Usability Evaluation of Domain-Specific Languages

Ankica Barišić, Vasco Amaral, Miguel Goulão
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
a.barisic@campus.fct.unl.pt, vma@fct.unl.pt, mgoul@fct.unl.pt

***Abstract*—Domain-Specific Languages (DSLs) are claimed to bring important productivity improvements to developers, when compared to General-Purpose Languages (GPLs). The increased Usability is regarded as one of the key benefits of DSLs when compared to GPLs, and has an important impact on the achieved productivity of the DSL users. So, it is essential to build in good usability while developing the DSL. The purpose of this proposal is to contribute to the systematic activity of Software Language Engineering by focusing on the issue of the Usability evaluation of DSLs. Usability evaluation is often skipped, relaxed, or at least omitted from papers reporting development of DSLs. We argue that a systematic approach based on User Interface experimental validation techniques should be used to assess the impact of new DSLs. For that purpose, we propose to merge common Usability evaluation processes with the DSL development process. In order to provide reliable metrics and tools we should reuse and identify good practices that exist in Human-Computer Interaction community.**

Keywords: Domain-Specific Languages, Usability Evaluation, Software Language Engineering

I. INTRODUCTION

An increasing number of people rely on software systems to perform their daily routines and responsibilities. As such, systems need to be developed rapidly. Domain-Specific Languages (DSLs) are claimed to contribute to a productivity increase in software systems development, while reducing the required maintenance and programming expertise. The main purpose of DSLs is to bridge the gap between the Problem Domain (crucial concepts, domain knowledge, techniques, and paradigms) and the Solution Domain (technical space, middleware, platforms and programming languages). The sooner we fill in this gap, the sooner we shall increase users' productivity. However intuitive this idea may be, we need to have means to assess the Quality and success of the developed languages. The alternative is to accept the risk of building inappropriate languages that could even decrease productivity or increase maintenance costs.

Software Language Engineering (SLE) is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages. One of the crucial steps in the construction of DSLs is their validation. However, this step is frequently neglected. The lack of systematic approaches to evaluation, and the lack of guidelines and a comprehensive set of tools may explain this shortcoming in the current state of practice. To assess the impact of new DSLs we could reuse experimental validation techniques designed for User Interfaces (UIs) evaluation. The focus of this research proposal is to build up a conceptual framework that supports the development process of DSLs concerning the Usability evaluation. This will include concepts, methods, languages, processes, implementation of tools, and metrics proposal.

DSLs can be regarded as communication interfaces between humans and computers. In that sense, using a DSL is a form of Human-Computer Interaction (HCI). As such, DSLs evaluation could benefit from techniques used for evaluating regular UIs. We reviewed current methodologies and tools for the evaluation of UIs and General Purpose Languages (GPLs), in order to identify opportunities for improving the current state of practice in DSL evaluation. That brought us closer to providing adequate techniques for supporting the evaluation process which, we argue, should be based on methods for assessing user experience and customer satisfaction, applied to DSL users. By promoting DSL Usability to a priority in the DSL development, Usability must be considered from the beginning of the development cycle. One way of doing this is through user-centered methods. In order to tailor such methods to DSL development, we need to establish formal correspondences for all stages of the DSL development process and the Usability evaluation process.

This paper is organized as follows. In section II we discuss the current state of the art in DSL development and potential contributions from HCI to improve it. In section III we detail our research objectives and methodology. In section IV we report on the preliminary results in this research project, while in section V we outline our plans for future work and expected results. In section VI we present the conclusions for this paper.

II. STATE-OF-THE-ART

The immersion of computer technology in a wide range of domains, leads to a situation where the users' needs become increasingly demanding and complex. The Quality of the users' interaction with this kind of technology is becoming of the utmost importance. Consequently, the development of successful software systems becomes increasingly more complex.

Software engineers need to cope with the growing of both essential and accidental complexity [1]. They have to provide solutions that solve a class of crucial problems in a given domain, which are sometimes very complex to learn,

such as the rules and technical jargon found in domains like the Physics, Finance, Medicine, etc. Also, they need to deal with the accidental complexity of the used technology, *e.g.*, the use of low level abstraction programming languages, while integrating a wide plethora of different tools and libraries.

