Evaluating the Quality in Use of Domain-Specific Languages in an Agile Way

Ankica Barisic

Evaluating the Quality in Use of Domain-Specific Languages in an Agile Way

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. To deal with the increasing complexity of the software systems to be developed, it is important to have high level approaches to development that support such complexity at affordable costs. This fosters the development of Domain-Specific Languages (DSLs) that are claimed to bring important productivity improvements to software developers. Because of that, the quality of the users interaction with this kind of technology is becoming of utmost importance. The goal of this research is to contribute to the systematic activity of software language engineering by focusing on the issue of quality in use evaluation of DSLs, in which design decisions are validated iteratively.

Keywords: Experimental Software Engineering, Domain-Specific Languages, Software Language Engineering, Quality in Use, Agile development

1 Motivation

As software moves to the daily routines and responsibilities of people, there is a need for developing tools rapidly. Domain-Specific Languages (DSLs) are claimed to contribute to a productivity increase in software systems development, while reducing the required maintenance and required 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). DSLs empower domain experts to specify systems for their own domains. However intuitive this idea may be, we need to have means to assert the quality and success of the developed languages. We argue that for this kind of software languages the measure of success has to be captured by assessing the real impact of using DSL in real context of use by their target domain users. The alternative is to accept the risk of deriving inappropriate products that bring more harm by decreasing productivity or even increasing maintenance costs at the long run.

2 Problem

Domain-driven development is becoming increasingly popular, as it raises the abstraction level. Language engineers 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 to create a DSL for target domain context. On the other hand, 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 (SLE) is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages [2]. Although, the phases of DSL life cycle are systematically defined [16], this process lack one crucial step, namely language evaluation just before the deployment [4].

The software industry does not seem to report investment on the evaluation of DSLs, as shown in a recent systematic literature review [11]. We argue that this is due the perceived high costs of DSL evaluation, that lack a consistent and computer-aided integration of two different and demanding complementary software language processes: development and evaluation. Language engineers should become aware of relevant quality concerns during development, and identify and apply best practices into their development plan. Evaluation experts should get better understanding of all the models involved in the DSL development in order to be able to give appropriate and reliable suggestions towards the improvement of the language under development.

The focus of this research is to propose a systematic evaluation process for DSLs with the usability concern [1]. This process is seen as a number of iterations that will enable cost control during the development process and will guarantee success of produced language from the user point of view.

3 Related work

A programming language is a model that describes the allowed terms and how to compose them into valid sentences. DSLs are generally conceived as communication interfaces between human and computers. User Interfaces (UIs) can be also seen as a realization of a language. From this perspective it was discussed in [4] that evaluating usability of DSLs is not much different from evaluating UIs.

Empirical (i.e. experimental) evaluation studies of UI with real users is a crucial phase of UI 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, often referred as Usability [13]. Meaning of usability for target domain users is a main criteria to be defined and analyzed, in order to identify which internal and external qualities will be relevant to be verified during development. These complex experimental evaluation studies are typically implemented by software evaluation experts. Their expertise is essential to properly design the evaluation sessions and to gather, interpret, and synthesize significant results. Although it is desirable to have 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 users [20].

There is an increasing awareness to the quality in use of languages, fostered by the competition of language providers. Better usability is a competitive advantage, although evaluating it remains challenging. While evaluating competing languages, it is hard to i) interpret the existing metrics in a fair, unbiased way, ii) provide reliable design changes; and iii) assure that the scope of evaluation is preserved to target user groups.

In the context of General Purpose Languages (GPLs), comparing the productivity impact of different languages during development process has some tradition [19]. Some of the common techniques is the use of popularity index[1], the cognitive dimensions framework [12], or heuristic based on the studies of cognitive effectiveness for visual syntax [17]. Some of these methods can be reused in a case when this techniques are identified relevant for usability of DSL (e.g. Moody work on cognitive effectiveness can be reused just if visual concrete syntax is given to target DSL) When usability problems are identified too late, a common approach to mitigate them is to build a tool support that minimizes their effect on users productivity [7].

4 Approach to DSL evaluation

A systematic approach based on UI experimental validation techniques should be used to assess the impact of new DSLs. For that purpose, it is necessary to merge common usability evaluation process with the DSL development process. We expect that introducing evaluation experts into DSL development will enable language engineers to build languages that are easier to use, leading to increased productivity and positive experience by their users.

In order to evaluate DSLs we need a rigorous collaborative procedure (both during and after their development), as well as evaluate their sentences (called instance models). For that it is necessary to (i) define the usability criteria to evaluate DSLs by focusing on indicators that are relevant for users perception of quality (e.g. efficiency, effectiveness, reliability,learnability, scalability, reusability ect.); (ii) integrate in an existing IDE support Quality in Use indicators; and (iii) define a methodological approach to support the evolution of a DSLs based on experience and infer its impact on quality improvement during its lifecycle (e.g. traceability of design decisions, reuse of experiment results in a domain).

