Evaluating the Usability of Domain-Specific Languages

Ankica Barisic, Vasco Amaral, Miguel Goulão, Bruno Barroca

Evaluating the Usability of Domain-Specific Languages

Ankica Barišić, Vasco Amaral, Miguel Goulão, Bruno Barroca
CITI - Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal

ABSTRACT
We can regard Domain-Specific Languages (DSLs) as User Interfaces (UIs) because they bridge the gap between the domain experts and the computation platforms. Usability of DSLs by domain experts is a key factor for their successful adoption. The few reports supporting improvement claims are persuasive, but mostly anecdotal. Systematic literature reviews show that evidences on the effects of the introduction of DSLs are actually very scarce. In particular, the evaluation of usability is often skipped, relaxed, or at least omitted from papers reporting the development of DSLs. The few exceptions mostly take place at the end of the development process, when fixing problems is already too expensive. A systematic approach, based on techniques for the experimental evaluation of UIs, should be used to assess suitability of new DSLs. This chapter presents a general experimental evaluation model, tailored for DSLs' experimental evaluation, and instantiates it in several DSL's evaluation examples.

INTRODUCTION
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 (Kleppe, 2009). A typical SLE process starts with the domain engineering phase, in order to elicit the domain concepts. The next phase consists in the actual design of the language, by capturing the referred concepts and their relationships. This is followed by its implementation, evaluation, deployment, evolution, and finally its retirement. Although this process is becoming streamlined, it still presents a serious gap in what should be a crucial phase - **language evaluation**, which includes acceptance testing. A good DSL is hard to build because it requires domain knowledge and language development expertise, and few people have both (Mernik, Heering & Sloane, 2005). We should evaluate claims such as *"our new language brings efficiency to the process"*, or that *"our new language is usable and effective"*, with an unbiased and objective process.
DSLs are meant to close the gap between the Domain Experts and the computation-platforms. As such, DSLs can be used as a structured/comprehensive means to achieve Human/Computer (H/C) Interaction. Most of the requirements concerning the evaluation of User Interfaces (UI) are actually associated with a qualitative software characteristic called **Usability**, which is defined by the quality standards in terms of achieving the Quality in Use (ISO, 2001a).
Usability evaluation involves a phase of acceptance testing with actual users, which is typically a very costly process. A poorly conceived evaluation process can ultimately undermine the conclusions about the quality of the UI under analysis. This generic UI problem also applies to the realm of DSL's construction.
In our opinion, usability can be fostered from the beginning of the DSL development cycle by adopting user centered methods. The objective is to ensure that the developed DSLs can be used by real people (the domain experts) to perform their tasks in the real world. This requires not only intuitive UIs, but also the appropriate functionality and support for the activities and workflows that are to be specified with the DSLs.
In this chapter, we discuss how user-centered design can be adapted to the context of DSL's development. In general, working with languages involves not only physical and perceptual activities, but also cognitive activities such as learning, understanding and remembering. Experimenters in human factors have

developed a list of tasks to capture those particular aspects. The process is complex and must be tailored case-by-case (Reisner, 1988). We will further discuss these issues, and show how they fit into the DSL's development process. Following that, we will define a general model for the DSL's experimental evaluation. This model will help us planning and designing the DSL's evaluation processes, as well as conducting post-mortem analysis of other DSL's evaluation efforts in a systematic way, thus fostering the aggregation of several DSL's evaluation results. As discussed in (Basili, 2007), a single study outside the context of a larger set of studies has limited value, but combined, they can be a valuable increment to the existing body of knowledge.

The usage of our model is illustrated through the systematic analysis of several evaluations of DSLs found in the literature. Our comparative analysis will help identifying the commonalities, differences, strengths and weaknesses of the compared studies. The usage of our model in future replications of this comparative study to other DSL evaluations has the potential for fostering meta-analysis, leading to sound increments of the body of knowledge in DSLs and their evaluation.

BACKGROUND

In general, the software industry does not report investment on the usability evaluation of DSLs, as shown in a recent systematic literature review (Gabriel, Goulão & Amaral, 2010). This conveys a perception that there is an insufficient understanding of the SLE process which, in our opinion, must include the evaluation of the produced DSLs. Many language engineers may perceive the investment in usability evaluation as an unnecessary cost and prefer to risk providing a solution which has not been properly validated, namely with respect to its usability, by the end users. This apparent state of practice contrasts with the return of investment on usability reported for other software products (Nielsen & Gilutz, 2003). In general, these benefits span from a reduction of development and maintenance costs, to increased revenues brought by an improved effectiveness by the end users (Marcus, 2004).

Software language evaluation

Comparing the impact of different languages in the software development process has some tradition in the context of General Purpose Languages (GPLs) (e.g., (Prechelt, 2000)), namely concerning their impact on developer's productivity. Typically, the popularity (see, for an instance of a popularity index, http://lang-index.sourceforge.net/) of a language is used as a surrogate for its usability. The rationale for this informal assessment is that, if there are so many people using a particular GPL, then that must say something about its usability. Naturally, usability is only one of several factors that make a language popular. Historic reasons, for instance, also play a major role. In any case, this kind of indirect usability assessment is not adequate to be applied to DSLs as they are often intended for a small number of users, and it is generally not easy to know neither the size of the community that is actually using a DSL nor the potential size of that community (i.e. other domain experts that might use the DSL in the future). In any case, it would only make sense to use community size for comparing DSLs within the same domain.

Other sorts of evaluations on GPLs include benchmarks, feature-based comparisons and heuristic-based evaluations (Prechelt, 2000). These language comparisons are done on different versions of the same language or on different languages, focusing on a subset of characteristics that indicate the suitability of languages to a specific intended Context of Use. There are also Heuristic-based evaluations that provide guidelines for evaluating syntax of visual languages based on the studies of cognitive effectiveness (Moody, 2009). Because the end users of the GPLs are the people who are usually close to computation concepts, while the ones of DSLs are closer to domain concepts of the context of use, these methods are not appropriate for DSLs in all cases. However, it is necessary to take in consideration these methods and adapt them for DSLs.

When usability problems are identified too late, a common approach to mitigate them is to build tool support that minimizes their effect on users' productivity (Bellamy, John, Richards & Thomas, 2010;

Phang, Foster, Hicks & Sazawal, 2009). There is an increasing awareness to the usability of languages, fostered by the competition of language providers. Better usability is a competitive advantage, although evaluating it remains challenging: it is hard to interpret existing metrics in a fair, unbiased way, which is resistant to external validity threats concerning the broad user groups, or internal ones – it is very easy to end up comparing apples with oranges, when evaluating competing languages.

The increased productivity achieved by using DSLs, when compared to using GPLs, is one of the strongest claims by the DSL community. With anecdotal reports of 3-10 times productivity improvements of DSLs, (Kelly & Tolvanen, 2000; MetaCase, 2007b; Weiss & Lai, 1999), or *"clearly boosted development speeds"* (MetaCase, 2007a) in industrial settings, why bother with their validation?

