Iterative Evaluation of Domain-Specific Languages

Ankica Barisic

Iterative evaluation of Domain-Specific Languages

Ankica Barišić

CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Campus de Caparica, 2829-516 Caparica, Portugal
a.barisic@campus.fct.unl.pt

Abstract. As software moves to the daily routines and responsibilities of people, there is a need for developing tools and languages rapidly. Domain-Specific Languages (DSLs) are claimed to contribute to this productivity increase, while reducing the required maintenance and programming expertise. DSLs are designed to bridge the gap between the problem domain (essential 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 assert the quality and success of the developed languages. The alternative is to accept the risk of deriving inappropriate products that bring more harm by decreasing productivity or even increasing maintenance costs.

Keywords: Experimental Software Engineering, Domain-Specific Languages, Software Language Engineering

1 Should language engineers evaluate their languages?

Domain-driven development is becoming increasingly popular, as it raise the abstraction level. The language engineer need to deal with the accidental complexity of the used computer technology e.g., the use of low level abstraction programming languages, while integrating a wide plethora of different tools and libraries. On other hand, the DSL development requires domain and language development expertise, and few people have both. This lead language engineers to cope with the growing of both essential and accidental complexity [8].

Software language engineering is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages. Although, the phases of DSL life cycle are systematically defined [17], [23], it seam that process lack one crucial step [5], namely language evaluation just before the deployment. This phase can be done with quality in use concerns in an incremental and iterative user-centric approach, while crosscutting all of the involving phases as suggested in [1].

The software industry does not seem to report investment on the evaluation of DSLs, as shown in a recent systematic literature review [11]. The lack of systematic approaches to evaluation, and the lack of guidelines and comprehensive set of tools may explain this shortcoming in the current state of practice. This is supported by the evidence of an interesting return of investment on usability evaluation for other software products [21]. Moreover, the benefits of usability evaluation span from a reduction of development and maintenance costs, to increased revenues brought by an improved effectiveness and efficiency by the end users [16].

It is arguable that the main reason for the perceived high costs of DSL evaluation, is the lack of a consistent and computer-aided integration of two different and demanding complementary processes: SLE process, and software language evaluation process. On the one hand, software language engineers should become aware of quality concerns during language development, and identify and apply best practices into their development plan. On the other hand, evaluation experts should get better understanding of all the models involved in the software language development in order to be able to give appropriate and reliable suggestions towards the improvement of the DSL under development. The focus of this research is to propose systematic evaluation process for DSLs with usability concern [4].

2 A language is a means of communication

A programming language is a model that describes the allowed terms and how to compose them into valid sentences. DSLs that are interest of this research are generally conceived as communication interfaces between human and computers [5]. User Interfaces (UIs) are also seen as a realization of language. Therefore, from one perspective evaluating DSLs is not much different from evaluating regular User Interfaces (UIs).

Empirical, or experimental, evaluation studies of UI with real users is a crucial phase of its validation [10]. A relevant set of quantitative and qualitative measurements must be inferred and combined together to lead to a useful assessment of the several dimensions that define software Quality in Use (such as Efficiency, Effectiveness and Satisfaction), often referred as Usability [12]. These complex experimental evaluation studies are typically implemented by software evaluation experts. Their expertise is essential to properly design the evaluation sessions, gather, interpret, and synthesize significant results. However desirable it can be to have such software evaluation experts in the teams, it is not always possible to have them available due to, among several reasons, the cost and time involved. This situation calls for the need of automatic tools that support these experts, as well as language developers. One way to obtain qualitative measurements is by means of observations and direct questionnaires to the end users [22].

There is an increasing awareness to the quality in use of languages, fostered by the competition of language providers. Better usability is a competitive ad-

vantage, although evaluating it remains challenging. While evaluating competing languages it is hard to interpret the existing metrics in a fair, unbiased way, provide reliable design changes and assure that scope of evaluation is preserved to target user groups.

When we consider General Purpose Languages (GPLs), their users are part of population that master well mathematical and technical concepts. In order to develop programming solutions they need to master also domain concepts. On other hand, as DSLs are ment to reduce use of computation domain concept by putting focus on the domain concepts, they are expected to be used by the much diverse target population.

The increased productivity achieved by using DSLs, when compared to using GPLs, is one of the strongest claims by the DSL community. With anecdotal boosted speed development reports of DSLs ([13], [24], [19], [18]) in industrial settings, why bother with its validation? The problem, of course, is that those anecdotal reports on improvements lack external validity.

3 Approach

As result of inspection of current methodologies and tools for evaluation of UIs and GPLs we propose an iterative user-centered approach for evaluation of DSLs. Goal of this approach is to establish formal correspondences between the DSL development process and the experimental evaluation at all the stages [7].

Approach is described by set of patterns that are introduced in order to provide a complete solution to a complex problem of placing intended users as a focal point of DSLs design and conception, and by that ensure that the language satisfies the user requirements [3]. Using the goal of these enter-depended patterns is to disseminate the knowledge of best practices to end users. It provides means of performing experimental validation in the most costeffective manner and is expected to give the rationale about correct and usable indicators that can eventually be reused.