The use of the Model Driven Development (MDD) techniques and tools is seen as a viable approach for dealing with this accidental complexity[2]. MDD is grounded on the notion of providing explicit Models, commonly called *"first class artifacts"*, that are further translated into other lower level, more detailed, Models. These translations are also considered as development artifacts and can be explicitly modeled by means of transformation models. This approach has special impact in dealing with the complexity of large scale problems, while enabling rapid prototyping, simulation, validation and verification techniques [3], [4].

In direct relation with the MDD approach, we have modeling languages that are able to express the models with adequate notations. DSLs provide a notation tailored towards an application domain as they are based on models of relevant concepts and features of the domain [5]. As DSLs are used to describe and generate members of a family of systems in the application domain, they give the expressive power to generate the required family members more easily. As such they separate domain experts' work from analysis/transformation experts' work. DSLs are claimed to match users' mental model of the problem domain by constraining the user to the given problem [6].

In general, the software industry does not report investment on the evaluation of DSLs, as shown in a recent systematic literature review [7]. This conveys a perception that there is an insufficient understanding of the SLE process which, in our opinion, must include the evaluation of the produced DSLs. This apparent state of practice contrasts with the return of investment attributed to usability improvements reported for other software products [8]. In general, those benefits span from a reduction of development and maintenance costs, to increased revenues brought by an improved productivity by the end users [9].

The end user of the DSL can be a domain expert, a regular domain user, or a programmer that developing software systems for a specific domain. Each of these users has a different background profile and a different role in the problem solution. Both are expected to impact the way these users use a DSL. We need comparable validation procedures to assess user experience with DSLs, in contrast with whatever was the previous problem solving approach in that particular context.

Comparing the impact of different languages in the software development process has some tradition in the context of GPLs (*e.g.*, [10]). Typically, the popularity of a language is used as a surrogate for its usability, but this simplistic approach is not particularly interesting for DSLs, which often have a well-bounded set of target users (*e.g.* people working in a particular organization) Another shortcoming of the "popularity" approach is that it does not help identifying the strengths and weaknesses of a language, be it DSL or GPL. Other sorts of evaluations on GPLs include benchmarks, feature-based comparisons and heuristic-based evaluations [10],[11]. Since the end users of GPLs are usually closer to computation concepts, while the end users of DSLs are generally closer to domain concepts of the context of use, these methods cannot be directly applied for DSLs either.

When usability problems are identified too late in the language development process, a common approach to mitigate them is to build tool support that minimizes their effect on users' productivity [12], [13]. Better Usability is a competitive advantage, although evaluating it remains challenging, because it is hard to interpret existing metrics in a fair and unbiased way.

When compared to using GPLs, the increased productivity achieved by using DSLs is the one of the strongest claims of the DSL community[3],[4],[14]. The problem is that this claim is mostly based on anecdotal reports on improvements that lack external validity. Other reports, present maintainability and extensibility improvements brought by a combination of DSLs and Software Product Lines (SPLs) [15]. The usage of DSLs has been favorably compared to the usage of templates in code generation, with respect to flexibility, reliability and usability [16]. In a recent survey DSL users reported that they achieved noticeable improvements in terms of reliability, development costs, and time-to-market [6]. Comparisons can also be made among competing DSLs: for instance, [17] compares a visual DSL against the textual language for which it is a front-end.

DSLs define a way for human to communicate with machines. Therefore, DSL evaluation should not be much different from evaluating a regular UI. We can argue that any UI is a realization of a language, where a language is considered as a theoretical object (a.k.a. model) that describes the allowed terms and how to compose them into the sentences involved in a particular human-computer communication. Examples of UIs range from compilers to command-shell and graphical applications, and in each of those examples we can deduce the human-computer (H/C) language that is being used to perform that communication [29]. The general goal for HCI is that "*it should increase efficiency of humans performing their duties within a computation infrastructure, without extra organizational costs, inconveniences, dangers and dissatisfaction, as well as undesirable impacts on the environment during long periods of learning, or maintenance, among others"* [18].