The problem, of course, is that those anecdotal reports on improvements lack external validity. Other reports, such as (Batory, Johnson, MacDonald & Von Heeder, 2002), present maintainability and extensibility improvements brought by a combination of DSLs and Software Product Lines (SPL), but it is unclear which share of the merits belongs to DSLs and which should be credited to SPLs. The usage of DSLs has been favorably compared to the usage of templates in code generation, with respect to flexibility, reliability and usability (Kieburtz, McKinney, Bell, Hook, Kotov, Lewis, Oliva, Sheard, Smith & Walton, 1996). Another success story can be found in (Hermans, Pinzger & Deursen, 2009), where a survey conducted with users of a particular DSL clearly reports on noticeable improvements in terms of reliability, development costs, and time-to-market. The usability of that particular DSL and its toolset are among the most important success factors of DSL introduction in that context. But are these improvements typical, or exceptional? The honest answer can only be one: we do not know. Comparisons can also be made among competing DSLs: for instance, (Murray, Paton, Goble & Bryce, 2000) compare a visual DSL against the textual language for which it is a front-end.

The incremental nature of a typical DSL life cycle may give to the language engineers an erroneous feeling that their language is being validated through the interaction with the domain experts that are helping to build the language. The problem is that these domain experts are not necessarily the language's end users, so they may introduce biases in the perception of the language's usability.

Domain-Specific languages as User Interfaces

Intuitively, a language is a means for communication between peers. For instance, two persons can communicate with each other by exchanging sentences. These sentences are composed by signs in a particular order. According to the context of a conversation, these sentences can have different interpretations. If the context is not clear, we call these different interpretations ambiguous. Here, we focus our attention in the communication between humans and computers. We only consider languages that are used as communication interfaces between humans and machines i.e. UIs. Human-human languages (e.g., natural languages) and machine-machine languages (e.g., communication protocols) are not discussed this chapter. Examples of UIs range from compilers, to command shells and sophisticated graphical applications. In each of those examples we can deduce the (H/C) language that is being used to perform that communication: in compilers we may have a programming language; in a graphical application, we may have an application specific language with some visual syntax, and so on. Moreover, we argue that any UI is a realization of a language. This view is in line with that of a growing community, built around the PLATEAU workshop series (http://ecs.victoria.ac.nz/Events/PLATEAU/WebHome), that aims to bridge the gap between language engineers and UIs experts, so that the former can build languages that are easier to use, leading to increased productivity by their users. In this perspective, we define DSLs as being languages that reduce the use of computation domain concepts and focus on the domain concepts of the contexts of use's problem.

Usability is a key characteristic for evaluating the Quality of UIs. Since we defined H/C languages as UIs, in our perspective, we should also use it for evaluating the Quality of this family of languages. The difference between usability and the other software qualities is that to achieve it, one has to concentrate

not only on system features but specifically on user-system interaction characteristics. The term usability is overloaded and has been given several interpretations and definitions. The need for a generally accepted usability definition is discussed in several references (Bevan, 1999, 2009; Petrie & Bevan, 2009).

ISO 9241-11 provides the definition of usability that is used in subsequent related ergonomic standards (ISO, 2001b). ISO 9126 (ISO, 2001a) extends that definition by introducing the notion of **Quality in Use**, i.e., the quality as perceived by the user during actual utilization of a product in a real Context of Use. Quality in Use is measured in terms of the result of using the software, rather than on properties of the software itself. The ISO standard states that achievement of Quality in Use can be assured by achieving internal and external Quality. These two types of Quality provide us metrics that can be used early in software development process.

The complete Quality model for achieving Quality in Use is given by ISO IEC CD 25010.3 (Petrie & Bevan, 2009) and is discussed in the context of DSL evaluation in (Barišić, Amaral, Goulão & Barroca, 2011a). This model provides us with a complete structure of quality, but we cannot take general conclusions about which characteristics will lead us to final usability, as they are dependent on the DSLs intended context of use.

EVALUATING A DSL

Evaluation with users, also known as Empirical Evaluation, is recommended (Nielsen & Molich, 1990) at all stages of development, if possible, or at least in the final stage of development. **Formative methods** focus on understanding the user's behavior, intentions and expectations in order to understand any problems encountered, and typically employ a 'think-aloud' protocol. **Summative methods** measure the product usability, and can be used to establish and test user requirements. Testing may be based on the principles of standards and measure a range of usability components. Each type of measure is usually regarded as a separate factor with a relative importance that depends on the Context of Use (Barišić, Amaral, Goulão & Barroca, 2011c). Iterative testing with small numbers of participants is usually preferable, starting early in design and development process.

Iterative User Centered Evaluation Practices

User Centered Design can reduce development and support costs, increase sales, and reduce staff cost for employers by allowing significant changes to correct deficiencies along the development process instead of just evaluating at the end of it, when it might be too late (Catarci, 2000). The essential activities required to implement User Centered Design are described in ISO 13407 (Bevan, 2005).

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. In the first phase it is important to study existing style guidelines, or standards for a particular type of system. Interviewing current or potential users about the current approach they are using to accomplish their tasks, can also help identifying its strengths and weaknesses, and their expectations for a new or re-designed system. It is also important to assess the Context of Use of a particular situation. All these contribute to an initial understanding of what the system should do for the users and how it should be designed. Initial design ideas can then be explored, considering alternative designs and how they meet user's needs. After developing potential designs it is time to build the prototypes that should be obviously simple and unfinished, as that allows people involved in evaluations to realize that it is acceptable to criticize them. In contrast, a prototype very close to the final product is likely to inhibit evaluators from openly criticizing it, which might lead to a loss of valuable feedback from those evaluators. It is important to explore particular design problems before considerable effort is put into full implementation and integration of components of the system. A number of iterations of evaluation, designing and prototyping may be required before acceptable levels of Usability are reached. Once the design of various components

of the system has reached acceptable levels, the integration of components and final implementation of the interactive system may be required. Finally, once the system is released to users, an evaluation of its use in real contexts may be highly beneficial (Petrie & Bevan, 2009). This kind of iterative evaluation approach should be merged with the DSL development cycle (Barišić, Amaral, Goulão & Barroca, 2011a).

Cognitive factors involved

In order to know the users we should identify the characteristics of the target user population. For several kinds of end users we should analyze these characteristics using techniques like questionnaires, interviews and observations. Understanding *"how"* and *"why"* should give us a deeper knowledge about the tasks. Performing task analysis by studying the way people perform tasks with existing systems, or by having a high level abstraction study of cognitive processes, we should identify what are the individual tasks that the language should enable to perform. From this, we can model the desired cognitive model for the language context based on user-task scenarios. For each task we should identify: *Goal*, *Pre-conditions*, *Dependencies*, *User background* and *Sub tasks*.

The cognitive activities that should be analyzed in the study of cognitive processes are:

- **Learning** both syntax and semantics;
- **Composition** of the syntax required to perform a function;
- **Comprehension** of function syntax composed by someone else;
- **Debugging** of syntax or semantics written by ourselves or others;
- **Modification** of a function written by ourselves or others.

Experimenters in human factors have developed a list of tasks to capture these particular aspects (Reisner, 1988): *Sentence writing*, *Sentence reading*, *Sentence interpretation*, *Comprehension*, *Memorization* and *Problem solving*. To evaluate these tasks, we can use tests like: *Final exams*, *Immediate Comprehension*, *Reviews*, *Productivity*, *Retention* and *Re-learning*.

