Do Software Languages Engineers Evaluate their Languages?

Pedro Gabriel, Miguel Goulão, and Vasco Amaral

CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
pedrohgabriel@gmail.com,{miguel.goulao,vasco.amaral}@di.fct.unl.pt
http://citi.di.fct.unl.pt/

Abstract. Domain Specific Languages (DSLs) can contribute to increment productivity, while reducing the required maintenance and programming expertise. We hypothesize that Software Languages Engineering (SLE) developers consistently skip, or relax, Language Evaluation. Based on the experience of engineering other types of software products, we assume that this may potentially lead to the deployment of inadequate languages. The fact that the languages already deal with concepts from the problem domain, and not the solution domain, is not enough to validate several issues at stake, such as its expressiveness, usability, effectiveness, maintainability, or even the domain expert's productivity while using them. We present a systematic review on articles published in top ranked venues, from 2001 to 2008, which report DSLs' construction, to characterize the common practice. This work confirms our initial hypothesis and lays the ground for the discussion on how to include a systematic approach to DSL evaluation in the SLE process.

Key words: Domain Specific Languages, Systematic Review, Usability Evaluation, Experimental Software Engineering, Language Engineering Process Model

1 Introduction

Domain-driven development is an approach to software development which relies on Domain Specific Languages (DSLs) and Models (DSMs) to raise the level of abstraction, while at the same time narrowing down the design space [1]. Among other claims, this shift of developers' focus to using abstractions that are part of the domain world, rather than general purpose abstractions closer to the design and code world, is said to bring important productivity gains, increased time-to-market responsiveness, and smaller training time, when compared to software development using general purpose design and programming languages [2]. The rationale is that developers no longer need to make error-prone mappings from domain concepts to design concepts, and onto programming language concepts. Instead, they can work directly with domain concepts. As such, domain experts 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 a key challenge for software language engineers.

Software Languages Engineering (SLE) is becoming a mature and systematic activity, building upon the collective experience of a growing community, and the increasing availability of supporting tools. A typical SLE process starts with the Domain Engineering phase, in order to elicit the domain concepts. The following step is to design the language, capturing the referred concepts and their relationships. Then, the language is implemented, typically using workbench tools such as MetaEdit [4], MetaEdit+ [5], GMF/EMF [6], GME [7] or Microsoft DSL Tools [8], and documented. A typical development process goes on to the testing, deployment, evolution, recovery, and retirement of languages. However streamlined the process is becoming, it still presents a serious gap in what should be a crucial phase: the Language Evaluation, or testing phase.

As with any other software product, we need to assure that the DSL is adequate to the end user (the Domain Expert). This covers not only the language's correctness, but also quality attributes, such as the language's usability, the maintainability of the produced systems, or the productivity of the developers using the DSL. A good DSL is hard to build because, as noted by Mernik et al. [9], it requires both domain knowledge and language development expertise, and few people have both. We should assert claims like that the newly designed language brings efficiency to the process, or that it is usable and effective, with an unbiased evaluation process. Furthermore, we should be able to quantify the extent to which DSL's introduction brings economic benefits to organizations. Intuitively, we expect to observe a positive impact on the maintainability of systems specified with DSLs, when compared to those specified with general-purpose languages. We can find anecdotal reports of 3-10 times productivity improvements (e.g. [2, 10, 11]) or *"clearly boosted development speeds"* [12]. Unfortunately, it is not possible to make a fair meta-analysis on these claims. They are too vague and supported by testimonials by project managers rather than by detailed data that can be independently verified and used. What is a typical improvement, in quantifiable terms, particularly considering that we must take into account the development and maintainability of the DSL, and its set of supporting tools, as well? What is the impact of using several DSLs in an organization, each with its own maintainability cost attached? What is the break-even point from which it pays off to use a DSL? These questions are relevant, particularly because we find references to studies comparing alternative approaches to DSL construction with a major concern on the costs involved [13, 14]. In short, when should we choose to develop and then adopt a DSL? We are able to find qualitative answers to this question [9], but not quantitative-based ones (although Weiss and Lai do propose a formula for computing that break-even point [10]).

There are a few exceptions to this state of practice. Batory *et al.* report on extensibility and maintainability improvements [15] brought by a combination of DSLs and Software Product Lines (SPLs), although it is unclear which share of the merit should be attributed to DSLs and which should be credited to SPLs.