Usually, there is a broad spectrum of issues to evaluate Software's Quality. Looking at the quality standards, and to the current Software Evaluation techniques we can fit them to the particular case of DSLs. In the literature, most of the requirements are actually associated with a qualitative software characteristic called Usability. The need for development of Usability definition is discussed in several

articles such as [19], [20]. The standards ISO/IEC 9241-11 (2001), ISO/IEC 9126 (2001) and ISO IEC CD 25010.3 [19] provide several definitions. The ISO IEC 9241-11 (2001) standard defines Usability as the *"extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*. ISO IEC 9126 (2001) gives us a quality model for achieving 'Goal Quality', *i.e.*, Quality in Use. ISO IEC CD 250100.3 estimated that model into complete Quality Model [21], where Usability is considered part of Quality in Use. In the context of DSL's evaluation [22], important notions such as Quality in Use, internal and external Quality were considered strongly dependent on the DSLs' intended context of use [27].

DSLs are built for a more confined context of use, and they capture one particular set of domain concepts. When we evaluate these languages, the population of users is smaller, and the external validity of the result is expected to be much higher than we would have for a UIs. In the context of potential language's optimization procedure, we expect to find more relevant and accurate interpretations for these results.

III. Research Objectives and Methodological Approach

Despite the advantages that DSLs might bring to Software Engineering (by mitigating the accidental complexity of software), in order to be widely adopted by Software Engineering professionals, we need to provide the means to assess their Quality in Use and success of implemented problem solution when compared to the other solutions. The alternative is to accept the risk of developing inappropriate DSLs that can decrease the domain developers' productivity or even increase maintenance costs.

We need a rigorous collaborative procedure in order to evaluate DSLs (both during and after their development), as well as evaluate their sentences (called instance models). For that it is necessary to:

a) Define the quality criteria to evaluate DSLs;

b) Integrate in an existing IDE support for development of DSLs with high Quality in Use; and

c) Define a methodological approach to support the evolution of a DSL's design based on user experience and infer its impact on quality improvement during its lifecycle (*e.g.* traceability of design decisions).

We propose to build a comprehensive methodology that involves Usability concern in all phases of existing DSLs' development process. We should research the most suitable means to provide both reliable DSL evaluation metrics and iterative suggestions during DSLs' development and evolution. This methodology will be based on user-centric techniques and cope with the DSL's evolution by assessing the impact of the changes in the DSL's design and implementation on user experience. In order to be able to build this methodology it is necessary to answer the following questions:

- What are the relevant quality concerns for DSL's evaluation, and associated metrics? How can we take advantage of these metrics to actually measure the quality in use of a DSL? Which existing standard DSLs can we take as a reference for performing DSLs comparison (or comparison of software languages in general)?
- How to plan an effective experimental evaluation of a DSL (*i.e.*, giving statistically significant results with the minimum effort)?
- How to guide the software language engineer in order to build a DSL with high level of Quality in Use? What are the good language design patterns? How can we foresee the Usability of a DSL while in an iterative evolution step?

The methodology will be validated by compilations based on recommendations that emerge from it in the development of the DSLs and experimental assessment of their impact trough few case studies on the different DSLs.

We foresee the following main research activities that need to be applied in each development step of DSL in order to introduce Usability evaluation into development process:

A. Domain Analysis

The Domain analysis phase is needed in order to understand the domain in consideration, by collecting information about it. The output of these phase is a domain model [23], that represents the common and varying properties of systems within the domain, the vocabulary used in the domain and defines concepts, ideas and phenomena, within the system. Existing systems, their artifacts (such as design documents, requirement documents and user manuals), standards, and customers are all potential sources of domain analysis input.