The patterns are divided in three spaces, that represent different level of abstraction. *Agile Development Process* gives set of patterns devoted to project management and engineering of a DSL. This is the most important set of patterns, as it is trough organization and planing of language development and evaluation activities and goals we are controlling and tracking success of produced language. After an iteration, goals are scoped and budget is fixed, we are ready to proceed to design and implementation activities that are guided by patterns given in *Iterative User-Centered Design* pattern space. As the users are the central part of a DSL evaluation, this patterns considers how to engage the user in the development process and how to collect valuable information about the DSL and its level of usability while it is being developed. Finally, they are expected to result with concrete hypothesis, tests, metrics, samples and statements that should be addressed and validated trough *Experimental Evaluation Design*.

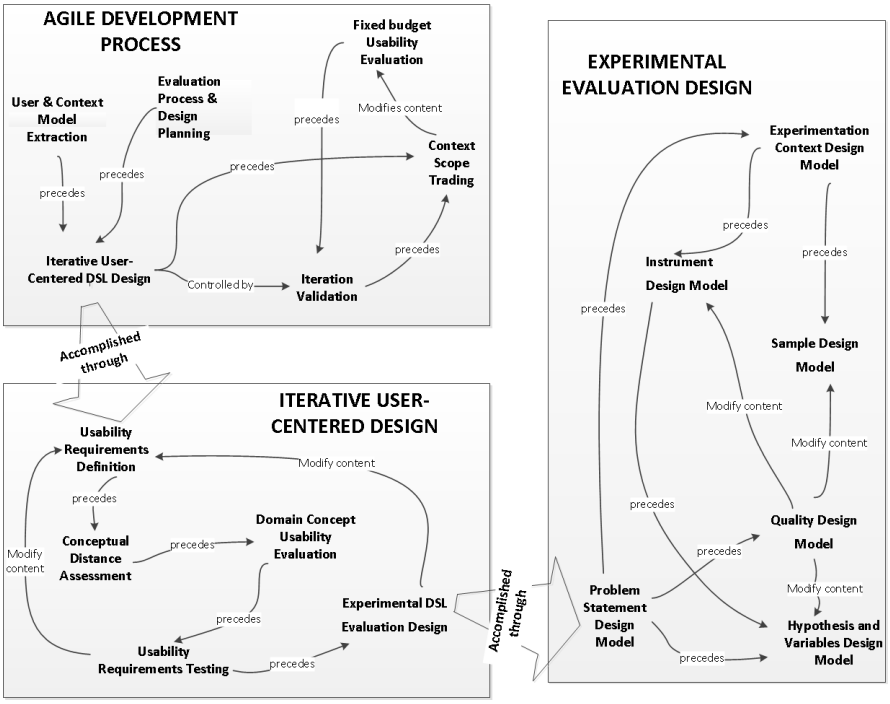

Fig. 1. Patterns for evaluating Usability of DSLs (taken from [3])

Following this best practices, each development iteration is focusing on different increment or level of abstraction that will be evaluated or refined. By planing carefully development process and organization of responsibilities and costs, goal is to establish balanced menagment and engineering plan that will satisfy both: business and user needs, by optimizing impact of evaluation feedback on language development. Also, time that is invested into strategy and design of problem and its solution can be planed well with technical implementation of solution.

According to it set of language and evaluation goals should be identified during *domain engineering phase* of DSL's construction i.e. while eliciting minimum set of domain concepts. A first step would be to understand and specify the context of use of DSLs and which kind of user groups it should target by constructing User and Context model. In order to achieve that, interviews or questionnaires with the DSL's intended end users should be designed in order to capture information about their working environment and the baseline approach to solve problems. In the *language design phase*, it is necessary to identify which quality attributes are impacted by the implementation of which domain concepts or layer of abstraction. During the *implementation phase*, the language engineer can benefit from the collected information by means of tools or instruments that implements chosen measures directly on the DSL prototype. Finally, in the *test-*

ing phase, the language engineer should conduct (at least) a expert evaluation to validate that the known quality problems and functional tests passed well. When seams that evaluation goals are met, we should conduct a user-based evaluation, in a real context of use, to assess the DSL's quality in use. That is done by giving the users real problems to solve in order to cover the most important tasks identified in the domain. Data about satisfaction and cognitive workload should also be evaluated subjectively through questionnaires. It is especially important in this phase to measure all the learnability issues, since DSLs should be (in principle) easy to learn and remember. Examples of the user based evaluations of DSLs, that presents examples of tasks and questions that are constructed to measure achieved Quality in Use can be seen in [6] and [15].

4 Experimental validation

Under the perspective of SLE, in order to experimentally evaluate a DSL, we need to know what is the criteria involved, understand notion of quality from the relevant perspectives and understand the experimental process itself. This complex challenge with respect to reuse was covered by general model for DSL experimental evaluation presented in [2]. This experimental model served as a set of proof of concept instantiations of the proposed experiment.

