Introducing Usability Concerns Early in the DSL Development Cycle: FlowSL Experience Report

Ankica Barišić[1], Vasco Amaral[1], Miguel Goulão[1], and Ademar Aguiar[2]

[1] CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Campus de Caparica, 2829-516 Caparica, Portugal
[2] Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
a.barisic@campus.fct.unl.pt, vma@fct.unl.pt, mgoul@fct.unl.pt, ademar.aguiar@fe.up.pt

Abstract. Domain-Specific Languages (DSLs) developers aim to narrow the gap between the level of abstraction used by domain users and the one provided by the DSL, in order to help taming the increased complexity of computer systems and real-world problems. The *quality in use* of a DSL is essential for its successful adoption. We illustrate how a usability evaluation process can be weaved into the development process of a concrete DSL - FlowSL - used for specifying humanitarian campaign processes lead by an international Non-Governmental Organization. FlowSL is being developed following an agile process using Model-Driven Development (MDD) tools, to cope with vague and poorly understood requirements in the beginning of the development process.

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

1 Introduction

Domain-Specific Languages (DSLs) and Models (DSMs) are used to raise the level of abstraction, while at the same time narrowing down the design space [1]. This shift of developers' focus to using domain abstractions, rather than general purpose abstractions closer to the computation world, is said to bring important productivity gains when compared to software development using general purpose languages (GPLs) [2]. As developers no longer need to make error-prone mappings from domain concepts to computation concepts, they can understand, validate, and modify the produced software, by adapting the domain-specific specifications [3]. This approach relies on the existence of appropriate DSLs, which have to be built for each particular domain. Building such languages is usually a key challenge for software language engineers. Although the phases of a typical DSL life cycle have been systematically discussed (e.g. [4, 5]), a crucial step is often kept implicit: the language evaluation.

DSLs are usually built by language developers in cooperation with domain *experts* [6]. In practice the DSL will be used by domain *users*. These domain *users*

are the real target audience for the DSL. Although domain *users* are familiar with the domain, they are not necessarily as experienced as the domain *experts* helping in the language definition. Neglecting domain *users* in the development process may lead to a DSL they are not really able to work with.

In this paper we apply action research to the development of a DSL, named FlowSL, designed to support managers in specifying and controlling the business processes supporting humanitarian campaigns. FlowSL is targeted to non-programmers. Their ability to use this language was identified as one of the highest concerns, so discovering usability issues in early development iterations, facilitated the achievement of an acceptable usability, while tracking the design decisions and their impact.

Usability has two complementary roles in design: as an *attribute that must be designed into the product*, and as the *highest level quality objective* which should be the overall objective of design [7].

This paper is organized as follows: Section 2 discusses related work; Section 3 provides a description of the evaluation approach; Section 4 discusses the language and evaluation goals and its development and evaluation plan; Section 5 discusses the lessons learned from the application of the described approach; finally, Section 6 concludes by highlighting lessons learnt and future work.

2 Related work

The need for assessing the impact of introducing a DSL in a development process has been discussed in the literature, often with a focus on the business value that DSL can bring (see, e.g. [8]). This business value often translates into productivity gains resulting from improved efficiency and accuracy in using a DSL [6], when compared to using a general-purpose baseline solution [9]. The quality in use of a DSL is, therefore, extremely important. In general, these assessments are performed with a final version of a DSL, when potential problems with the DSL are expensive to fix. A key difference in the work described in this paper is that we introduce language evaluation *early* in the DSL development process, so that problems can be found 'on-time' and fixed at a fraction of the cost it would take to fix them, if detected only in the deployment phase.

The term *quality in use* is often referred to more simply as *usability* [7], and includes dimensions such as *efficiency*, *effectiveness*, *satisfaction*, *context coverage* and *freedom of risk* (ISO 25010 2011). Usability evaluation investments have brought an interesting return on investment in software development [10]. Usability evaluation benefits span from a reduction of development and maintenance costs, to increased revenues brought by an improved effectiveness and efficiency by the product users [11].

