured by Vulanovic [23], by creating a metric for grammar complexity. A way to estimate the complexity of building a model using a given metamodel is presented by Sprinkle [20]. Constructing comparable conceptual models with domain specific languages is examined by Pfeiffer [4].

In a report on technology transfer from academia to industry [2], the evolution of a domain specific language and its interface is analyzed and lessons learned include organizational and social aspects. The practical use of a visual DSL is reported by Karaila [12]. In particular, the evolutionary history is described and a supervised evolution process is considered a requirement for successful language development. Strembeck and Zdun [21] examined activities in a DSL engineering process, which were derived by analyzing industry and research DSL projects.

3. PRACTICAL DSL DEVELOPMENT EXPERIENCES

The BIZWARE consortium consists of two academic and eight industrial partners who develop software themselves to address different industrial domains: health care, finance, publishing, facility management, industrial production, web application development, and information systems integration. During the first two years of our collaboration, we analyzed motivation and benefits of a DSL development for the different domains as well as the demands for capabili-

[1] a.k.a. technical or horizontal vs. business or vertical DSLs

[2] funded by the German Federal Ministry of Education and Research (BMBF) under grant number (Förderkennzeichen): 03WKBU01B

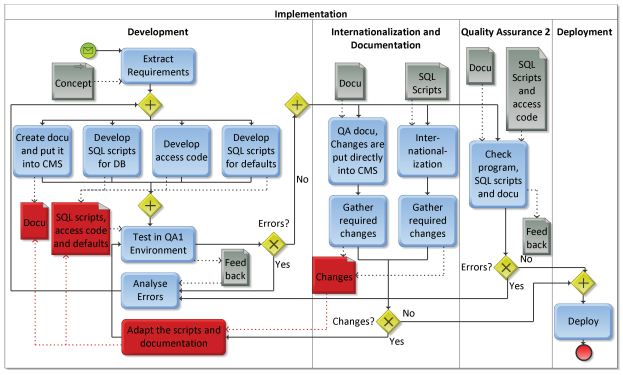

Figure 1: Development process prior to DSL usage

ties of DSL development toolkits over the full development lifecycle. The development process of such DSLs with industrial restrictions remains a complex task. In order to enable participative modeling together with domain experts, the academic partners concentrated on providing guidance to all participating stakeholders involved in a DSL development lifecycle from DSL designers to respective users and provided a toolkit to support the DSL development process with semantic knowledge bases and metrics [1].

One of our observations is that in business and industry, the use of non-formal DSLs is very common. Furthermore, there are popular tools like Microsoft Excel and Visio that are used to document workflows or software requirements. These tools also influence the way non-programmers conceptualize domain specific problems. It has proven to be a strong requirement that a DSL toolkit can be integrated into the existing software development workflow and its tool chain. One approach to orchestrate the exchange of models in a tool landscape is described by Ritter et al. [10].

Involving domain experts in software development often faces a mismatch in communication of requirements, concepts and use cases. Programmers do not understand the domain and how its concepts are used or intertwined; domain experts lack understanding of programming languages and computational limits. By using DSLs to allow modeling of domain concepts or to express domain functions, rules and computation, it is attempted to shorten this gap. The DSLs created in cooperation between domain user and DSL designer may be of different quality and suitable for different purposes, like (partial) code generation, test case generation, or for pure documentational purposes. However, to leverage any usefulness, the syntax has to be adapted carefully to the needs of the domain expert, as user acceptance of the notation plays a key role. At the same time, we observed that little changes made to the desired syntax due to tool limitations were accepted as well, as long as they were small enough.

Monitoring the DSL developments of our eight industrial partners, we found the most challenging aspects of the DSL design proved to be:

- finding the right abstractions
- finding an appropriate syntax
- collaboration issues
- lack of evolution support

In this paper we focus on two actual DSL developments; one textual DSL for the configuration of a large software product in the publishing sector; and one graphical DSL for the definition of processes for a web-based software product in the facility management sector. The DSLs were integrated into the existing software development workflow and toolkits: Java/Eclipse for the textual DSL and C#/Visual Studio for the graphical DSL.