An iterative user-centered approach DSL for evaluation was proposed in [4]. Goal was to establish formal correspondences between the DSL development process and the experimental evaluation. We should research the most suitable means to provide both reliable DSL evaluation metrics and iterative suggestions during DSLs development and evolution. The initial vision of the approach is detailed by a set of practices that are introduced in order to provide a complete

[1] http://lang-index.sourceforge.net/

solution to a complex problem of placing intended users as a focal point of DSLs design and conception, and by that we ensure that the language satisfies the user requirements [3]. These interdependent patterns are expected to disseminate best practices to the end users. It provides means of performing experimental validation in the most costeffective manner and is expected to give the rationale about correct and reliable indicators that should be reusable.

These patterns outline different roles relevant for development and evaluation of DSL, and are divided into three sets that correspond to a different abstraction space: (i) *Agile Development Process* includes patterns devoted to project management and engineering of a DSL. Through organization and planning of language development we are controlling evaluation activities and tracking success of the DSL. After each iteration, evaluation and development goals are scoped and budget is fixed in order to proceed to design and implementation activities that are guided by (ii) *Iterative User-Centered Design* patterns. These support engagement of evaluation expert into development process, that collects relevant informations concerning the language engineer conception of problem solution and users interpretation. Depending on artifacts to be evaluated (cognitive model instances), abstraction layers that iteration focus on (abstract or concrete syntaxes, integration environment, given semantics), evaluation methodology, metrics definition and relations used, each iteration evaluation design can vary a lot. The third set of patterns, (iii) *Experimental Evaluation Design* supports the definition of evaluation process artifacts e.g. hypothesis, tests, metrics, samples and statements [2]. This artifacts are expected to be reused in a cases when they do not present threat to validity of results.

By planning carefully the development process and organization of responsibilities and costs, the goal is to establish balanced management 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 planned well with the technical implementation of solution.

By adopting UI evaluation process to DSL development [4], it is found that a set of language and evaluation goals should be identified during the *domain engineering phase* of DSL's construction i.e. while eliciting minimum set of domain concepts. A first step is to understand and specify the context of use of DSLs and which kind of user groups it should target. Interviews or questionnaires with the DSL's intended users should be designed to capture information about working environment and baseline approach to solve problems in terms of user stories. In the *language design phase*, it is necessary to identify which quality in use attributes are impacted by internal and external quality indicators. 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 *testing phase*, the language engineer should involve (at least) evaluation expert to validate that the known quality problems and functional tests can be accepted, and to perform heuristic evaluation or quasi experiments with domain experts. When evaluation

goals seem to reach acceptable levels, end-user-based evaluation should be conducted. This experiment is executed by giving the users real problems to solve in order to cover the most important tasks related to scope of evaluation. Data about satisfaction and cognitive workload should also be evaluated subjectively through questionnaires. It is 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 can be seen in [6] and [14].

5 Preliminary work and contribution

When we consider GPLs, their users are part of population that master well mathematical and technical concepts (e.g. they are highly skilled in computations and logical reasoning). In order to develop programming solutions, GPL users (i.e. programmers) need to understand also domain concepts. On the other hand, as DSLs are meant to reduce use of computation concepts by putting focus on the domain concepts. As such, DSLs are expected to be used by the much diverse target population in different context of use [5] (e.g. experts from physics, chemistry, finance, management etc.). This means that evaluation criteria for DSLs need to be appropriate for their target users, technical environment and social and physical environment. However, the positive coincidence, when compared to deveres UI users, is that this expected user groups are relatively small with precisely defined knowledge sets.

Under the perspective of SLE, in order to experimentally evaluate a DSL, we need to know what are the criteria involved, understand the notion of quality from the relevant perspectives and understand the experimental process itself. This complex challenge was covered by general model for DSL experimental evaluation, i.e. [2], that 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 that are examples of best practices.

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. The value of contribution in this line of research is supported by the evidence of the interesting return of investment on usability evaluation for other software products [18]. 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 [15]. By allowing significant changes to correct deficiencies along the DSL development process, instead of just evaluating them in the end of it, presented user-centered design is meant to reduce development and support costs, increase sales, and reduce staff cost for employers [9]. This approach is expected to contribute to increase the external quality of the dsl and its correct construction according to the target expectations. Us-

ability is seen as an end goal, but to achieve it we need a means to trace how close we are to our goal trough development process.

6 Evaluation methodology

Our goal is to evaluate our approach with respect application of approach to academical and industrial real cases. In particular, we address following research questions:

Q1: Can our approach identify usability problems early in development cycle?

Q2: By using the proposed approach to language development, are we able to trace and validate design decisions?

Q3: By using proposed design of evaluation are we able to replicate experiments and reuse data and metrics?