In this activity, we find it essential to define and model DSLs target users and intended context of use. Also, we propose new models, *e.g.* scenario-based modeling and goal-oriented modeling, which are based on assessment of users' previous experience. They should be included into the existing domain analysis models in order to define the usability requirements and crucial tasks that should be supported by the DSL under evaluation. Also, we find it crucial to relate these requirements to dependent user and context models. These models should be considered from the beginning of the DSL's development process as quality criteria for the newly designed language. During the development process these models should be refined according to results of validation recommendations.

B. Language Design

Designing DSLs remains a difficult and under-explored problem [31]. Recent work has focused mainly on the implementation of DSLs and supporting tools. Also, Volter

presents a collection of design patterns for describing the process of MDD. However, there still lacks detail for language design, development and implementation. We expect to contribute here with design patterns of Usability evaluation of DSLs.

In the Language Design activity, we propose to perform corpus evaluation of DSLs. Here, the main objective is to identify the means to evaluate the internal quality of a language, *i.e.*, in the perspective of language's evolution and validation. We expect to trace the impact of metamodel design changes, and collected statistic on the DSLs Usability.

C. Testing – Controlled experiment

The main objective of the testing activity is to identify the means to evaluate the Quality in Use of a language according to the requirement models described in the domain analysis phase. This involves the definition of experimental procedures/processes, heuristics and questionnaires. In order to be able to provide proper instrumentation for experimental evaluation, it is necessary to design support that will log Quality indicators, and present quantitative metrics result, so that developer is able to reason about the Quality in Use of implemented solution.

Designed instrumental support should be integrated into experimental model, so it can be validated trough controlled experiments. The quality in use of a language may be evaluated distinctly according to either its abstract syntax or concrete syntax which also implies the adoption of a (arguably) good interaction model. However, that is another aspect of usability evaluation of DSLs that is not part of this work. In scope of this work we find it necessary to evaluate only functional quality of concrete syntax, and not concentrate on evaluating concrete syntax by itself. Also, we will distinguish between evaluating a DSL from evaluating its implementing tool.

D. Deployment and Maintenance - Collect and evaluate the Quality the Instance Models (sentences)

The objective of this activity is to identify the appropriate means to qualify the instance models based on the users' feedback in the production environment. To be able to compare the (semantically equivalent) instance models expressed on the same language in a cognitive perspective we should revisit and improve corpus evaluation tools/techniques from testing activity. Also we should monitor the language's ability to support the evolution of the instance models without having negative impact of the languages usability.

E. Validation - Iterative life-cycle

The main objective of this activity is to build a conceptual framework to reason about the pertinence of the results of the language's Quality in Use in the overall language's life-cycle. It is important to identify what quality attributes (and corresponding metrics) have the most relevant impact on overall Quality in Use. We should evaluate impact of those quality metrics during following the language development step, as well as to validate suggestions for further improvements on the following steps. The framework should enable us to trace the impact of design changes on user experience with language and be interactively connected to the usability models proposed for another development activity.

By using existing language evaluation case studies we can compare the decisions from the reasoning framework, with the conclusions (considered sound by the community) taken from other language evaluation approaches. The expected output is a report containing a proof of correctness (completeness and soundness) of the conclusions taken by the reasoning framework on the observed case studies.

IV. Past Work and Preliminary Results

There are already many publications about UI Usability evaluation. However, we find that the Usability evaluation of a UI is typically superficial when compared to the required usability evaluation of DSLs. Existing methodologies do not cover all the relevant aspects and dimensions of usability evaluation, *e.g.* learnability, efficiency, effectiveness for all intended users and features of product. As it is hard to capture all the intended contexts of use for UIs at once, supporting tools are developed to support some parts of methodologies, usually built to provide questionnaires or collect some quantitative data, and are in most cases too general. Existing practices have very a low level of external validity, and sometimes it is hard to interpret what the collected information means, probably because of the wide spectrum of contexts of use that they target.

DSLs can have a precise definition of the end user's profile and task models, as well as syntactic models, that our method uses in order to achieve better results from its Usability evaluation. Moreover, we can rely on these results in order to validate the claim that DSLs can effectively narrow the gap between humans and computers, when compared to regular GPLs.