4. EVALUATING DSM SOLUTIONS

Theoretically, in order to evaluate a DSM solution, the development would have to take place twice: once with and once without incorporating the DSM solution. Then, the overall development costs could be compared directly. As this is not feasible in practice, we followed an approach of comparing the development processes before and after the introduction of an DSM solution. Another possibility is to conduct experiments [5], however we found it difficult to find a way to estimate the worth of a good documentation which leads to fewer errors in the future. It is not feasible to estimate cost savings due to prevented errors. As a pragmatic way to deal with the reality of software development, we concentrated on qualitative aspects using interviews, and on simple quantitative measures using an effective[3] lines of code (LOC) comparison for starters. While in the literature we reviewed, there are some complex metrics to be applied to evaluate DSM solutions, we did not find any methodology for how to find the right abstractions for the concepts used in a DSL.

4.1 Our Evaluation methodology

From the observation of the DSL developments of our eight industrial partners, we developed a first sketch of a systematic evaluation method. Our evaluation methodology is summarized as follows:

- Assessment of motivation
  What problem is the DSL going to solve? Is a DSL adequate? What goals are to be achieved?
- Assessment of status quo and outcome
  Assess the situation before and after the implementation of DSM using qualitative interviews. What goals and benefits were achieved?
- DSL design validation
  Validate abstraction levels and concrete syntax; document the effort for DSL implementation
- Assessment of quantitative benefits
  Select an appropriate metric, e.g. LOC for otherwise manually written code
- Assessment of changes to the development process
  Assess organizational impacts and compare workflows

This way, it is possible to extract criteria that can be used to measure the benefits of the DSM solution. In our project's case, it was used to deal with the different experiences that our industry partners encountered, using different tools and working in different domains.

[3]excluding whitespace and comments, sometimes also called eLOC

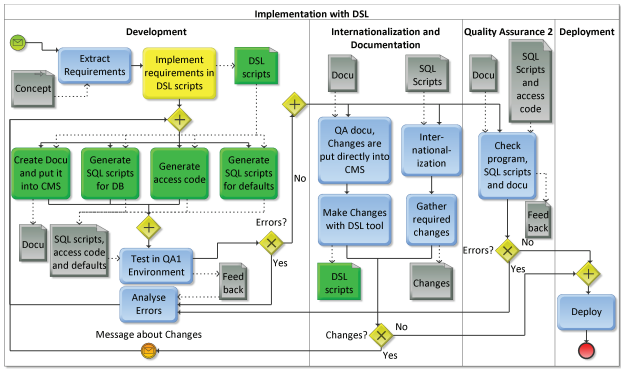

Figure 2: New development process with DSL

4.2 Evaluating the textual DSL

Klopotek develops complex software for the publishing sector. The main goal of the DSL development was to be able to better cope with complexity and to be able to quickly refactor and to increase developer efficiency.

4.2.1 Analysis of the development process

Out of four possible DSL developments, Klopotek chose the one with the most indicators for a successful DSM application: A clearly defined problem, a solution idea, available tools and support for its implementation.

The problem which was addressed existed in a specific part of the development process. It was a consistency problem and involved a tedious, repetitive task for the developers. Also, the development required collaboration with members of a different department. For example, changes deemed necessary were reported back from the other department as text documents and then had to be implemented by the development department, causing a slow process throughput time. The established process led to frustration among the developers and to inconsistencies between artifacts. Then, during the DSL design phase, it became clear that the parts that should be generated were relatively easy to implement using a DSL framework.

We documented the development process of our industrial partner Klopotek before and after the adoption of DSM during a series of interviews. Figure 1 depicts the development workflow prior to the introduction of the DSL. The problematic aspect was that the tasks of creating the documentation and the SQL scripts for the database were executed manually, often leading to inconsistencies (red boxes). Moreover, in the communication between the development department and the department for internationalization and documentation, changes deemed necessary were exchanged via a document, which proved to be inefficient (red boxes).

Figure 2 depicts the process after incorporating the DSL. While the diagrams are not completely different, a comparison shows that the steps previously marked critically have been replaced and extra steps have been added to include usage of the DSL editor. At the same time, an artifact which previously was created became obsolete and consistency issues could be resolved. Several steps that used to be manual labor changed to being automatically created (green boxes),

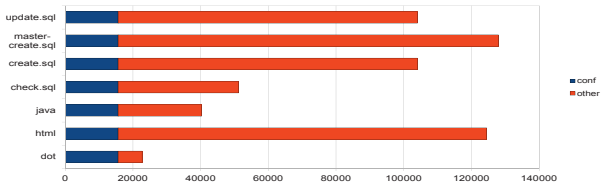

Figure 3: LOC for each generator in relation to the original DSL LOC

while the extra step of creating the DSL scripts is now critical (yellow box).