This is a potential confounding factor which can also be identified in most of the improvement claims concerning DSLs, as they are often used in combination with SPLs. Kieburtz *et al.* report a series of experiments comparing favorably the usage of DSLs *vs.* templates in code generation, with respect to flexibility, productivity, reliability and usability [16]. Hermans *et al.* conducted a case study where the reliability, development costs were improved with the introduction of DSLs [17]. In the context of general purpose languages, we can find experimental comparative analysis of the productivity achieved by practitioners using different languages (e.g. [18]).

The software industry, in general, does not seem to invest much on the evaluation of DSLs. Among other possible explanations, this state of practice may stem from a lack of enough software experts that completely understand the SLE process, or from a lack of experimental evidence that clearly backs up the qualitative improvement claims that we often find in the literature. Without such evidence, it may be the case that decision makers consider proper language evaluation as a waste of time and resources. If so, they may prefer to risk using or selling inadequate DSLs rather than evaluating them properly. The incremental nature of a typical DSL life cycle may also give the erroneous feeling that the language is being implicitly validated due to the intense interaction with the domain experts. The problem there is that the domain experts involved in the language development may not be the end users, and may therefore introduce biases in the perception of the language design and usability.

In this paper, we will assess whether or not we can find evidence in the literature to back up our hypothesis: in general, software language engineers do not evaluate their languages with respect to their impact in the software development process in which the DSLs will be integrated. To the best of our knowledge, there is no available systematic review and meta-analysis on the level of evaluation of DSLs reported in the literature. The review presented in this paper aims to fill in this gap. Ultimately, we aim to raise the community's awareness to the problem of poor validation of DSLs, and its impact on our ability to support the "Engineering" title in Software Languages Engineering. This paper reports on a survey that quantitatively characterizes the description of experimental validation of DSLs in papers published in 15 of the most important scientific venues covering this research area, from 2001 to 2008.

This paper is organized as follows. In section 2, we present the research protocol followed in this systematic review of the current state of practice in SLE. In section 3, we discuss the inclusion and exclusion criteria in this review. In section 4, we present the main findings of our review. In section 5, we discuss those findings, the strengths, and weaknesses of the evidence collected for this review and their generalizability to the current state of practice. In section 6, we summarize the main conclusions and their implications to the SLE community.

2 Review questions

Our main motivation was to determine the extent to which the DSL community had presented evidence of its commitment to usability experimentation, in the context of proposals of new DSLs. In order to guide our systematic review on the state of practice, we start by stating our research questions:

- Is there a concrete and detailed evaluation model to measure DSLs Usability?
- Is the DSL community concerned about experimental evaluation as a mechanism to prevent future problems emerging from the proposed DSLs?
- To what extent does the DSL community present evidence that the developed DSLs are easy to use and correspond to end-users needs?

In order to facilitate an objective and consistent view on each of the inspected papers, we broke these questions into more detailed criteria that we then used to classify the surveyed papers. These more detailed questions were:

- **RQ1:** Does the paper report the development of a DSL?
- **RQ2:** Does the paper report the DSL development process with some detail?
- **RQ3:** Does the paper report any experimentation conducted for the assessment of the DSL?
- **RQ4:** Does the paper report the inclusion of end-users in the assessment of a DSL?
- **RQ5:** Does the paper report any sort of usability evaluation?

3 Review methods

This paper reports on a survey that quantitatively characterizes the description of experimental validation of DSLs in papers published in 15 of the most important scientific publications covering this research area, from 2001 to 2008.

The selected publications include: 1 special issue of a journal (*Journal of Visual Languages and Computing (JVLC)*), 2 conferences (*International Conference on Software Language Engineering (SLE)* and *International Conference on Model Driven Engineering Languages and Systems (MODELS)*), and 10 workshop series focused in domain driven development and languages engineering, namely: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, O*OPSLA Workshop on Domain-Specific Modeling (DSM)*, *OOPSLA Workshop on Domain-Specific Visual Languages (DSVL)*, *ECOOP Workshop on Domain-Specific Program Development (DSPD)*, *International Workshop on Language Engineering (ATEM)*, *Model-Driven Development Tool Implementers Forum (MDD-TIF)*, *Modellierung DSML (DSML)*, *International Workshop on Software Factories at OOPSLA (OOPSLA-SF)*, *ECOOP Workshop on Evolution and Reuse of Language Specifications for DSLs (ERLS)*, *ETAPS Workshop on Language Descriptions*, *Tools and Applications (LDTA)*. The survey also covers 2 general Software Engineering publications, namely *IEEE Transactions on Software Engineering (TSE)* and the *International Conference in Software Engineering (ICSE)* conference series.