Two important issues are *how* and *when* to assess DSL usability.

Concerning the *how*, we have argued that we can think of DSLs and their supporting editors as communication interfaces between DSL users and a computing platform, making DSL usability evaluation a special case of evaluating User Interfaces (UIs) [12] . This implies identifying the key quality criteria from

the perspective of the most relevant stakeholders, in order to instantiate an evaluation model for that particular DSL [13, 14]. These criteria are the evaluation goals, for which a set of relevant quantitative and qualitative measurements must be identified and collected. We borrow from UI evaluation several practices, including obtaining these measurements by observing, or interviewing, users [15]. In general, it is crucial that the evaluation of human-computer interactions includes real users [16], for the sake of its validity. In the context of DSLs, the "real users" are the domain users.

Concerning the *when*, we argued that we should adopt a systematic approach to obtain a timely frequent usability feedback, while developing the DSL, to better monitor its impact [17]. This implies the integration of two complementary processes: language development and evaluation. Software language engineers should be aware of usability concerns during language development, in order to minimize rework caused by unforeseen DSL usability shortcomings. In turn, usability designers should have enough understanding of the DSMs involved in software language development to be able to properly design the evaluation sessions, gather, interpret, and synthesize meaningful results that can help language developers improving the DSL in a timely way. This requirement is in line with agile practices, making them a good fit for this combined DSL building (i.e.software development) and evaluation process (i.e. usability design) [18].

3 Building usability into a DSL development process

Building a DSL may have a rather exploratory nature, with respect to the DSL requirements, particularly when the DSL is aimed for users with limited computational skills or poorly understood, or evolving domains. To build up a cost-effective and high quality process, we defined an agile and user centered DSL evaluation process [17, 13].

By placing DSL users as a focal point of DSLs' design and conception, the goal was to ensure that the language satisfies the user expectations. Besides involving Domain Experts and Language Engineers, as typically happens in the development of a DSL, we add the role of the Usability Engineer to the development team. Usability engineers are professionals skilled in assessing and making usability recommendations upon a given product (in this case, the DSL) and gathering unbiased systematic feedback from stakeholders [18].

Each development iteration focuses on a different increment or level of abstraction to be evaluated or refined. In the early phases it is important to study existing guidelines or standards for a particular domain and interview current or potential users about their current system or tools they are using to help them in accomplishing their tasks. This *context of use* study of a particular situation is intended to elicit the strengths and weaknesses of the baseline approach as well as the user expectations for the DSL.

Finally, once the language is deployed to users, an evaluation of its use in real contexts should be conducted, reusing the methods and metrics that were validated in the previous iterations.

4 Flow Specification Language (FlowSL)

The generic process described in the previous section was instantiated to the development of a concrete DSL — the FlowSL. FlowSL is a DSL for specifying humanitarian campaigns to be conducted by a non-governmental organization. FlowSL is integrated in MOVERCADO[3] (MVC), a mobile-based messaging platform at the core of an ecosystem that enables real-time and a more efficient impact, by facilitating interactions among beneficiaries, health workers and facilities, e-money and mobile operators. The platform is meant to allow data mining in the search of insights that can be used to improve the effects of the campaigns while supporting a high degree of transparency and accountability.

A first version of the system (MVC1) was developed as a proof-of-concept to validate the key underlying principles. The second version of the system (MVC2) was developed in the form of a platform easily customizable by managers and extensible by developers of the organization's team. An important goal was to develop a language, FlowSL, to empower the Campaign Managers to define new kinds of campaign flows taking advantage of their domain knowledge.

Without FlowSL, managers needed to specify the flows orchestrating their campaigns exclusively by means of presentations and verbal explanations. The implementation and evolution of campaigns often resulted in rework and unexpected behavior, usually due to vague specifications, incorrect interpretations, and difficulties in validating the implementation, a phenomenon known as *impedance mismatch* [19]. Therefore, the primary goal was to evolve the system to enable new users to easily create new campaigns and underlying flows. FlowSL is expected to enable the organization to streamline the process of defining campaigns and their base workflows, namely participants, activities, interaction logic, and messages.