It is not possible to measure the time saved due to future errors and to inconsistencies that were prevented. But, as our qualitative analysis showed, developer contentedness has increased since the solution is in place. Analyzing the development process can also contribute to find a future field of application for DSLs. In the case of Klopotek, several DSL development opportunities were identified, and the most promising was then chosen and its development prioritized. The DSL was then implemented using Xtext.

4.2.2 Applying LOC metrics to a textual DSL

We then programmed a tool to analyze and visualize the LOC of all DSL files in use along with the generated files, ignoring comments and whitespace. From the data acquired, we could see that in some cases, the resulting generated LOC were lower than the LOC in the source DSL file. In these cases, the generated code was in fact another DSL code for a visualization tool. Measuring productivity in LOC only would have resulted in a negative outcome. As this generated code is used to improve the documentation, there is no way to express any benefit with the LOC metric. Figure 3 depicts the relation of total lines of code of the DSL scripts (”conf”) and the corresponding generated code (”other”).

The textual DSL was implemented using Xtext: It consists of 250 LOC for the grammar definition, 200 LOC for the validation, 140 LOC for the formatting and seven generator classes totalling 1500 LOC. At the time of this writing, 52 DSL scripts have been created, with a total of 15.600 LOC of DSL code. The generated code is totalling at 465.000 LOC.

Most of the generated code would have been written manually. But, as an existing code base has been replaced with the generated code, the real efficiency gain is for future changes. Also, some errors were discovered while rewriting the code based on the DSL. Since previously all code was written by experienced developers, we decided to use a simple LOC metric for the quantitative aspect of our evaluation. As the DSL is purely of declarative nature, it is difficult to apply other code metrics that measure complexity of the code although we plan to apply them to analyze the validation and generator classes in the future.

4.3 Graphical DSL evaluation

For the graphical DSL, a LOC metric is not directly applicable, although it would be possible to use its serialization as the basis. Instead, we decided to use a simple model metric counting the elements of the graphical view, and the proper-