Testing different tasks in the language usage is interesting, but to perform an exhaustive evaluation of all of them would be very expensive. Therefore, the evaluation usually concerns only the most critical activities.

Evaluation process experiments

We argue that the quality in use of a DSL should be assessed experimentally. In Software Engineering, a controlled experiment can be defined as *"a randomized experiment or quasi-experiment in which individuals or teams (the experimental units) conduct one or more Software Engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages or tools (the treatments)."* (Sjøberg, Hannay, Hansen, Kampenes, Karahasanovic, Liborg & Rekdal, 2005). For our purposes, this can be instantiated with developers typically conducting software construction, or evolution tasks, for the sake of comparing different languages – including the DSL under evaluation and any existing baseline alternatives to that DSL.

Experiment activity model

Figure 1 outlines the activities needed to perform an experimental evaluation of a software engineering claim, following the scientific method. During **requirements definition**, the problem statement (i.e. research questions), experimental objectives and context are defined. The next step is to perform **design planning**, where context parameters and hypotheses are refined, subjects are identified, a grouping strategy for subjects is selected, and a sequence and synchronization of observations and treatments for each of the experimental groups is planned. The sequencing and synchronization of such interventions, their nature (observations or treatments) and the group definition policy, define the **experimental design**. The data collection activities plan is also set during design planning. This is followed with **data collection**, which often includes a pilot session, to correct any remaining issues, and the evaluation itself,

following the designed plan. This step is followed with **data analysis** where data is described in the form of statistical tables and graphs, and, if necessary, the data set is reduced. Hypotheses are then tested. Finally, during **results packaging**, the results are interpreted and possible validity threats and lessons learned are identified. A detailed discussion on how this process can be followed in a software engineering experimentation context can be found in (Miguel Goulão, 2008; Goulão & Abreu, 2007). Experimental reporting guidelines, generally followed by the experimental software engineering community, are also available (Jedlitschka, Ciolkowski & Pfahl, 2008). By reporting a given language's quality in use, and the evaluations adhering to such guidelines, the overall ability to make study replications (for independent validation and validity threats mitigation) and its meta-analysis (for building a body of knowledge supported by the evidence collected in different contexts) is expected to increase.

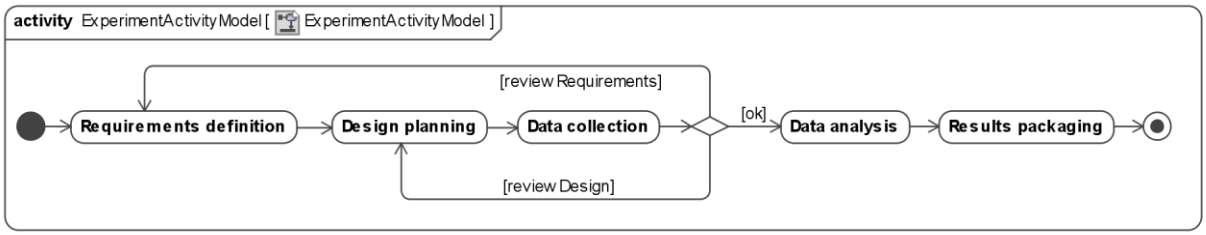

Figure 1. Experiment Activity Model Overview.

Experiment Design Model

In order to contrast the selected DSL experimental validations, we start by modeling their relevant information. This is captured in the class diagrams, adapted and extended from (M. Goulão, 2008). In a nutshell, this model partially captures some of the essential information of an experimental language evaluation, namely the details on evaluation requirements and planning.

Before conducting an experimental language evaluation, one should start by clearly defining the problem that the evaluation will address as modeled in *Figure 2*. This includes identifying where this problem can be observed (i.e., its context, typically where the language will be used), and by whom (i.e., the stakeholder who is affected by the problem – e.g., the language user). It is also important to state how solving the identified problem is expected to impact on those who observe it, and which quality attributes will be affected. The class QualityAttribute can take values that are defined in Quality model from ISO Standards (Barišić, Amaral, Goulão & Barroca, 2011a).

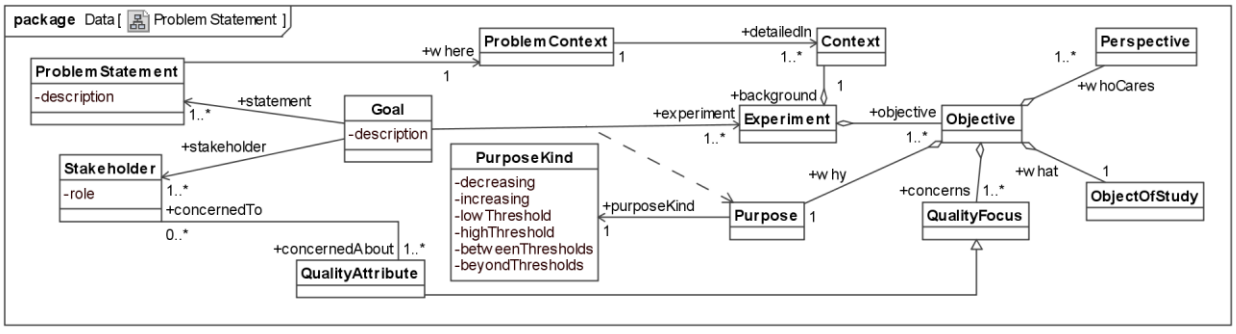

Figure 2. Problem Statement design model.

When conducting language evaluation experiments, one should clearly define the experiments' objectives. Building upon Basili's earlier work (Basili, 1996), Wohlin *et al.* proposed a framework to guide the experiment definition (Wohlin, Runeson, Höst, Ohlsson, Regnell & Wesslén, 1999). The

framework is to be mapped into a template with the following elements: the **object of study** under analysis, the **purpose** of the experiment, its **quality focus**, the **perspective** from which the experiment results are being interpreted, and the **context** under which the experiment is run.
While the experiment definition expresses something about why a particular language evaluation was performed, the experiment planning expresses something about how it will be performed. Before starting the experiment, decisions have to be made concerning the **context** of the experiment, the **hypotheses** under study, the set of **independent** and **dependent variables** that will be used to evaluate the hypotheses, the selection of **subjects** participating in the experiment, the experiment's **design** and **instrumentation**, and also an evaluation of the experiment's validity. Only after all these details are sorted out should the experiment be performed. The outcome of planning is the **experimental language evaluation design**, which should encompass enough details in order to be independently replicable.

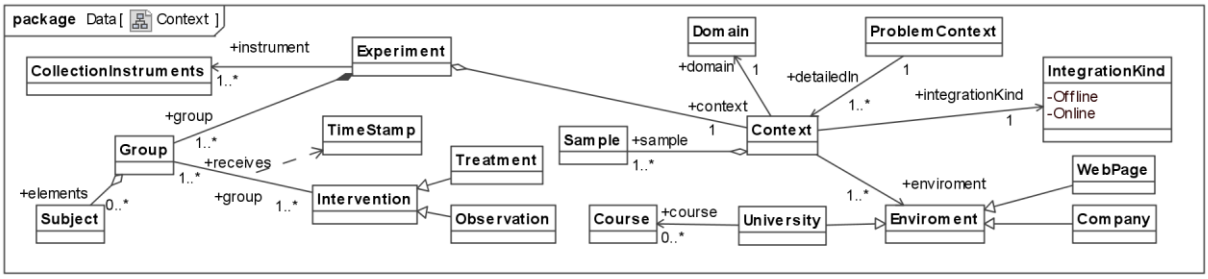

Figure 3. Context design model.