Paper selection was performed in two steps: (i) a direct inspection of paper abstracts and conclusions, to identify papers covering our research questions, followed by (ii) a full review of selected papers, to answer our research questions. This process was followed in 10 publications. The exceptions were Models, VL/HCC, LDTA, ICSE and TSE. In these venues, we started by using a web search mechanism to identify good candidates for further review, and then followed steps (i) and (ii) for those papers. The search keywords were "domain-specific language", "domain-specific modeling", "DSL", and "DSM".

Table 1 presents an overview of the selected papers. We grouped the publications in two categories, corresponding to the two paper selection strategies identified earlier Each table row presents the publication name, the number of available papers in that publication, from 2001 to 2008, the number of inspected papers, the number of selected papers and their percentage with respect to the number of inspected papers.

Table 1. Selected papers

Selection	**Publication**	**Available articles**	**Inspected articles**	**Selected articles**	**Selection Percentage**
Direct	OOPSLA-DSM	97	97	14	14.4%
	OOPSLA-DSVL	27	27	5	18.5%
	DSPD	19	19	3	15.8%
	SLE	18	18	0	0.0%
	ATEM	13	13	2	15.4%
	MDD-TIF	10	10	3	30.0%
	DSML	12	10	0	0.0%
	OOPSLA-SF	9	9	0	0.0%
	ECOOP-ERLS	6	6	0	0.0%
	JVLC	5	5	2	40.0%
Query	VL/HCC	141	16	2	12.5%
	LDTA	10	2	1	50.0%
	MODELS	200	4	1	25.0%
	ICSE	42	6	2	33.3%
	TSE	32	2	1	50.0%
	Total	641	242	36	14.6%

Although we have used a common time frame for all publications, several of these publications were only available in some of the years under scrutiny. This diversity of number of editions of the publications explains the variability of the number of scrutinized papers, with respect to their origin. Nevertheless, we believe that our sample is representative of the current state of practice in DSL development.

The large majority of selected papers were published in workshops. This distribution can be regarded as an indicator of the relative novelty of this research area.Furthermore, our survey targeted publications with a strong concentration on discussions on DSLs, and their creation. However, the development of DSLs crosscuts the whole software industry. Therefore, it is fair to assume that many DSLs that are developed in this community are disseminated in journals, conferences, and workshops targeted to the domains they address.

4 Included and excluded studies

As discussed in section 3, the selection of eligible papers followed a two-step protocol. We started by a pre-selection, which was then followed by an in-depth analysis of each of the reviewed papers. We were conservative in the first step: when in doubt, papers went on to full review.

During the second step, we selected papers which would help answering each of our research questions. To facilitate paper selection, we defined strict paper inclusion criteria, namely: (i) the paper reported on the development of at least one DSL; (ii) the paper reported on the experimental evaluation of DSLs; or (iii) the paper reported on specific techniques of DSLs Usability evaluation.

Criterion (i) helped establishing a baseline for developed DSLs which could be, or not, assessed through some sort of experimental evaluation. Criterion (ii) covered such evaluation. We also used criterion (iii) in our search, but had no success while applying it. This criterion was targeted at finding out trends with respect to specific techniques for usability testing that suit well the DSL community.

We excluded from our survey any papers not covering any of these issues. For instance, SLE includes, among other subjects, research on development frameworks for DSLs, generative programming, and model transformations to cope with the evolution of DSLs and its impact on the evolution of software built upon an evolving DSL. While these subjects are essential to the SLE research area, they are out of the scope of our survey.

A total of 36 papers were finally selected [19–54].

5 Results

In this section, we report the obtained results, for each of our research questions.

RQ1: Does the paper report the development of a DSL? It is important to understand which of the selected papers do report on the development of a DSL, so that we can later compute the percentage of such reports which include information on DSL evaluation.