4.1 FlowSL development process

In order to balance the development effort with effective reusability (e.g. while envisioning new marketing solutions), MVC2 was developed in a fast-paced way, iteratively, along six two-weeks sprints, following an agile development process based on Scrum[4] and best practices of evolving reusable software systems [20]. In the process of FlowSL development, the Domain Experts were part of the Product Owners team, while the Language Engineers were part of the Scrum Team. The DSL evaluation process was guided by the FlowSL development stages, as different effort was estimated in each sprint for its development.

The problem analysis was performed by mutual interaction and brainstorming between Domain Experts and Language Engineers in each sprint planning. Usability Engineers, in this case the researchers, had the role of observing and guiding the analysis outputs, while preparing the evaluation plan, without being directly involved in the language specification. To better understand and define

[3] http://enter.movercado.org/ (accessed in July 19, 2014)
[4] http://www.scrum.org/ (accessed in July 18, 2014)

the problem, the required functionalities were described in terms of small user stories. Also, the new description of the user roles was introduced as the FlowSL is expected to change existing organizational workflows. To improve interaction between the development team and the users, all the produced results from the analysis were continuously documented in a wiki. As Scrum suggests, the project management was based on a product backlog maintained and shared on-line.

The relationship between the MVC system, FlowSL development, and relevant language users and expected workflow is presented in Fig.1. The original MVC1 system was developed in a GPL (Ruby). FlowSL was first developed as a Ruby-based internal DSL. This approach allowed an optimal use of resources while keeping the existing system running. The second phase of language development was intended to support the managers to design the campaign flow specifications by themselves, using simple and understandable visual language constructs. In the planned third phase (future work), the focus will be on evolving the language's editor to be collaborative and web-based. It will also be an opportunity to work on language's optimizations in the generation process.

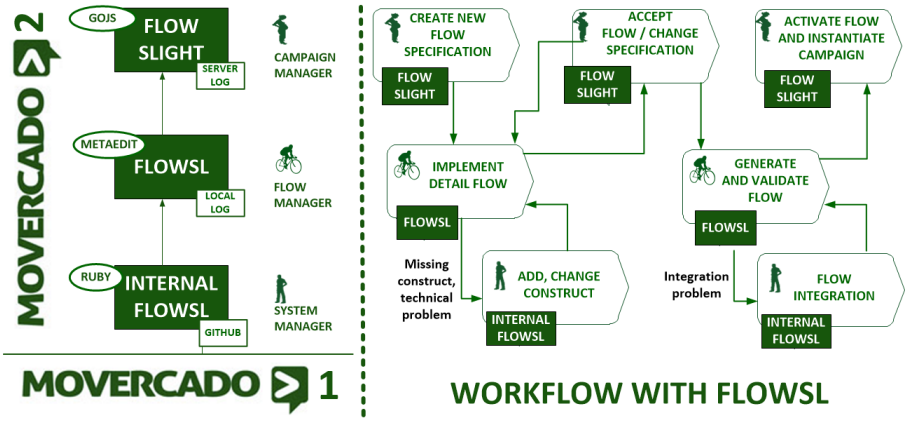

Fig. 1. FlowSL development and relevant language users with expected workflow

After defining the evaluation plan, the Usability Engineer prepared the usability requirements, using a goal-question-metric approach presented in Table 1, where goals conform to the Quality in Use model. These requirements were detailed and related to the right metrics and measurement instruments to perform appropriate usability tests in each development cycle. The validation of some of these requirements in earlier stages (e.g. understandability, readability) are stepping stones to achieve other soft requirements that cannot be evaluated in early phases (e.g. learnability). Multiple evaluations helped in validating and improving the set of identified metrics.