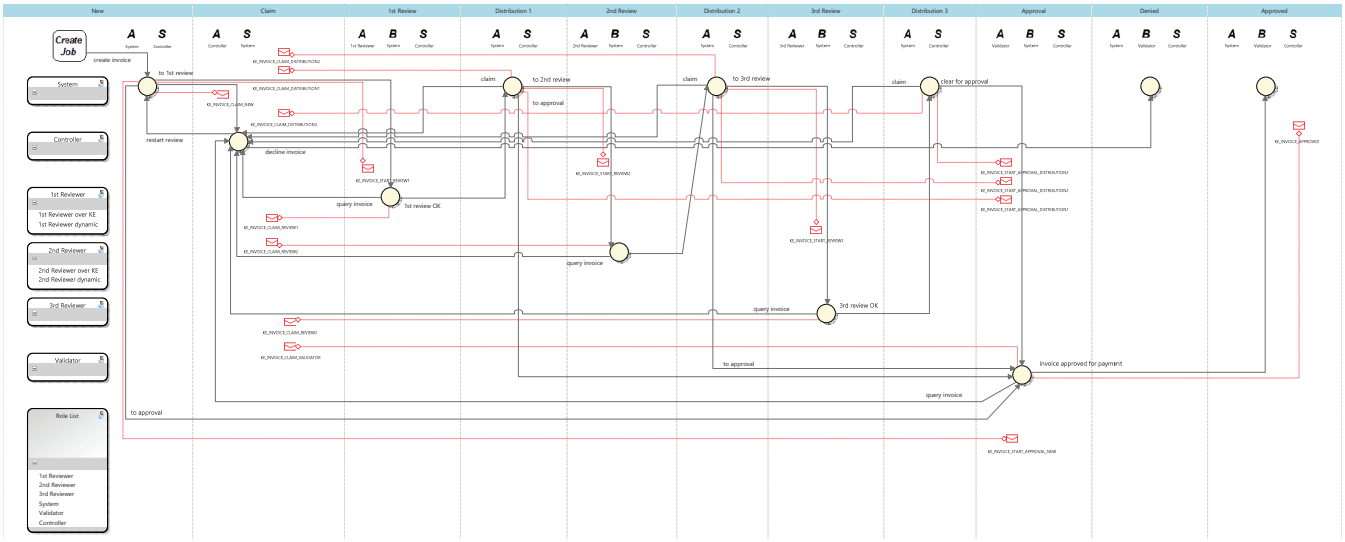

Figure 4: Graphical DSL implemented with Visual Studio DSL Tools

	script 1	script 2	script 3	script 4
graphical elements used (max: 14)	14	11	6	14
total elements used in diagram	75	112	17	111
total properties defined	394	542	68	517
LOC generated	5075	7108	614	7158

Table 1: Model Metrics

ties which have to be defined for each element in the editor. The advantage here is that prior to introducing DSM, the graphical view served as an isolated documentation artifact, while afterwards it was the basis for the generation of the code. Thus, the need to manually create the code was effectively traded for having to input a set of properties at the model editor. We analyzed four graphical models based on the DSL as shown in table 1.

We also evaluated the choice of appearance of the elements of the graphical DSL. From the 14 elements available in the editor, we created eight groups of elements of similar appearance, sharing the same visual features. We used similiarity groups to classify items and we then analyzed the semantic meaning of the elements in the group, determining a "conceptual distance". This uncovered an anomaly where two groups were nearly identical, the difference being one element which resembled the other elements too close, compared to the conceptual distance in the semantic net. Therefore, it was suggested to change the appearance of the element. Figure 4 depicts the graphical representation of one of the four DSL models.

5. CONCLUSIONS

We have illustrated how an evaluation of a DSM solution requires a mix of quantitative and qualitative criteria. Even though simple metrics cannot cover the complete benefits and risks of applying a DSM solution, they can be taken into account during the decision process whether or not to adopt DSM for future projects. We analyzed changes to the development process and found that for successful DSM solutions, they were not of radical, but rather gradual nature. Defining an evaluation strategy can help identify possible DSL fields of applications. While it is often not feasible to quantify and compare complex development efforts and the DSL benefits directly, a quality measure approach can be chosen to validate the DSM application, along with even simple metrics that can display benefits in an understandable way. We also introduced a method of comparing relative abstraction distances in order to assess concrete syntax and showcased the application of the method in a real-life development scenario. The graphical views of the DSL metrics can also be used to document DSL evolution, e.g. to compare different versions of the DSL in terms of code generation.

6. LIMITATIONS AND FUTURE WORK

Apart from the simple metrics we applied for a declarative DSL, for other kinds of DSLs more complex metrics could be applied. Also, the other metrics could be used to analyze the definition of the DSL, especially the validation and generator classes. Another possible metric could be to compare the number of concepts used in the DSL and the number of concepts used in the generated code. E.g., when HTML code is generated, there are a number of concepts that are used to define the HTML that remain hidden to the DSL user.

The cost of implementing and the learning curve for the implementation of the DSL is another factor which is relevant but hard to measure. In future work, we plan to give guidance to chosing the right abstractions on abstract as well as concrete syntax level.

7. REFERENCES

[1] H. Agt, R.-D. Kutsche, and T. Wegeler. Guidance for domain specific modeling in small and medium enterprises. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 63–70, New York and NY and USA, 2011. ACM.

[2] T. Aschauer, G. Dauenhauer, and W. Pree. A modeling language's evolution driven by tight interaction between academia and industry. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 49–58, New York, NY, USA, 2010. ACM.

[3] A. Carioni, A. Gargantini, E. Riccobene, and P. Scandurra. A scenario-based validation language for asms. In *Proceedings of the 1st international conference on Abstract State Machines, B and Z*, ABZ '08, pages 71–84, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Daniel Pfeiffer. Constructing comparable conceptual models with domain specific languages. In *15th European Conference on Information Systems (ECIS2007)*, pages 876–888, 2007.

[5] W. J. Dzidek, E. Arisholm, and L. C. Briand. A realistic empirical evaluation of the costs and benefits of uml in software maintenance. *IEEE Trans. Softw. Eng*, 34(3):407–432, 2008.

[6] F. Gailly. Conceptual modeling using domain ontologies : Improving the domain-specific quality of conceptual schemas. In *Domain-specific modelling workshop (DSM-10) - SPLASH/OOPSLA workshop*, number 2009/573. University Ghent, 2010.