Table 2 summarizes the relative weight of papers reporting DSL development among the selected papers. A considerable percentage of the total selected papers (91.7%) reports the development of a DSL. Some of these DSLs were developed to satisfy a specific demand in the real world, while others were presented as a proof of concept, targeted to improve a specific domain in software production (e.g. a DSL for Interactive Television applications [19, 36, 42], or a DSL for interoperability between object-oriented and mainframe systems [48]). In contrast, the 3 papers which do not report DSL development were selected for this survey because they covered topics which were relevant to it, namely the usage of quantitative analysis to assess domain-specific modelling techniques [22], a survey on experiences while developing DSMLs [37], and a paper which refers to the usage of usability techniques for the assessment of DSLs [31].

Table 2. Papers reporting DSL development

DSL development	N	%
Paper reports DSL development	33	91.7%
Paper does not report DSL development	3	8.3%

RQ2: Does the paper report the DSL development process with some detail? This question aims to help us characterizing the extent to which authors provide details about the DSLs whose development was detailed in the papers. This information is relevant to our context, as an extra element for comparison. So, for each paper, we looked for details on the DSL construction. Out of the 33 papers reporting a DSL development, 16 provide some in-depth details on how those DSLs are built. The presence of a metamodel was not imperative, for this classification, but in some cases it proved to be a good help explaining the developed DSL. The distribution of papers reporting DSL construction details over the years is presented in table 3.

Table 3. Papers reporting DSL development details

Year	DSLs	Detailed Descriptions	Percentage
2008	9	4	44%
2007	10	7	70%
2006	7	2	29%
2005	1	1	100%
2004	3	2	67%
2003	1	0	0%
2002	3	0	0%
2001	2	0	0%
Totals	36	16	44%

Among the 3 papers which do not report the development of a DSL, two of them [37, 31] also discuss in some detail how DSLs are built. Overall, 16 out of the 36 selected papers provide this sort of details, and we can observe that the vast majority of the papers providing these details were published in the most recent half of the time frame considered in this survey.

RQ3: Does the paper report any experimentation conducted for the assessment of the DSL? As we have seen so far, 33 of our selected papers report the development of a DSL. The next step is to find out how many of these report having performed any sort of experimental evaluation of the developed DSLs, and to characterize the reported experimentation. To achieve this, we will use two orthogonal categorizations of experimental validation.

The first one is used to identify if the experimental validation reported by authors is of a **quantitative**, or a **qualitative** nature. A Quantitative Method is based on the evaluation of measurable property (or properties) from real data, with the aim of supporting or refuting an hypothesis raised by the experimenter. Qualitative Methods focus on qualitative data obtained through observation, interviews, questionnaires, and so on, from a specific population. The data is then cataloged in such way that it can be useful to infer to other situations. In contrast with Quantitative Methods, no kind of measurable evaluation is performed. In spite of this apparent fragility, in many cases, qualitative methods may help to explain the reasons for some relationships and results, which otherwise would not be well understood [55]. As some of the papers claim that some sort of experimental validation is performed, but do not provide enough information for the reader to know which kind of evaluation was performed, we add a third category, labeled as **unknown**. Finally, some papers report no experimental validation at all. Table 4 summarizes our findings, using this classification.

Table 4. Quantitative vs. Qualitative experimentation

Experimentation kind	**N**	**Percentage**
Quantitative	3	8.3%
Qualitative	2	5.6%
Unknown	21	58.3%
Without Experimentation	9	27.8%

The first noticeable information is that only five of the papers are explicit about using a particular kind of experimental validation of DSLs. Two of the papers reporting on using quantitative experimentation do so within the scope of the description of a particular DSL. Zeng et al. report on comparison between the number of Lines of Code (LOC) generated by a DSL and the LOC when using a "traditional" approach to software development [54]. Merilinna et al. [39] also use a LOC-based comparison, among three development alternatives: "traditional software development", UML-based software development, and DSL-based software development. They use an atomic model element to measure the software production effort. The third paper addressing experimental evaluation makes a comparison between two different approaches: the traditional software implementation versus using a DSML in an effort to show DSMLs' benefits [22]. None of these papers presents data in such a manner that it could be reused in other experiments.