Figure 3 includes information on the **context**, including where the experimental language evaluation will take place. The context of an experiment determines our ability to generalize from the experimental results to a wider context. Experiments can be conducted in different contexts, each of them with their own benefits, costs, and risks. These constraints have to be made explicit, in order to ensure the comparability among different studies, and to allow practitioners to evaluate the extent to which the results obtained in a study, or set of studies, are applicable to their own particular needs. Throughout the experiment, there are a number of context parameters that remain stable and their value is the same for all the subjects in the experiment during the whole process. Therefore, we can safely assume that differences observed in the results cannot be attributed to these parameters, while the actual parameters to be reported may vary (Wohlin, Runeson, Höst, Ohlsson, Regnell & Wesslén, 1999). Concerning their integration within the language development process, experiments can be conducted either **online**, or **offline**. The former, carried as part of the software process in a professional environment, involve an element of risk, since experiments may become intrusive in the underlying development activity. This intrusiveness may even manifest itself through resources and time overheads on a real project. A common alternative is to carry out the experiment offline.
An experimental language evaluation design prescribes the division of our sample into a set of groups, according to a given strategy. Each of those groups receives a set of **interventions**, which may be either **observations** where data is collected, or **treatments**, where the groups receive some sort of input (e.g., training in using a language). The association class with the time stamp allows this data to be ordered in time, so that a sequence of observations and treatments can be established. The sequencing and synchronization of such interventions, their nature, and the group definition policy, define the **experimental design**.
The **instrument** design presented in *Figure 4* includes the definition of the artifacts that will be used in the experiment. For instance, in a language evaluation experiment, the syntactical problem instantiation specified with a language can be used as an artifact that will then be changed by the evaluation participants. These changes could be monitored, using a collection instruments such as those depicted in

Figure 4 – e.g., a combination of a test with a post-test questionnaire. This kind of evaluation allows addressing the instrument perspectives as cognitive activities that are fundamental to assessing the usability of a language, and the quality of instantiation, especially during modification (see, for instance the usage of cognitive dimensions in (Kosar, Mernik & Carver, 2012; Kosar, Oliveira, Mernik, Pereira, Črepinšek, Cruz & Henriques, 2010). The instrumentation also concerns the production of guidelines, and tools (not necessarily computer-based ones) that will support the measurements performed in the experiment. The rationale is to foster the comparability of the collected data by streamlining data collection in a consistent way. Note that instrumentation may also include any training material distributed to the participants, before their participation in the experiment.

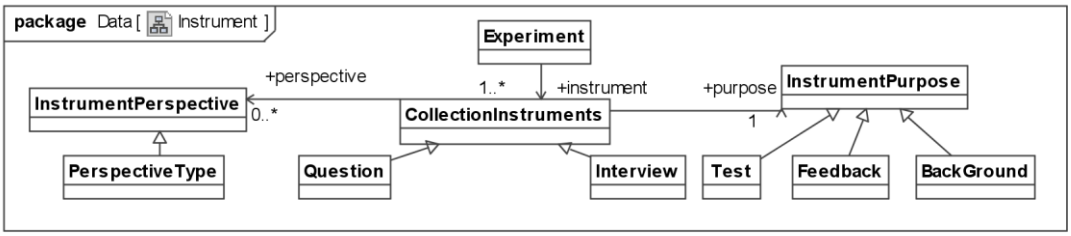

Figure 4. Instrument design model.

In *Figure 5*, we can see the **sample** design model that includes the participants' profile and the artifacts used in the language evaluation. An orthogonal classification of context concerns the people involved in the language evaluation. One may choose among performing the language evaluation with professional practitioners, or with surrogates for those practitioners (e.g., students). The first option leads to results that are more easily comparable to others obtained in a professional context, but care must be taken to reduce potential overheads to practitioners' activities. Using students as surrogates for professional practitioners is less expensive, but makes the experimental results harder to extrapolate for a professional community. In order to reduce this gap between the students and the practitioners, the researcher should prefer using graduate students, whose expertise is closer to that of novice practitioners.

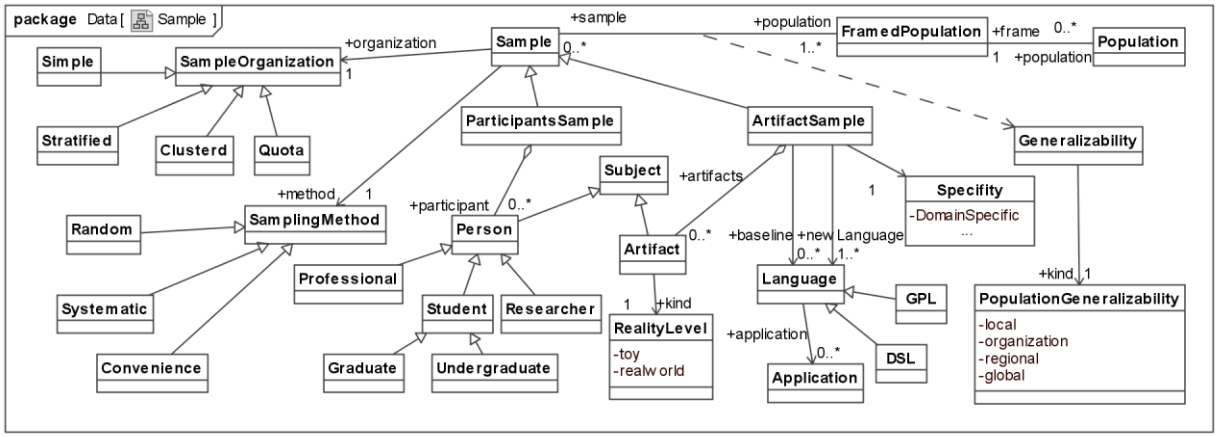

Figure 5. Sample design model.

It is common to use a frame of the population, if it is not feasible to identify all the population's members. In contrast, all members of the chosen population frame are identified. For example, rather than considering all the language components available, one can use a frame that considers only the selected language components as the population. Often, it is not possible to perform the evaluation using all the

relevant framed population as evolution subjects. Instead, a sample of that framed population is chosen using a selected sampling technique, with the objective of being as much representative of the framed population as possible, considering the available resources of the experimenter.
Yet another dimension constraining the language evaluation is the usage of **toy** vs. **real** problems. There are at least two issues that motivate the usage of toy problems: the resources available for the language evaluation and the risks concerned with the outcome of the evaluation. The former results from the often very limited amount of time that the subjects can devote to the evaluation. The latter relates to the potential harm caused by the outcome of the evaluation (e.g. while experimenting with using different languages on a real problem, a language that leads to worse productivity can lead to additional costs to a customer). The question, here, is whether the results obtained with a toy problem will scale up to real problems, or not. Toy problems are often used in early evaluations, as their usage is less expensive. If the results of evaluations conducted with toy examples are satisfactory, the risk of scaling up the problem to a real one may be mitigated to a certain extent, although it will not be completely eradicated.
The artifacts used in these evaluations can be *generic* or *domain-specific*. When comparing programming languages it is common for these artifacts to be domain-specific, regardless of the original language they were built with. This means, that we can use this model, taking in the consideration this attribute specification, to compare GPLs, DSLs, or GPLs vs. DSLs.