Table 1. Usability requirements

Requirement	*Metric*
Understandability: Does the user understand the different concepts and relations, and when and why to use each one of the concepts?	NCon - number of concepts, NRel - number of relationships NErrSpec - incorrect verbal definitions of total NCon and Nrel given in language NErrMod - incorrect interpretations of presented NCon and NRel given in modeled solution
Readability: How accurately is the user able to read the specified flows and interpret their meaning?	NConInst - number of concept instances in the model (flow), NReInst - number of relationship instances in the model NErrInst - number of incorrect verbal interpretation of NConInst and NRelInst given in language
Efficiency: How much time is needed for a user to read existing or specify a new flow?	TModInst - time necessary to read existing model instance (flow) TModSpec - time necessary to implement a new model instance (flow)
Effectiveness: Is the user able to correctly implement a flow from a given high-level description of the required flow?	NErrModInst - number of misinterpretation while reading existing model instance (flow) NErrModSpec - number of errors while implementing new model instance (flow)
Learnability: How much time is needed for users to learn the FlowSL language?	TLearNov - training time necessary to learn novice users to use language TLearExp - training time necessary to learn domain experts to use language
Flexibility How long does it take to quickly change or correct existing flow specifications?	TModEvol - time necessary to evolve model instance (flow) TModCorr - time necessary to correct incorrect implementation of model instance (flow)
Reusability How often user reuse existing flow specifications?	NModReuse - number of reusing existing model instance (flow) NModEvol - number of evolving existing model instance (flow)
Expressiveness Is the user able to specify all parts of flow?	NErrCon - number of concept, or its property that user is missing to implement model instance (flow) NErrRel - number of relationships, or its appropriate role that user is missing to implement model instance (flow)
Freadom of Risk Is the user able to implement the specifications in a way that can lead to unexpected or unwanted system behavior?	NEconDem - number of occurrence of economic damage due to incorrect flow specification NSofCor - number of occurrence of software corruption due to incorrect flow generation to system (flow)
Satisfaction How much is the user satisfied with FlowSL?	ConfLevel - self rated confidence score in a Likert scale LikeLevel - self rated likability score in a Likert scale

5 FlowSL evaluation and lessons learned

5.1 First FlowSL iteration: bottom-up approach (MVC2.1)

The *language goal* of the first iteration was to find the differences and commonalities in the Ruby code relevant for visual FlowSL and then do a corresponding mapping into a graphical representation, which would define the first draft of the concrete visual syntax of FlowSL. This is considered as a way to describe appropriate activities step by step by mapping relevant fragments of extracted code to a visual representation and to identify repetitive patterns that represent

reusable code artifacts. The *evaluation goal* was to assess whether this representation would be good enough to enhance the *understandability* and *readability* of flows from the perspective of Campaign Managers. It was expected that with the flow abstraction, the Domain Experts could describe more concrete requirements for the visual flow concepts.

The *evaluation intervention* was conducted when all existing flows of the MVC1 system were migrated to MVC2. This was the moment when the stakeholders could more clearly express the language purpose by distinguishing campaign processes from the flows underneath. The intervention was followed by an interview conducted with one representative *subject*: the Domain Expert with the role of Campaign Manager that was involved in specifying flows using the MVC1 system and who was also involved in the MVC2 Scrum development assuming, in that case, the role of Product Owner.

The *evaluation document* was prepared by the Usability Engineer containing 4 tasks: *Task 1* and *Task 2* describing user scenarios by roles and a global organization scenario that evaluator was asked to clarify and improve by placing him in organization workflow; *Task 3* presenting alternative feature models of FlowSL that are reviewed and redefined with a goal of separating campaign instantiation data and improving a vague language definition; *Task 4* presenting campaign flow based on simple and complex level of specification of the flow example (*IPC Validation*) that was found to be the most representative to describe. This task used metrics from the GQM table, which showed that the considered solution is very hard to understand.