One of the papers [25] reports the usage of a qualitative approach to the assessment of DSM techniques targeted to the definition and improvement of software process models within a software development company. The data presented in this study is also not well suited to facilitate a meta-analysis. The other paper covering qualitative assessment [37], reports a 20 industrial project research using DSMs and MetaEdit+. The qualitative data was collected through diverse means: interviews and discussions with consultants or in-house developers who created the DSMLs, with domain engineers, responsible personnel for the architectural solution and tool support.

In contrast, 10 papers do not report the implementation of any kind of experimental evaluation of DSLs [20, 21, 28, 30, 31, 33, 43, 44, 51, 52].

The remaining 21 papers share two common features: (i) they do not provide information concerning whether the work they describe involved any sort of experimental validation, but (ii) they do provide some information concerning the kinds of examples used in such validation. 20 of these papers use ad-hoc, or toy examples [19, 23, 24, 26, 27, 29, 32, 34–36, 38, 40–42, 45–50, 53], while 1 claims to have obtained their information at an industrial level [37], but does not provide details on the particular evaluation. The 4 papers reporting either a qualitative or a quantitative validation provide no details concerning the kind of examples used in that validation, with respect to whether they are toy or industrial-level examples. Table 5 summarizes this information.

Table 5. Toy vs. industrial examples usage

Experimental material kind	**N**	**Percentage**
Ad-hoc/Toy example	21	58.3%
Industrial level	1	2.8%
Unknown	4	11.1%
Without Experimentation	10	27.8%

RQ4: Does the paper report the inclusion of end-users in the assessment of a DSL? When developing a new system, regardless of it being completely new or a new system version, it is important to know its' the intended users profile, and how they will use the system. Their impact on usability is enormous, so an early definition of their capabilities allows developers to understand what is important and what is disposable, reducing the number of redundant or unnecessary features in the system [56]. This observation is applicable to software users in general, and to DSL users in particular. In order to characterize the DSL users who participate in a DSL evaluation, we define three categories: (i)**Industrial or specialized personnel** - we use this classification for papers reporting subjects with expertise in the domain. The domain expert does not necessarily need to have knowledge about DSLs, in general; (ii) **Academic** - the typical example is the usage of graduate students as surrogates for the real end-users of a DSL; (iii) **Not defined** - we use this category whenever we are not able to find the user profile in the paper.

3 papers reported using domain experts, including seismologists [46] and other specialized developers [25, 37]. The remaining 2 papers [33, 54] did not specify the type of subjects involved in the evaluation of the DSL.

Table 6. Domain experts usage

Domain experts usage	N	Percentage
Industrial or specialized personnel	3	8.3%
Academic	0	0.0%
Not defined	2	5.6%
Unknown	31	86.1%

RQ5: Does the paper report any sort of usability evaluation? Usability is a quality attribute based on users' and/or stakeholders' needs satisfaction by assessing how easy a system is to use. We assess the extent to which DSLs were tested for usability, and whether they fulfill the end user needs and identify three categories for this: (i) **Usability Techniques** - the papers report a set of techniques that allow DSLs becoming more accurate to the end users; (ii) **Ad-hoc** - the paper reports an ad-hoc approach to improving DSLs' usage without a detailed rationale; (iii) **No usability evaluation** - the paper provides no usability evaluation. Table 7 summarizes this information.

Table 7. Reported Usability techniques in DSL validation

Usability Approach	**N**	**Percentage**
Usability techniques	1	2.8%
Ad-hoc	6	16.6%
No usability evaluation	29	80.6%

The paper that used Usability Techniques has adapted techniques for general purpose languages to the DSLs' context [31]. Papers in the Ad-hoc category focused on visual issues pointed out by subjects, such as layout [33], usage of familiar icons and commands [40], interactive dialogs to increase users performance [47], and the impact that an iterative development process in cooperation with subjects has in usability [46]. Finally, [30] developed three Domain Specific Visual Languages, each one with an intended target user group, and reported using usability trials, without specifying the exact procedure. The author also compared several parameters, such as consistency and error-proneness.

6 Discussion

We found a low level of experimentation reported in the surveyed papers. Although roughly half of the papers report with some detail the development process of the DSL, only about 14% of the papers report either a quantitative or a qualitative evaluation of the DSL and they provide very few details on what was done. Researchers planning to replicate such evaluations would suffer from a lot of tacit knowledge, which is a well-known factor hampering the independent validation of claims supported through experimentation[57, 58].