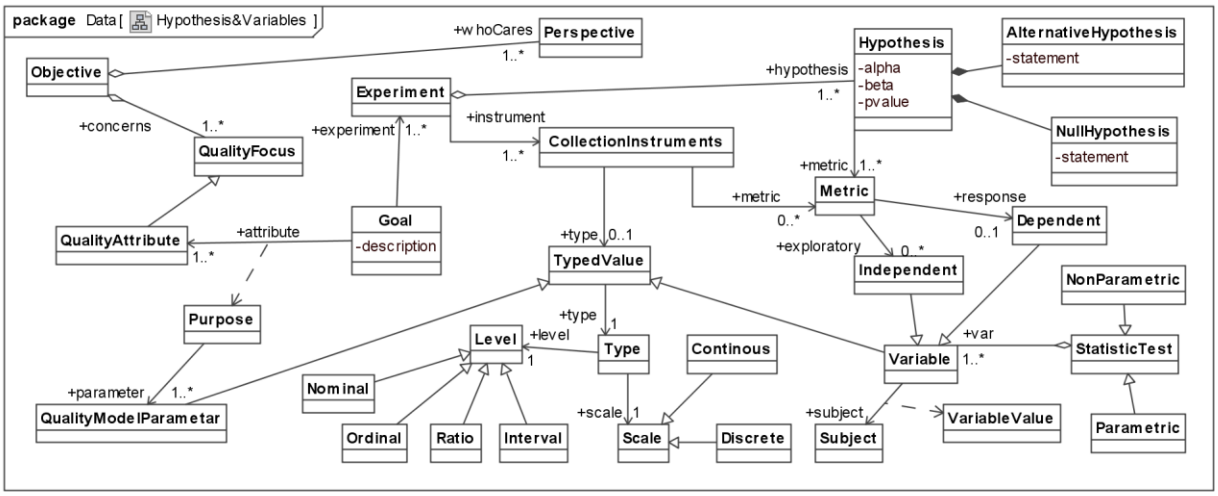

Figure 6. Hypothesis and Variables design model.

Figure 6 includes the **hypotheses tested** and the **variables** used with their characteristics, such as **type**, **scale**, and **level**. The hypothesis formulation should be stated as clearly as possible and pesented in the context of the theoretical background it is derived from. The **null hypothesis** states that there is no observable pattern in the experimental evaluation setting, so any variations found are resulting from coincidence. This is the hypothesis that the researcher is trying to reject. The alternative is that the variations observed are not resulting from coincidence. When the null hypothesis is rejected, we can conclude that the null hypothesis is false. However, if we cannot reject the null hypothesis, we can only say that there is no statistical evidence to reject it. Conversely, if we reject the null hypothesis, we can accept its alternative. If we cannot reject the null hypothesis, we cannot accept the alternative.
Hypothesis testing always assumes a given level of significance denoted by **alpha**, which represents a fixed probability of wrongly rejecting the null hypothesis, if it is in fact true. The probability value (**p-value**) of a statistical hypothesis test is the probability of getting a value of the test statistic as extreme as or more extreme than that observed by chance alone, if the null hypothesis is true. *Figure 6* presents the relationships between the main concepts involved in hypotheses definitions, starting from the overall

objectives of the research, through the specific goals of the experiment, and the questions that will allow assessing the achievement of the goals. The hypotheses are then assessed using metrics.
The language evaluator selects both **dependent** and **independent** variables. Dependent variables should be explicitly tied to the research goals (in the context of this chapter, these typically involve evaluating DSLs), and chosen for their relevance with respect to those goals. When it is not feasible to collect direct measures of the level of achievement of the research goals, surrogates can be used, although such replacement is to be avoided, when possible, and clearly justified. When not – e.g. when assessing the usability of a DSL – we may use **effectiveness** in specifying a system with it as a surrogate for the DSL's usability. Similarly, independent variables are chosen according to their relevance to the research goals.
The analysis techniques chosen for the language evaluation experiment depend on the adopted language evaluation design, the variables defined earlier, and the research hypotheses being tested. More than one technique may be assigned to each of the research hypotheses, if necessary, so that the analysis results can be cross-checked later. Furthermore, each of the hypotheses may be analyzed with a different technique. This may be required if the set of variables involved in that hypothesis differs from the set being used in the other hypotheses under tested. Discussions relating statistical tests (in particular, **parametric** *vs.* **non-parametric** ones) with variable types can be found in statistics text books, such as (Maroco, 2003).
By capturing a rich set of data of an language evaluation, we can pave the way for further analysis, where the information collected in several independently conducted language evaluations can be combined. To do so, the next step is to instantiate this model. In *Figure 7*, we illustrate a partial instantiation of this model, using information collected from the family of language evaluation experiments described (Kosar, Mernik & Carver, 2012). This particular example is chosen for illustration because that family of evaluation experiments is an excellent example of how DSL properties validation can be performed in a sound way. The instantiation is only partial, as the whole instantiation would be extremely cluttered.

Experiments overview

The main point in streamlining the evaluation of DSLs and making information available in a common framework is that we can build upon that framework an evidence-based body of knowledge on DSLs and their properties with respect to their usability. To illustrate this, we present a systematic comparison of four language evaluation experiments. As noted earlier in this chapter, these evaluations are currently exceptional in the realm of DSLs and are chosen precisely for that: they are examples of best practices in languages evaluation with a concern on usability, from which we can perform some meta-analysis, leading not only to a collection of lessons learned "from the trenches", but also to the identification of opportunities to further improve existing validation efforts. Table 1 outlines our comparison. The selected studies are (Barišić, Amaral, Goulão & Barroca, 2011b; Kieburtz, McKinney, Bell, Hook, Kotov, Lewis, Oliva, Sheard, Smith & Walton, 1996; Kosar, Mernik & Carver, 2012; Murray, Paton, Goble & Bryce, 2000). In this table, the first column represents a specific criterion that we will use in our comparative overview of these studies. The four remaining columns provide information on each of the selected studies. Kosar *et al.* conducted a family of three experiments, while the remaining selected studies are single experiments. The generic lack of families of experiments, rather than single experiments is a long identified shortcoming in the experimental validation of software engineering claims, so this should be highlighted as a very strong point in this work. Families of experiments help mitigating validity threats that occur in single experiments. In this particular case, the fact that the tested hypotheses have consistent results in all the three experiments in the experiment family increases the confidence in the soundness of the obtained results. Ideally, there should also be experiments within the family run by completely separate research groups, so that any biases by the experiment team that might exist would also be removed. Independent replication of experiments is a standard practice in other domains. For example, the Cochrane Collaboration (http://www.cochrane.org/) supports a common repository for health care evidence, which is fed by independently run families of experiments.