Experiment planning. To evaluate first research question (Q1), a set of evaluation experiments during DSL development that use presented technique should be conducted, where the main role of author is to be involved as evaluation expert (i.e. usability engineer). Comparison to baseline approaches to solving problem is expected to give us means of better quality of use for the produced solution. By comparing solutions within same domain, and iteratively comparing the design solutions from each of iterations we are planning to find answers to our second and third research questions (Q2,Q3). By systematically comparing more DSL solutions within same domain, where each one cycle is focusing of different high level goals, we expect to have a means of reusing evaluation data, and building up statistical evidence of their validity and correlations. This will help validating our approach.

Threats to validity. In some evaluation cases it will be hard to distinguish if the success of produced DSL is to be due to our approach for usability or building a DSL by itself. However, we believe that is possible to overcome this threat by validating success of design decisions through different iterations and performing comparative analysis between them. It is needed to take a special care that for same context of experiment we use different subjects. Threat is that as there will be hard to find a lot of expert users in domain, and often we will end up by having available just novice users for new problem solving approach. However, it is possible to validate different artifacts or abstraction layers with a same user groups. While performing evaluation experiments we need to be careful that we don't compare non-related things; like trying to compare DSL with Java (e.g. we should restrict the GPL just to the constructs that are related with a target domain: query procedures of Java with a query DSL). That is the reason why we need to precisely define scope and assure that context of use can fit inside. It will be also challenging to evaluate iterative life-cycle.

7 Research status

Our analysis of target evaluation domain resulted in the conceptual models that should be involved into the methodological framework (e.g. context model, sur-

vey instruments and indicators, goal and requirements model etc.). In order to support evaluation and language design, presented patterns are illustrated by a real life case study of the usability evaluation of a DSL for the High-Energy Physics [6] by the author in a role of evaluation expert. This language was found appropriate for analysis, as it had detailed description of domain problem and development process. It was also rare example of a language that concentrates on group of users that are not programmers and it had systematic usability evaluation performed in the end.

Author is currently performing two evaluations of the approach. The first one in industrial context that involves iterative development of a DSL for the humanitarian campaign flow specification (FlowSL). This example is expected to result in the detailed definition of all artifacts that are needed to support development process. During the development process, FlowSL is being evaluated in comparison with baseline approach (hacking Ruby code), and comparative evaluations of iteration releases. In the end it is expected to be compared with BPMN with novice users, to assess learnability issues. Other experiment is run in context of summer schools on DSL development. Here the goal is to capture and validate design decisions and quality metrics between language solutions developed parallelly by different student groups.

An additional step is to conceptualize models for performing DSLs evaluation, metrics and traceability model of design changes and their impact. This support should be tailored to internal and external quality (e.g. syntactic and semantic models) and quality in use (e.g. interaction model) artifacts while using a DSL along several iterative evolution steps. It is expected to support more efficient performance of evaluation, and enable evaluator to explicitly model the process. This procedure will give us faster convergence of language development, as it enable us to monitor the impact of language evolution to the efficiency and effectiveness of practitioners using the language (and its companion toolset).

As a side effect, this evaluation work is expected to contribute to the validation of the claim that DSLs are more usable than GPLs. The impact of an evaluation process for DSLs could 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 can aid them in reaching more usable languages.

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).

References

1. A. Barišić, V. Amaral, and M. Goulão. Usability evaluation of domain-specific languages. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the Quality of Information and Communications Technology (QUATIC'2012)*, pages 342–347. IEEE, 2012.

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. A. Barišić, V. Amaral, M. Goulão, and B. Barroca. How to reach a usable DSL? moving toward a systematic evaluation. *ECEASST*, 50, 2011.
5. 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.
6. Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. Quality in use of domain-specific languages: a case study. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*, PLATEAU '11, pages 65–72, New York, NY, USA, 2011. ACM.
7. R. Bellamy, B. John, J. Richards, and J. Thomas. Using CogTool to model programming tasks. *Evaluation and Usability of Programming Languages and Tools (PLATEAU 2010)*, page 1, 2010.
8. Fred Brooks. *The Mythical Man-Month*. 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. Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: a cognitive dimensions framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.
13. International Standard Organization. Iso/iec fdis 25010:2011 systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models, March 2011.
14. 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.
15. Aaron Marcus. The roi of usability. In Bias and Mayhew, editors, *Cost-Justifying Usability*. North- Holland: Elsevier, 2004.
16. Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.
17. D.L. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, pages 756–779, 2009.
18. Jakob Nielsen and S. Gilutz. Usability return on investment. Technical report, Nielsen Norman Group, 2003.
19. Lutz Prechelt. An empirical comparison of seven programming languages. *IEEE Computer*, 33(10):23–29, 2000.
20. J. Rubin and D. Chisnell. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. Wiley-India, 2008.