Another shortcoming in the reviewed work was the predominance of toy examples, when compared to the usage of industry level examples. This represents a threat to the validity of claims made in such papers, as the conclusions drawn from toy examples do not necessarily scale up for industry level ones. However, it may be the case that our sample is bloating this effect. Most of the publications scrutinized in this review are workshops. Therefore, it may be the case that the kind of chosen examples are more targeted to what we typically find in workshops: it is common for authors to present work in progress papers in such venues, to get valuable feedback from the community to their approaches, and then mature their work and publish more validated claims in major conferences and journals. Another point to consider is that the focus of the chosen venues is on the technical aspects of the construction and engineering of these languages, rather than on reporting real-world cases. Such cases are more likely to be published in domain-specific venues.

A side effect of the level of detail on the reports of experimentation is that we often do not know who were the subjects involved in the process itself. Again, this is a threat, as we do not know the extent to which domain experts were really involved in this process, in most cases. Without characterizing the users, the validity of any conclusions concerning the usability of the DSLs is completely questionable. For instance, a newbie might have difficulties using a DSL, not because of problems with the DSL itself, but due to shortcomings of his own expertise in the domain the DSL is targeted to.

In any systematic review, we must explicitly address its inherent validity threats [59]. Although we were very conservative in our selection (when in doubt, we kept the paper for further review), it is always possible that some papers may have been missed, either because we failed to understand the abstract, or because the abstract was incomplete and did not cover the validation of the proposals with enough detail. Another common threat in systematic reviews concerns the misclassification of papers. This can happen when the reviewers mis-understand some important information about the paper and classify it in the wrong category (e.g., a qualitative study is counted as a quantitative one). We mitigated this threat by creating objective criteria to classify the surveyed papers, thus minimizing subjectiveness in the data collection.

7 Conclusions

Domain driven development is an increasingly popular way of developing software that aims to leverage the contribution of domain experts in such development. There are several claims of the benefits of this approach, in well-defined and constrained domains. However, when we look for evidence supporting such claims, we do not find them, either because they were not made explicit, or because they do not exist.

The SLE community does not systematically report on the realization of any sort of experimental validation of the languages it builds. While this does not necessarily mean that no evaluation is performed, it sends the wrong message to Engineering practitioners, which should always be concerned in systematically evaluating its products. In this paper, we support this claim by reviewing a large set of publications from the SLE community.

One of the outcomes from this work is that now the SLE community has an evidence to back up the awareness to this problem, which is necessary first step toward solving it. Therefore, one of the present challenges to the community, which is clearly interested in reinforcing the “Engineering” in SLE, is to foster the systematic evaluation of the produced languages as part of the standard of practice in the development process. The solution may either to go for a “de facto” solution, based on a high standard state of practice (this would require the community to be concerned with this subject, when publishing its work), or for a “the jure” solution, in which the community would set up an agreed standard for enforcing this kind of validation.

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. Smolander, K., Tahvanainen, V.P., Marttiin, P.: Metaedit - a flexible graphical environment for methodology modelling. In: International Conference on Advanced Information Systems Engineering, CAISE’91, Trondheim, Norway (1991)
5. Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In: 8th International Conference on Advanced Information Systems Engineering, CAiSE’96. Volume 1080/1996., Heraklion, Crete, Greece, Springer Berlin / Heidelberg (1996) 1–21

6. Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Redbooks. (2004)
7. Vanderbilt: Gme: Generic modeling environment (2007)
8. Cook, S., Jones, G., Kent, S., Wils, A.C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley Professional (2007)
9. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys **37**(4) (2005) 316–344
10. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison Wesley Longman, Inc. (1999)
11. MetaCase: Nokia case study. Technical report (2007)
12. MetaCase: Eads case study. Technical report (2007)
13. Kosar, T., López, P.E.M., Barrientos, P.A., Mernik, M.: A preliminary study on various implementation approaches of domain-specific language. Information and Software Technology **50**(5) (2008) 390–405
14. White, J., Hill, J.H., Tambe, S., Gokhale, A., Schmidt, D.C.: Improving domain-specific language reuse with software product line techniques. IEEE Software **26**(4) (2009) 47–53
15. Batory, D., Johnson, C., MacDonald, B., Heeder, D.v.: Achieving extensibility through product-lines and domain-specific languages: a case study. ACM Transactions on Software Engineering and Methodology **11**(2) (2002) 191–214
16. Kieburtz, R.B., McKinney, L., Bell, J.M., Hook, J., Kotov, A., Lewis, J., Oliva, D.P., Sheard, T., Smith, I., Walton, L.: A software engineering experiment in software component generation. In: International Conference on Software Engineering (ICSE'1996), Berlin, Germany, IEEE Computer Society (1996) 542–552
17. Hermans, F., Pinzger, M., Deursen, A.V.: Domain-specific languages in practice: A user study on the success factors. In: 12th International Conference on Model Driven Engineering Languages and Systems. Volume 5795/2009., Denver, Colorado, USA, Lecture Notes in Computer Science (2009) 423–437
18. Prechelt, L.: An empirical comparison of seven programming languages. IEEE Computer **33**(10) (2000) 23–29
19. Barbero, M., Bézivin, J., Jouault, F.: Building a dsl for interactive tv applications with amma. In: Model-Driven Development Tool Implementers Forum, Zurich, Switzerland (2007)
20. Bencomo, N., Blair, G.: Genie: a domain-specific modeling tool for the generation of adaptative and reflective middleware families. In: OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA (2006)
21. Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G.: Genie: Supporting the model driven developmnt of reflective, component-based adaptative systems. In: 30th International Conference on Software Engineering (ICSE'2008), Leipzig, Germany (2008) 811–814
22. Bettin, J.: Measuring the potencial of domain-specific modelling techniques. In: OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA (2002)
23. Bierhoff, K., Liongosari, E.S., Swaminathan, K.S.: Incremental development of a domain-specific language that supports multiple application styles. In: OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA (2006)
24. Carlson, J.W.: A visual language for data mapping. In: OOPSLA 2001 Workshop on Domain-Specific Visual Languages, Tampa Bay, Florida, USA (2001)

25. Correal, D., Casallas, R.: Using domain specific languages for software process modeling. In: OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada (2007)
26. Evermann, J., Wand, Y.: Toward formalizing domain modeling semantics in language syntax. IEEE Transactions on Software Engineering **31**(1) (2005) 21–37
27. Furtado, A.W.B., Santos, A.L.M.: Using domain-specific modeling towards computer games development industrialization. In: OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA (2006)
28. Grant, E.S., Whittle, J., Chennamaneni, R.: Checking program synthesizer input/output. In: OOPSLA Workshop on Domain-Specific Modeling, Anaheim, California, USA (2003)
29. Gray, J., Bapty, T., Neema, S.: An example of constraint weaving in domain-specific modeling. In: OOPSLA Workshop on Domain-Specific Visual Languages, Tampa-Bay, Florida, USA (2001)
30. Grundy, J.C., Hosking, J.G., Amor, R.W., Mugridge, W.B., Li, Y.: Domain-specific visual languages for specifying and generating data mapping systems. Journal of Visual Languages and Computing, Elsevier **15** (2004) 243–263
31. Haugen, ., Mohagheghi, P.: A multi-dimensional framework for characterizing domain specific languages. In: OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada (2007)
32. Hemel, Z., Verhaaf, R., Visser, E.: Webworkflow: An object-oriented workflow modeling language for web applications. In Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M., eds.: 11th International Conference on Model Driven Engineering Languages and Systems (MODELS 2008). Volume LNCS 5301., Toulouse, France, Springer (2008) 113–127
33. Hosking, J., Mehandjiev, N., Grundy, J.: A domain specific visual language for design and coordination of supply networks. In: IEEE Symposium on Visual Languages and Human-Centric Computing. (2008) 109–112
34. Howard, L.: Cape: A visual language for courseware authoring. In: OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA (2002)
35. Jouault, F., Bézivin, J., Consel, C., Kurtev, I., Latry, F.: Building dsls with amma/atl a case study on spl and cpl telephony languages. In: ECOOP Workshop on Domain-Specific Program Development (DSPD), Nantes, France (2006)
36. Kolovos, D.S., Paige, R.F., Rose, L.M., Polack, F.A.C.: Implementing the interactive television applications case study using epsilon. In: Model-Driven Development Tool Implementers Forum, Zurich, Switzerland (2007)
37. Luoma, J., Kelly, S., Tolvanen, J.P.: Defining domain-specific modeling languages: Collected experiences. In: OOPSLA Workshop on Domain-Specific Modeling, Vancouver, British Columbia, Canada (2004)
38. Merilinna, J.: Domain-specific modelling language for navigation applications on s60 mobile phones. In: OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA (2008)
39. Merilinna, J., Pärssinen, J.: Comparison between different abstraction level programming: Experiment definition and initial results. In: OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada (2007)
40. Mora, B., García, F., Ruiz, F., Piattini, M.: Smml: Software measurement modeling language. In: OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA (2008)
41. Patki, T., Al-Helal, H., Gulotta, J., Hasen, J., Sprinkle, J.: Using integrative modeling for advanced heterogeneous system simulation. In: OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA (2008)