Experimental model outlines the activities needed to perform an experimental evaluation of a software engineering claim, following the scientific method. In order to effectively reason about experimental process and eventually detect flaws before it is applied and analysed we systematically compared four language evaluation experiments ([6], [14], [15], [20]). These evaluations are currently exceptional in the realm of DSLs and are chosen precisely for that: they are examples of best practices in languages evaluation with a concern on quality in use, from which we can perform some meta-analysis, leading not only to a collection of lessons learned from the trenches, but also to the identification of opportunities to further improve existing validation efforts.

By allowing significant changes to correct deficiencies along the development process instead of just evaluating at the end of it (when it might be too late), presented user-centered design is ment to reduce development and support costs, increase sales, and reduce staff cost for employers [9]. The proof of this claims is expected to be justified by the set of experiments of DSL development in academical and industrial cases.

Acknowledgments I gratefully thank to my supervisors Vasco Amaral and Miguel Goulão . This work was partially supported by the CITI - PEst - OE /EEI /UI0527 /2011, Centro de Informtica e Tecnologias da Informao (CITI/FCT/UNL) - 2011-2012)

References

1. Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20:36–41, September 2003.

2. A. Barišić, V. Amaral, M. Goulão, and B. Barroca. Evaluating the usability of domain-specific languages. In Marjan Mernik, editor, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pages 386–407. IGI Global, 2012.
3. A. Barišić, V. Amaral, M. Goulão, and M.P. Monteiro. Patterns for evaluating usability of domain-specific languages. *Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012*, September 2012.
4. Ankica Barisic, Vasco Amaral, and Miguel Goulão. Usability evaluation of domain-specific languages. In *Eighth International Conference on the Quality of Information and Communications Technology (QUATIC), 2012*, pages 342–347. IEEE, 2012.
5. A. Barišić, V. Amaral, M. Goulão, and B. Barroca. How to reach a usable dsl? moving toward a systematic evaluation. *Recent Advances in Multi-paradigm Modeling (MPM 2011), Electronic Communications of the EASST*, 50, October 2011.
6. A. Barišić, V. Amaral, M. Goulão, and B. Barroca. Quality in use of domain specific language: a case study. *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2011), held at Splash 2011*, pages 65–72, October 2011.
7. A. Barišić, V. Amaral, M. Goulão, and B. Barroca. Quality in use of dsls: Current evaluation methods. *Proceedings of the 3rd INForum - Simpósio de Informática (INForum2011)*, September 2011.
8. Fred Brooks. *The Mythical Man.* Addison-Wesley, 1975.
9. T. Catarci. What happened when database researchers met usability. *Information Systems*, 25(3):177–212, 2000.
10. Alan Dix. *Human computer interaction.* Pearson Education, 2004.
11. Pedro Gabriel, Miguel Goulão, and Vasco Amaral. Do software languages engineers evaluate their languages? In Xavier Franch, Itana Maria de Sousa Gimenes, and Juan-Pablo Carvallo, editors, *XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), ISBN: 978-9978-325-10-0*, pages 149–162, Cuenca, Ecuador, 2010. Universidad del Azuay.
12. International Standard Organization. Iso/iec 9126: Information technology - software product evaluation - software quality characteristics and metrics, 2004.
13. Steven Kelly and Juha-Pekka Tolvanen. Visual domain-specific modelling: benefits and experiences of using metacase tools. In Jean Bézivin and J. Ernst, editors, *International Workshop on Model Engineering, at ECOOP'2000*, 2000.
14. 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. *Proceedings of the 18th international conference on Software engineering*, page 552, 1996.
15. Toma Kosar, Marjan Mernik, and Jeffrey Carver. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering*, pages 1–29, 2011.
16. Aaron Marcus. The roi of usability. In Bias and Mayhew, editors, *Cost-Justifying Usability.* North- Holland: Elsevier, 2004.
17. Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.
18. MetaCase. Eads case study, http://www.metacase.com/papers/metaedit_in_eads.pdf. Technical report, MetaCase, 2007.
19. MetaCase. Nokia case study, http://www.metacase.com/papers/metaedit_in_nokia.pdf. Technical report, MetaCase, 2007.

20. 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*, 11(2):151–189, 2000.
21. Jakob Nielsen and S. Gilutz. Usability return on investment. Technical report, Nielsen Norman Group, 2003.
22. J. Rubin and D. Chisnell. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. Wiley-India, 2008.
23. Eelco Visser. Webdsl: A case study in domain-specific language engineering. *In Generative and Transformational Techniques in Software Engineering II, Ralf Lmmel, Joost Visser, and Joo Saraiva (Eds.). Lecture Notes In Computer Science*, 5235, 2007.
24. David M. Weiss and Chi Tau Robert Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison Wesley Longman, Inc., 1999.