A. Iterative user-centered design

According to Mernik *et al.*, the Language life cycle consists of a set of phases [5]: *Decision*, *Domain Analysis*, *Design*, and *Implementation*. Visser adds *Deployment* and *Maintenance* to this process [23]. Besides adding *Testing* (as in any typical Software Product), we propose to introduce *Language Evaluation* just before *Deployment* [24]. This *Language Evaluation* phase is done with language quality concerns in an incremental and iterative user-centric approach, with the DSL end users, while crosscutting all of the involved phases, as suggested in [25].

By allowing significant changes to correct deficiencies along the development process, instead of just evaluating the DSL at the end of the process, when it might be too late, user-centered design can reduce the cost of development and support. The critical activities required to implement user-centered design are described in ISO 13407 [20]. Once the system is released to the users, an user experience assessment of DSLs and associated IDE may be highly beneficial [19].

An *Iterative Usability evaluation approach* should be merged with the DSL development cycle, as described in [22]. This approach supports reasoning about the already implemented and wished problem domain concepts of DSLs users. In a first moment, by defining them for the user and context models in the domain engineering phase, designing and implementing them in the language. In a later stage, the language concepts should be validated in the testing phase, along with the development environment proposed for using the DSL. Note that the combination of language and tool support is essential in the evaluation, because language usage will be significantly impacted by its tool support. As such, it is essential that the iterative usability evaluation covers both.

B. Context-dependent evaluation

Empirical evaluation with users, is recommended at all stages of development, or at least in the final stage of development [26]. To do so, we can use several methods, with different kinds of measures, where each type of measure is usually regarded as a separate factor with a relative importance that depends on the DSL's context of use [27]. These evaluations can be designed to target specific profiles of DSL users in order to increase their replicability.

For several predefined groups of DSL users we should use techniques like questionnaires, and observations to analyze the tasks involved while using a given DSL. Observations should include capturing quantitative indicators related to users' interaction with the DSL environment (*e.g.* mouse movements, keystrokes, heartbeats, or eye tracking). Experimenters in human factors have developed a list of tasks that can capture these particular aspects [28]. These tasks should be designed to capture relevant Usability concerns, *e.g., effectiveness, efficiency* or *satisfaction*. We propose a systematic approach based on UIs experimental validation techniques to assess the impact of the introduction of DSLs on the productivity of its end users. To illustrate this evaluation approach we have presented a case study of a DSL for High Energy Physics [29].

C. Experimental Language Evaluation

We argue that the Quality in Use of a DSL should be assessed experimentally. In Software Engineering, a controlled experiment can be defined as "*a randomized experiment or quasi-experiment in which individuals or teams (the experimental units) conduct one or more Software Engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages or tools (the treatments)*" [30]. In the case of DSLs, this can be instantiated in early phases of development with domain experts that typically have to conduct with software construction, or evolution tasks. For the sake of comparing different languages, including the DSL under evaluation and any existing baseline alternatives to that DSL, representative user groups should be modeled and involved.

We proposed a general experimental evaluation model, tailored for DSLs' experimental evaluation, and its instantiation with several DSL evaluation examples [24]. These instantiations served as a proof of concept for the proposed experimental evaluation process. Our evaluation model can be instantiated for repeated evaluations of a DSL, thus building up a longitudinal evaluation of the DSL, while it evolves. This enables us to track and control the impact and scope of changes in the DSLs. The model also facilitates reasoning about which Usability levels are achieved for each user profile population, which can help language engineers in determining when the desired quality in use level is achieved (*i.e.* when additional changes do not have any more significant impact in the Usability of DSL). The representation of the evaluation as an instance of our evaluation model also facilitates the comparison of alternative DSL solutions, as well as the replication of previous evaluations, their approaches and decision models.