The two major threats to validity of this evaluation were that it was subjective and only one user surrogate was involved. However, as the intended solution was seen as a step that helped to understand and to model the domain better, the guided interview helped to redefine the technical concepts using domain terms. Evaluation resulted in a clearer plan for the next development cycles as well as clarifying usability requirements and appropriate tasks. The textual FlowSL makes explicit all relevant domain concepts, but also many extra more. considered more technical, The performed evaluation helped the DSL developers to adjust the level of abstraction to the needs of the DSL end users. The language at this phase, could be used by the System Managers (knowledgeable of the concepts of the baseline system), but not by Campaign Managers.

5.2 Second FlowSL iteration: top-down approach (MVC 2.2)

The *language goal* of this iteration was to develop a visual FlowSL prototype using the MetaEdit[5] language workbench, that was selected for its support to top-down development. The *evaluation's goal* was to assess whether both the campaign managers and novice system managers were able to validate the specified flows using the newly proposed visual language and editor. These evaluations covered also the effectiveness and expressiveness of the target language.

[5] http://www.metacase.com/ (accessed in July 19, 2014)

The *First evaluation intervention* was organized very quickly and involved interviewing two *subjects*: the campaign manager from the first development iteration and the system manager who was involved in the DSL development. The intervention consisted of one task where the subjects had the opportunity to compare two alternative concrete flow representations for the same ongoing example.

Based on the evaluation results the Usability Engineer produced designs of the concrete syntax for the DSL development team.

The *second evaluation intervention* involved the same subjects. The *evaluation document* had three tasks: Task 1 focused in assessing the *understandability* and *expressiveness* of the individual symbols; Tasks 2 and Task 3 meant to measure the *readability* and *efficiency* of the designed solution of the simple and complex flow. In addition to that, the Domain Expert was asked to describe the use of the symbols from Task 1 to produce the presented flow solutions and to describe the situations in which the existing flows can be reused. The evaluation session with the System Manager made it possible to identify important missing relationships between FlowSL concepts, as well as their connection points (hot spots) with the MVC system underneath.

For the *third evaluation intervention* the usability engineer introduced the design improvements motivated by the feedback obtained the previous evaluation. The new notations were designed and implemented, to be again compared. The tasks were similar to the previous intervention, although more elaborated. Here, the same subjects from the previous interventions were involved, as well as a member of the Scrum team.

For this third intervention the rules related to the usage of a certain activity were discussed. The usability engineer evaluated the cases where the system manager would have the need to hack the existing campaign flows, in order to customize certain functionality or rule. The goal was to use an example-based approach to identify improvements in the language.

It became clear that the evaluation materials prepared earlier helped to speed up the following evaluation phases and reduced their implementation costs. Besides, they became templates for the corresponding learning materials. Also, it was possible to abstract the language one level further, so that an online visual editor was built to support rapid high level specifications of flows. To better deal with the increasing complexity of the specified models, rather than presenting all the concepts related to the flow definition visually, a better option would be to present just high level concepts that are reused often, while others are hidden and based on predefined rules that can be eventually reconfigured textually. This approach empowered both the domain experts and the product owners to better control the design decisions.

6 Conclusions and future work

In this paper, we presented an experience report on how to integrate top-down usability engineering practices into a bottom-up agile development of a DSL

from its beginning. While playing the role of Usability Engineers, we experienced that small iterations involving Domain Experts, Product Owners and End Users can help us to clarify the meaning and the definition of the relevant language concepts. This enables an early identification of possible language usability shortcomings and helps reshaping the DSL accordingly.

Early evaluations can be executed with a relatively low cost thanks to model-driven tools that support production of rapid prototypes and presenting the idea. These evaluations support well-informed trade-offs among the strategy and design of the DSL under development, and its technical implementation, by improving communication. Besides, they improve the traceability of decisions, and of the solution progress. These iterations also help to capture and clarify contractual details of the most relevant language aspects that need to be considered during DSL development, and are a key element to improve the End Users experience while working with FlowSL.