42. Pohjonen, R., Kelly, S.: Interactive television applications using metaedit+. In: Model-Driven Development Tool Implementers Forum, Zurich, Switzerland (2007)
43. Prähofer, H., Hurnaus, D., Mössenböck, H.: Building end-user programming systems based on a domain-specific language. In: OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA (2006)
44. Prähofer, H., Hurnaus, D., Wirth, C., Mössenböck, H.: The domain-specific language monaco and its visual interactive programming environment. In: IEEE Symposium on Visual Languages and Human-Centric Computing, Coeur d'AlÃ¨ne, Idaho, USA (2007) 104–110
45. Reichert, T., Klaus, E., Schoch, W., Meroth, A., Herzberg, D.: A language for advanced protocol analysis in automotive networks. In: International Conference on Software Engineering, ICSE'2008, Leipzig, Germany, ACM (2008) 593–602
46. Sadielek, D.A.: Prototyping domain-specific languages for wireless sensor networks. In: Workshop on Software Language Engineering, Nashville, Tennessee, USA (2007)
47. Schmidt, C., Pfahler, P., Kastens, U., Fischer, C.: Simtelligence designer/j: A visual language to specify sim toolkit applications. In: OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA (2002)
48. Souza, M.R.S., Nelson, M.A.V.: A domain-specific language for interoperability between object-oriented and mainframe systems. In: Workshop on Domain-Specific Program Development, DSPD 2008, Nashville, Tennessee, USA (2008)
49. Sprinkle, J., Karsai, G.: A domain-specific visual language for domain model evolution. Journal of Visual Languages and Computing, Elsevier **15**(3-4) (2004) 291–307
50. Svansson, V., Lopez-Herrejon, R.E.: A web specific language for content management systems. In: OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada (2007)
51. Tairas, R., Liu, S.H., Jouault, F., Gray, J.: Coclorep: A dsl for code clones. In: Workshop on Software Language Engineering, Nashville, Tennessee, USA (2007)
52. Teiken, Y., Floring, S.: A common meta-model for data analysis based on dsm. In: OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA (2008)
53. Trask, B., Roman, A.: Using domain specific modeling in developing software defined radio components and applications. In: ECOOP Workshop on Domain-Specific Program Development (DSPD), Nantes, France (2006)
54. Zeng, J., Mitchell, C., Edwards, S.A.: A domain-specific language for generating dataflow analyzers. In: Workshop on Language Descriptions, Tools and Applications, Vienna, Austria (2006)
55. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering **25**(4) (1999) 557–572
56. Nielson, J.: Usability Engineering. AP Professional (1993)
57. Shull, F., Basili, V.R., Carver, J., Maldonado, J.C., Travassos, G.H., Mendonça, M., Fabbri, S.: Replicating software engineering experiments: addressing the tacit knowledge problem. In: 2002 International Symposium on Empirical Software Engineering (ISESE'02), IEEE Computer Society (2002) 7–16
58. Shull, F., Mendonça, M., Basili, V.R., Carver, J., Maldonado, J.C., Fabbri, S., Travassos, G.H., Ferreira, M.C.: Knowledge-sharing issues in experimental software engineering. Empirical Software Engineering **9**(1-2) (2004) 111–137
59. Kitchenham: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report (July 2007)