[7] M. Genero, M. Piattini, S. Abrahao, E. Insfran, J. A. Carsi, and I. Ramos. A controlled experiment for selecting transformations based on quality attributes in the context of mda. *Empirical Software Engineering and Measurement, International Symposium on*, 0:498, 2007.

[8] M. Genero, G. Poels, and M. Piattini. Defining and validating metrics for assessing the understandability of entity-relationship diagrams. *Data Knowl. Eng.*, 64:534–557, March 2008.

[9] J. Gray, K. Fisher, C. Consel, G. Karsai, M. Mernik, and J.-P. Tolvanen. Dsls: the good, the bad, and the ugly. In *Conference on Object Oriented Programming Systems Languages and Applications archive*, Nashville and États-Unis, 2008. ACM.

[10] C. Hein, T. Ritter, and M. Wagner. Model-driven tool integration with modelbus. In *Workshop Future Trends of Model-Driven Development*, 2009.

[11] E. Jackson and J. Sztipanovits. Formalizing the structural semantics of domain-specific modeling languages. *Software and Systems Modeling*, 8:451–478, 2009. 10.1007/s10270-008-0105-0.

[12] M. Karaila. Evolution of a domain specific language and its engineering environment - lehmanâĂŹs laws revisited. In *Workshop on DomainSpecific Modeling at OOPSLA*, 2009.

[13] A. Khalaoui, A. Abran, and E. Lefebvre. Dsml success factors and their assessment criteria. *METRICS News*, Vol.13(No.1):p.p 43–51, February. 2008. Research Notes: 63.

[14] S. Lahtinen, J. Peltonen, I. Hammouda, and K. Koskimies. Guided model creation: A task-driven approach. In *Proceedings of the Visual Languages and Human-Centric Computing*, pages 89–94, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Parastoo Mohagheghi and Øystein Haugen. Evaluating domain-specific modelling solutions. In Juan Trujillo, Gillian Dobbie, Hannu Kangassalo, Sven Hartmann, Markus Kirchberg, Matti Rossi, Iris Reinhartz-Berger, Esteban Zimányi, and Flavius Frasincar, editors, *Advances in Conceptual Modeling - Applications and Challenges, ER 2010 Workshops ACM-L, CMLSA, CMS, DE@ER, FP-UML, SeCoGIS, WISM, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, volume 6413 of *Lecture Notes in Computer Science*, pages 212–221. Springer, 2010.

[16] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A. Fernández. 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.

[17] L. Reynoso, E. Manso, M. Genero, and M. Piattini. Assessing the influence of import-coupling on ocl expression maintainability: A cognitive theory-based perspective. *Information Sciences*, 180(20):3837 – 3862, 2010.

[18] M. Rodriguez, M. Genero, D. Torre, B. Blasco, and M. Piattini. A methodology for continuos quality assessment of software artefacts. In *Quality Software (QSIC), 2010 10th International Conference on*, pages 254 –261, july 2010.

[19] D. Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 56(1):91–99, 2001.

[20] J. Sprinkle. Analysis of a metamodel to estimate complexity of using a domain-specific language. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, pages 79–85, October 2010. ISBN: 978-952-60-1043-4, ISSN: 0356889X.

[21] M. Strembeck and U. Zdun. An approach for the systematic development of domain-specific languages. *Softw. Pract. Exper.*, 39:1253–1292, October 2009.

[22] Y. Sun, Z. Demirezen, M. Mernik, J. Gray, and B. Bryant. Is my dsl a modeling or programming language? In *Proceedings of 2nd International Workshop on Domain-Specific Program Development (DSPD)*, Nashville, Tennessee, 2008.

[23] R. Vulanovic. On measuring language complexity as relative to the conveyed linguistic information. *SKY - Journal of linguistics*, 20:399–427, 2007.

[24] H. Wu, J. Gray, and M. Mernik. Unit testing for domain-specific languages. In *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, DSL '09, pages 125–147, Berlin, Heidelberg, 2009. Springer-Verlag.

[25] Y. Wu, F. Hernandez, F. Ortega, P. J. Clarke, and R. France. Measuring the effort for creating and using domain-specific models. In *Proceedings of 10th Domain Specific Modeling Workshop*, 2010.