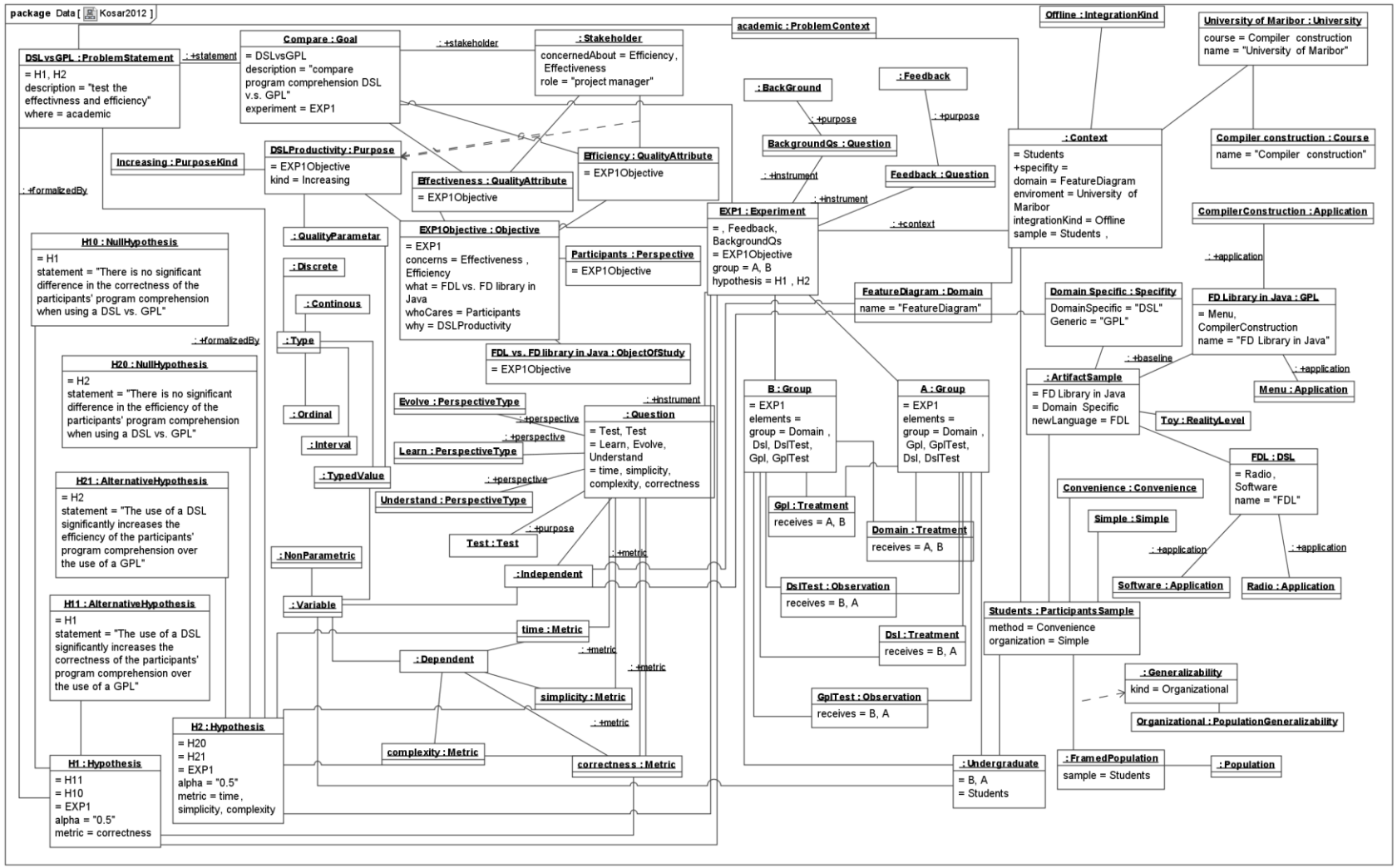
package Data [Kosar2012]
DSLvsGPL : ProblemStatement
= H1, H2
description = "test the effectivness and efficiency"
where = academic
: +statement
Compare : Goal
= DSLvsGPL
description = "compare program comprehension DSL v.s. GPL"
experiment = EXP1
: +stakeholder
: Stakeholder
concernedAbout = Efficiency, Effectiveness
role = "project manager"
academic : ProblemContext
Offline : IntegrationKind
University of Maribor : University
course = Compiler construction
name = "University of Maribor"
: BackGround
: Feedback
: +purpose
: +purpose
: Context
= Students
+specifity =
domain = FeatureDiagram
enviroment = University of Maribor
integrationKind = Offline
sample = Students ,
Compiler construction : Course
name = "Compiler construction"
BackgroundQs : Question
Feedback : Question
: +instrument
: +instrument
Increasing : PurposeKind
DSLProductivity : Purpose
= EXP1Objective
kind = Increasing
Efficiency : QualityAttribute
= EXP1Objective
Effectiveness : QualityAttribute
= EXP1Objective
: +formalizedBy
EXP1 : Experiment
= , Feedback, BackgroundQs
= EXP1Objective
group = A, B
hypothesis = H1 , H2
: +context
CompilerConstruction : Application
: +application
H10 : NullHypothesis
= H1
statement = "There is no significant difference in the correctness of the participants' program comprehension when using a DSL vs. GPL"
: QualityParametar
EXP1Objective : Objective
= EXP1
concerns = Effectiveness , Efficiency
what = FDL vs. FD library in Java
whoCares = Participants
why = DSLProductivity
Participants : Perspective
= EXP1Objective
: Discrete
: Continous
: Type
FeatureDiagram : Domain
name = "FeatureDiagram"
Domain Specific : Specifity
DomainSpecific = "DSL"
Generic = "GPL"
FD Library in Java : GPL
= Menu, CompilerConstruction
name = "FD Library in Java"
: +formalizedBy
: +baseline
: +application
H20 : NullHypothesis
= H2
statement = "There is no significant difference in the efficiency of the participants' program comprehension when using a DSL vs. GPL"
FDL vs. FD library in Java : ObjectOfStudy
= EXP1Objective
: ArtifactSample
= FD Library in Java
= Domain Specific
newLanguage = FDL
Menu : Application
Toy : RealityLevel
: Ordinal
: Interval
: TypedValue
Evolve : PerspectiveType
Learn : PerspectiveType
Understand : PerspectiveType
: +perspective
: +perspective
: +perspective
: +instrument
: Question
= Test, Test
= Learn, Evolve, Understand
= time, simplicity, complexity, correctness
B : Group
= EXP1
elements = group = Domain , Dsl, DslTest, Gpl, GplTest
A : Group
= EXP1
elements = group = Domain , Gpl, GplTest, Dsl, DslTest
FDL : DSL
= Radio , Software
name = "FDL"
Convenience : Convenience
Simple : Simple
Gpl : Treatment
receives = A, B
Domain : Treatment
receives = A, B
: +application
: +application
Software : Application
Radio : Application
H21 : AlternativeHypothesis
= H2
statement = "The use of a DSL significantly increases the efficiency of the participants' program comprehension over the use of a GPL"
: +purpose
Test : Test
: +metric
: NonParametric
: Independent
DslTest : Observation
receives = B, A
Students : ParticipantsSample
method = Convenience
organization = Simple
H11 : AlternativeHypothesis
= H1
statement = "The use of a DSL significantly increases the correctness of the participants' program comprehension over the use of a GPL"
: Variable
time : Metric
Dsl : Treatment
receives = B, A
: Generalizability
kind = Organizational
: +metric
: Dependent
: +metric
simplicity : Metric
GplTest : Observation
receives = B, A
Organizational : PopulationGeneralizability
: +metric
H2 : Hypothesis
= H20
= H21
= EXP1
alpha = "0.5"
metric = time, simplicity, complexity
complexity : Metric
correctness : Metric
H1 : Hypothesis
= H11
= H10
= EXP1
alpha = "0.5"
metric = correctness
: Undergraduate
= B, A
= Students
: FramedPopulation
sample = Students
: Population

Figure 7. Experiment design model instantiation, built from info in (Kosar, 2012).