We plan to validate our decisions, metrics, and the overall merit of the developed DSL, by performing experimental evaluations with both expert and novice users, by making comparisons to the baseline approach in Ruby, as well as to other process modelling languages that are natural candidates to serve for similar purposes (e.g. BPMN, JWL).

An additional step is to conceptualize the traceability model of design changes and evaluate its impact on the decision making process. We expect that in each iterative evaluation step we will not only identify opportunities to improve the usability of the DSL, but also to improve the evaluation process itself (e.g. through the validation, in this context, of the chosen metrics).

Weaving usability concerns into agile process is helping us to continuously evolve FlowSL, improving the cost-effectiveness of DSL usage in specifying campaigns, and supporting a clearer assessment of which language concepts are more relevant to the different kinds of language users, which in turn helps finding the right level of abstraction and granularity of concepts. All these benefits come with the cost of adding usability skills and of introducing new practices in the agile process, namely the introduction of lightweight metamodeling tools. The balance however, seems to be very positive, but ROI should be calculated precisely to support this claim.

References

1. Gray, J., Rossi, M., Tolvanen, J.P.: Preface. Journal of Visual Languages and Computing, Elsevier **15** (2004) 207–209
2. Kelly, S., Tolvanen, J.P.: Visual domain-specific modelling: benefits and experiences of using metacase tools. In Bézivin, J., Ernst, J., eds.: International Workshop on Model Engineering, at ECOOP'2000. (2000)
3. Deursen, A.V., Klint, P.: Little languages: Little maintenance? Journal of Software Maintenance: Research and Practice **10**(2) (1998) 75–92
4. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys **37**(4) (2005) 316–344

5. Visser, E.: WebDSL: A case study in domain-specific language engineering. In Generative and Transformational Techniques in Software Engineering II, Ralf Lämmel, Joost Visser, and João Saraiva (Eds.). Lecture Notes In Computer Science **5235** (2007)
6. Voelter, M., Dietrich, C., Engelmann, B., Helander, M., Kats, L., Visser, E., Wachsmuth: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform (2013)
7. Petrie, H., Bevan, N.: The evaluation of accessibility, usability and user experience. Human Factors and Ergonomics. CRC Press (2009)
8. Kelly, S., Tolvanen, J.P.: Domain-specific modeling: enabling full code generation. John Wiley & Sons (2008)
9. Kosar, T., Mernik, M., Carver, J.: Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical Software Engineering **17**(3) (2012) 276–304
10. Nielsen, J., Gilutz, S.: Usability return on investment. Technical report, Nielsen Norman Group (2003)
11. Marcus, A.: The ROI of usability. In Bias, Mayhew, eds.: Cost-Justifying Usability. North- Holland: Elsevier (2004)
12. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: 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, New York, NY, USA, ACM (2011) 65–72
13. Barišić, A., Monteiro, P., Amaral, V., Goulão, M., Monteiro, M.: Patterns for evaluating usability of domain-specific languages. Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP), SPLASH 2012 (October 2012)
14. Kahraman, G., Bilgen, S.: A framework for qualitative assessment of domain-specific languages. Software & Systems Modeling (2013) 1–22
15. Rubin, J., Chisnell, D.: Handbook of Usability Testing: How to plan, design and conduct effective tests. Wiley-India (2008)
16. Dix, A.: Human computer interaction. Pearson Education (2004)
17. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: How to reach a usable DSL? moving toward a systematic evaluation. Electronic Communications of the EASST **50** (2011)
18. Lárusdóttir, M., Cajander, Å., Gulliksen, J.: Informal feedback rather than performance measurements–user-centred evaluation in scrum projects. Behaviour & Information Technology (ahead-of-print) (2013) 1–18
19. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information systems **27**(6) (2002) 365–389
20. Roberts, D., Johnson, R.: Evolving frameworks: A pattern language for developing object-oriented frameworks. In: Proceedings of the Third Conference on Pattern Languages and Programming, Addison-Wesley (1996)