V. FUTURE WORK AND EXPECTED RESULTS

Our research will follow by proposing metrics and methodologies for Usability evaluation of DSLs, whose validity should be supported by real life experiments with users of existing DSLs. In order to do that, we find it necessary to define conceptual distance as the distance between concepts in the users' mind and the conceptual domain of a language. If we are able to measure that distance, and have methods that will minimize it, we can support the claim that DSLs are able to close the gap between domain experts and solution domain.

An additional step is to conceptualize models for performing DSL's evaluation *i.e.* quality model, instruments model, metrics and traceability model of design changes and their impact. This support should be tailored to internal and external quality attributes (such as syntactic and semantic models of the DSL under evaluation) and user's experience while using a DSL along several iterative evolution steps.

By providing that kind of support, we can effectively perform evaluation, whose outcome can be used to help increasing users' productivity, and explicitly model all the process. This evaluation procedure will give us faster convergence of language development, as we are able to monitor the impact of language evolution in the efficiency and effectiveness of practitioners using the language (and its companion toolset). As a side effect, we expect our evaluation work to contribute to the validation of the claim that DSLs are more usable then GPLs.

The impact of an evaluation process for DSLs is expected to be interesting from an industry point of view. With many organizations developing their own languages, or hiring companies to develop such languages for them, this framework will aid them in reaching more usable languages.

VI. CONCLUSION

Building DSLs is becoming very popular and by that there are increasing needs of some pointers in topic of their cognitive congeniality to end user. Although pragmatic, reactive approaches would not be necessary if domain experts could develop applications easily. It is necessary to explore more proactive approaches to improving DSLs'

Usability. We need to build a comprehensive methodology that support all phases of the Usability evaluation process and indicate ways to provide reliable metrics for supporting this evaluation. This is expected to enhance the community's awareness and recognition of the relevance of this topic in the process of SLE.

ACKNOWLEDGMENT

The authors would like to acknowledge CITI - PEst - OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) - for the financial support for this work.

REFERENCES

[1] F. P. Brooks, "The Mythical Man-Month: Essays on Software Engineering", Addison-Wesley Publishing Company, 3rd Edition, 1995.

[2] M. Volter and T. Stahl, "Model-Driven Software Development". Glasgow, UK: Wiley, ISBN: 0-470-02570-0, 2006.

[3] S. Kelly and J.-P. Tolvanen, "Visual domain-specific modelling: benefits and experiences of using metaCASE tools," Proc. International Workshop on Model Engineering, at ECOOP'2000, 2000.

[4] D. M. Weiss and C. T. R. Lai, "Software Product-Line Engineering: A Family-Based Software Development Process". Addison Wesley Longman, Inc., ISBN: 0-201-69438-7, 1999.

[5] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," ACM Computing Surveys, vol. 37, No. 4, pp. 316-344, December, 2005.

[6] F. Hermans, M. Pinzger, and A. V. Deursen, "Domain-Specific Languages in Practice: A User Study on the Success Factors," Proc. 12th International Conference on Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, Denver, Colorado, USA, October, 2009, pp. 423-437, doi: 10.1007/978-3-642-04425-0.

[7] P. Gabriel, M. Goulão, and V. Amaral, "Do Software Languages Engineers Evaluate their Languages?," Proc. XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), ISBN: 978-9978-325-10-0, Universidad del Azuay, Cuenca, Ecuador, April, 2010, pp. 149-162.

[8] J. Nielsen, and S. Gilutz, "Usability Return on Investment", Nielsen Norman Group, 4th edn. 2003.

[9] A. Marcus, "The ROI of Usability", in Bias, and Mayhew (Eds.): "Cost-Justifying Usability", North- Holland, Elsevier, 2004.

[10] L. Prechelt, "An Empirical Comparison of Seven Programming Languages," IEEE Computer, vol. 33, No. 10, pp. 23-29, October, 2000, doi: 10.1109/2.876288.

[11] D. L. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering", IEEE Transactions on Software Engineering, 2009, pp. 756-779.

[12] K. Y. Phang, J. S. Foster, M. Hicks, and V. Sazawal, "Triaging Checklists: a Substitute for a PhD in Static Analysis". Proc. Evaluation and Usability of Programming Languages and Tools (PLATEAU 2009), 2009.