Back in 1997, Brooks advocated that meta-analysis should be used to combine the results of independent study replications in Software Engineering (Brooks, 1997). Miller attempted to perform meta-analysis on a set of independent defect detection experiments, but found serious difficulties concerning the diversity of the experiments and heterogeneity of their data sets, and was unable to derive a consistent view on the overall results (Miller, 2000).A noticeable feature in the quality concerns row is that, either directly or indirectly, all these studies are concerned with the quality in use of a DSL, including perspectives such as its effect on the productivity of practitioners, which is sometimes indirectly assessed through the effectiveness and efficiency of the language usage. This is, of course, not surprising, as these examples were chosen precisely because they illustrate how such evaluation can be performed, in different contexts. Kosar *et al.*'s work is an independent evaluation of several DSLs, and is mostly concerned with program comprehension correctness and efficiency while using the DSLs, when compared with using GPLs. A detailed analysis of their data could be used to identify opportunities for improving the tested DSLs. Kieburtz *et al.*'s experiment addresses DSL evolution as part of the concern with flexibility, while the remaining two experiments explicitly look for opportunities for improving the respective DSLs under scrutiny.

The four studies are run *in vitro* (*i.e.*, in the laboratory, under controlled conditions), *off-line*. This context is particularly interesting in that the researchers can better control extraneous factors that would otherwise bring validity threats to each of the experiment. Being off-line, the risks for the organizations where the studies are conducted are also mitigated, in the sense that if anything goes wrong with the experimentation, this will have no visible effect to external stakeholders (*e.g.*, clients that were considering using a DSL). The downside for this is that there are validity threats concerning the realism of an assessment performed in vitro, as well as that of conducting the experiment off-line. Clearly, there are interesting research opportunities to mitigate these threats, by evaluating the same DSLs in a real-world, uncontrolled environment, to strengthen the external validity of the obtained results. The same holds for selection of participants in the experiment, where, whenever possible, real users of the DSL should be involved (as it happened, for instance, with the experienced participants in the PHEASANT experiment).

The number of participants is also an issue, due to the relatively high costs of engaging real users in the validation of languages. Concerning this, we would highlight Kieburtz's experiment as it shows how a meaningful assessment can be performed, even with a very low number of participants (only 4). Of course, for statistical soundness, larger numbers of subjects should be used, but, as noted by usability experts, a small number of users can still detect a high number of usability improvement opportunities in a product (Nielsen, 1993). Using a small number of participants is an interesting option in early evaluations aimed at identifying defects of the language, to reduce costs. In order to draw more definitive conclusions (with high reliability and validity) that state if the language is better than the previous baseline it is necessary to use a larger number of participants. For instance, in Kieburtz's experiment, the conclusions were sound with respect to the participants, but had a threat with respect to their external validity: with only 4 participants, it was not possible to rule out the possibility of their individual skills playing a role in how the competing languages were evaluated. A similar comment might be made for the evaluation experiments described by Murray and Barišić, with 10 and 15 participants, respectively. In isolation, each of these experiments has its own external validity threats. Interestingly, if we combine the results in all these experiments, a consistent pattern of DSL success starts to emerge. Last, but not the least, several of these evaluation experiments uses academic examples for validation, rather than "real-world" problems. This is, of course, a convenience constraint which entails the obvious threat of external validity, if the examples are not representative of the actual tasks real users will have to perform with the DSLs. Even with real-world examples, the (lack of) coverage of the DSL language with those examples is also a common threat.

Criteria	Kieburzt1996	Murray1998	Kosar2012	Barišić2011
Experiment runs	Single	Single	Family of 3 runs	Single
Quality concerns	Flexibility, productivity, reliability, usability	Learnability, understandability, usability, user satisfaction and language evolution	Effectiveness, time frame, efficiency, usability, perceived complexity	Effectiveness, efficiency, self-confidence in results and language evolution
Context	In-vitro, offline	In-vitro, offline	In-vitro, offline	In-vitro, offline
Comparison	DSL vs. GPL	Visual DSL vs. Textual DSL	DSL vs. GPL	DSL vs.GPL
Participants profile	Professionals	Graduate students?	Graduate students	Graduate students
Participants # (DSL/Baseline)	(4/4)	(10/10)	(108/107)	(15/15)
Domain(s)	Messages translation and validation for military command, control, and communications	Object databases query specification	Feature diagrams, graph descriptions, and graphical user interfaces	High energy physics analysis
DSL	MTV-G	Kaleidoquery	FDL, DOT, XAML	PHEASANT
Baseline	ADA templates	OQL (textual DSL)	FD library in Java, GD library in C, Windows form Library in C#	BEE/C++
Tasks/Participant (DSL/Baseline)	(31/31)	(12/12)	(22/22)	(4/4)
Tasks kind	New+Evolution	New	Evolution	New
Materials origin	Industry-level	Academic	Academic	Academic
Pre-test/Interview	Implicit (Yes?)	Implicit (Yes?)	Yes	Yes
Training in DSL	Yes	Yes	Yes	Yes
Training in Baseline	Yes	Yes	Yes	For inexperienced users
Group participants	2 similar groups	4 similar groups	6 similar groups	4 similar groups
Group assignment	Stratified by gender, so that each group 1 woman and 1 man, and each group had a different training order.	Stratified, so that all combinations of programming expertise (programmers vs. non programmers) and training order (Kaleidoquery first vs. last) have 5 elements.	Convenience, based on university courses classes; arrangements made so that half of the participants started learning the DSL first, and then the GPL, while the other half did the opposite.	Stratified, so that all combinations of programming expertise (programmers vs. non programmers) and training order (PHEASANT first vs. last) have a similar number of elements
Independent variables	Language type, participant	Language type, language factor, experience	Language type, domain, question type, experience	Language type, question type, experience
Dependent variables	Effort, effort/task, acceptance test failures, task difficulty classification, and perceptions on flexibility, productivity and confidence.	Correctness, user preferences concerning both languages	Program comprehension, time, efficiency, simplicity of use, test complexity	Time Correctness Confidence scale
Evaluation type (Pre/Eval/Post)	(None/Tool-based evaluation / Questionnaire)	(Interview – implicit, in the paper / Paper and pencil test / Questionnaire)	(Questionnaire/Multi choice Questionnaire/ Questionnaire)	(Interview/Tool-based test/Questionnaire)
Analysis	ANOVA	Paired sample T-test, independent samples T-Test	Wilcoxon Signed Ranks Test	Wilcoxon Signed Ranks Test, Sign Test
Results summary	Increased productivity, reliability, flexibility, with MTV-G. Users preferred its usability to the alternative baseline.	Increased effectiveness and self confidence in results with Kaleidoquery for non-programmers, who clearly preferred Kaleidoquery. No significant difference with programmers, who generally outperformed non-programmers	Increased effectiveness and efficiency of programs written in DSLs when compared to baseline GPL	Increased effectiveness, efficiency, self-confidence in results with PHEASANT, when compared to the baseline C++/BEE. Experts generally outperformed non-experts.