[13] R. Bellamy, B. John, J. Richards, and J. Thomas, "Using CogTool to model programming tasks". Proc. Evaluation and Usability of Programming Languages and Tools (PLATEAU 2010), 2010.

[14] MetaCase: "EADS Case Study", http://www.metacase.com/papers/MetaEditinEADS.pdf, 2007.

[15] D. Batory, C. Johnson, B. MacDonald, and D. v. Heeder, "Achieving extensibility through product-lines and domain-specific languages: a case study," ACM Transactions on Software Engineering and Methodology, vol. 11, No. 2, pp. 191-214, April, 2002, doi: http://doi.acm.org/10.1145/505145.505147.

[16] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton, "A Software Engineering Experiment in Software Component Generation," Proc. 18th International Conference on Software Engineering (ICSE'1996), IEEE Computer Society, Berlin, Germany, March, 1996, pp. 542-552, doi: 10.1109/ICSE.1996.493448

[17] N. S. Murray, N. W. Paton, C. A. Goble, , and J. Bryce, "Kaleidoquery--a flow-based visual language and its evaluation", Journal of Visual Languages & Computing, 2000, 11, (2), pp. 151-189.

[18] T. Catarci, "What happened when database researchers met usability", Information Systems, 2000, 25, (3), pp. 177-212

[19] H. Petrie, N. Bevan, "The evaluation of accessibility, usability and user experience", in C. Stephanidis, (Ed.): "The Universal Access Handbook", CRC Press, 2009.

[20] N. Bevan, "Cost benefits framework and case studies", in "Cost-Justifying Usability: An Update for the Internet Age". Morgan Kaufmann, 2005

[21] N. Bevan, "Extending quality in use to provide a framework for usability measurement", Human Centered Design, 2009, pp. 13-22

[22] A. Barišić, V. Amaral, M. Goulão, and B. Barroca, "How to reach a usable DSL? Moving toward a Systematic Evaluation", Electronic Communications of the EASST (MPM), 2011

[23] E. Visser,: "WebDSL: A Case Study in Domain-Specific Language Engineering", in Book WebDSL: A Case Study in Domain-Specific Language Engineering' (Springer, 2007, edn.), pp. 291-373

[24] A. Barišić, V. Amaral, M. Goulão, and B. Barroca, "Evaluating the Usability of Domain-Specific Languages", in M. Mernik, (Ed.): "Formal and Practical Aspects of Domain-Specific Languages: Recent Developments", IGI Global, 2012, in press.

[25] C. Atkinson, and T. Kühne, "Model-Driven Development: A Metamodeling Foundation', IEEE Softw., 2003, 20, pp. 36-41

[26] J. Nielsen, and R. Molich, "Heuristic evaluation of user interfaces". Proc. SIGCHI Conference on Human Factors in Computing Systems: Empowering People (CHI'90), Seattle, Washington, United States, 1990.

[27] A. Barišić, V. Amaral, M. Goulão, and B. Barroca, "Quality in Use of DSLs: Current Evaluation Methods". Proc. 3rd INForum - Simpósio de Informática (INForum2011), Coimbra, Portugal, September 2011.

[28] P. Reisner, "Query languages": in "Handbook of Human-Computer Interaction", North-Holland, 1988, pp. 257-280.

[29] A. Barišić, V. Amaral, M. Goulão, and B. Barroca, "Quality in Use of Domain Specific Languages: a Case Study". Proc. 3rd ACM SIGPLAN workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2011), Portland, USA, October 2011 pp. 65-72

[30] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. Rekdal, "A survey of controlled experiments in software engineering," *IEEE Transactions on Software Engineering*, vol. 31, No. 9, pp. 733-753, September, 2005.

[31] M. Pfeiffer, and J. Pichler, "A comparison of tool support for textual domain-specific languages", UAP Printing Solutions, 2008, pp. 1-7.

[32] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley Publishing Company, ISBN, 1995.