Table 1. Experiments overview.

In all these DSLs, there is a high variability of domains and techniques to build DSLs, suggesting that the lessons learned from this collection of language evaluation experiments should, in principle, apply to DSLs from other domains. All the selected studies compare DSLs with an existing baseline which is, in most cases, a GPL-based solution. The noticeable exception is Murray's experiment, where a graphical DSL is contrasted with the textual notation it is built upon. This illustrates how, in most reported cases, the usability evaluation of DSLs is performed once. In a user-centered design process, this should not be the case. As such, we would expect to find DSL usability assessments covering several versions of the same language, thus supporting the language evolution. Language evolution is covered in some of these studies, usually in the final questionnaire that is prepared for participants, in the end of the evaluation. This feedback can be valuable for language engineers, but the effect of implementing the changes suggested by participants' feedback should ideally also be assessed by a new replica of the experiment, to run with the new version of the DSL.

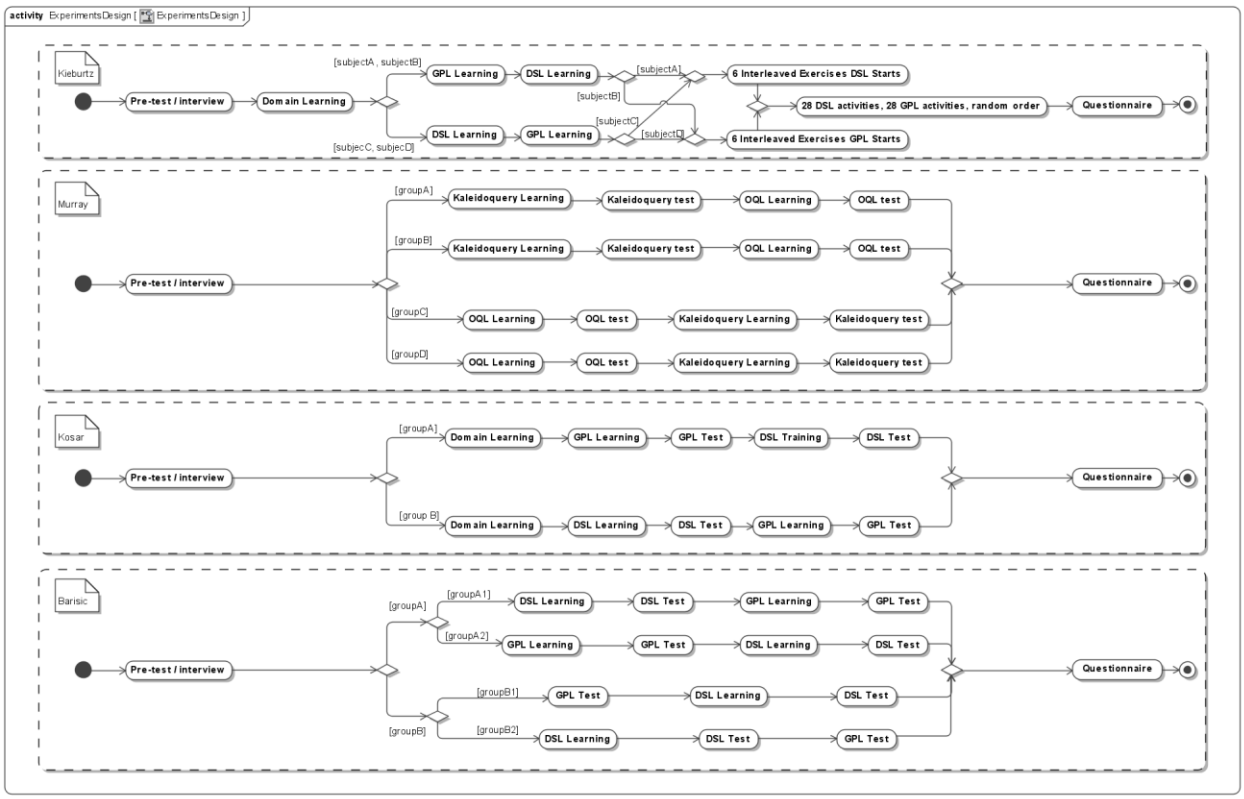

Figure 8. Experiments design: Observations and Treatments.

Concerning the experimental designs (see *Figure 8*), whether implicitly or explicitly, they all report collecting some background information. In some of them, domain training was necessary, while in others, it was not. One of the common concerns in all experiments was to cancel possible learning effects, by splitting participants into at least a couple of groups, so that one of the groups would learn the baseline first and then the DSL, while the other group would have its training and testing path in the reverse order. Whenever more than one category of participants existed (e.g., programmers *vs.* non-programmers), the groups were further split so that there was a balanced number of experienced and non-experienced subjects following each of the training and testing paths. Experiments usually ended with a questionnaire,

so that participant's perceptions on their performance in the experiment, as well as suggestions for improvement in the languages, or other relevant information could be recorded.

All experiments used some statistical approach to assess the extent to which the differences in collected data between using the DSL, or the baseline, were significant. In all cases, some statistically significant differences in the results were reported. These differences should be regarded as indicators of a tendency, rather than as definitive, due to the already discussed external validity issues of these experiments, when considered in isolation, but their overall consistency gives us some trust on the observed trends. In all experiments, the quality impacts of using DSLs *vs.* using the existing baselines are noticeable, and strengthen the claims concerning a stronger usability using DSLs, when compared to their baselines, with an impact on the productivity of professionals using them, in these tests. We also note how, whenever there is a separation among experienced and non-experienced test participants, the improvement effects are more noticeable in the non-experienced participants. The overall feedback, usually collected through a mix of Likert-scale questionnaires (*e.g.*, each answer is encoded in a symmetric scale expressing the level of agreement with a given statement, ranging from a strong agreement to a strong disagreement), and open questions is, in general, favorable to DSLs, or indifferent, but only rarely favorable to the baseline.

The obvious conclusion of all these studies is that, in general, the analyzed DSLs outperformed their baselines, confirming the anecdotal stories on the benefits of DSLs, with varying differences between the baselines and the DSLs. This is not surprising for at least two motives: (i) those DSLs were built to be a better alternative than the baselines they were compared with, in most cases, so the language engineers had a grasp of how to improve on the existing baselines – the DSLs were built to be good at those tasks they were tested with so, the tests showed that this objective was met; (ii) taking a skeptic's view, it is also arguable that, due to publication bias, we are mostly bound to have access to success stories, rather than failure ones. A proponent of a new language is less likely to write a report explaining how the language fails to meet some of its goals, whereas the author of a successful language is interested in illustrating, through validation, the advantages of using the new language. This skeptic's view is a strong argument for the independent validation of claims on DSLs' advantages over existing baselines. That said, it should be noted that Kosar's family of validation experiments is an independent one, in the sense that the evaluators are not simultaneously the developers of the solutions under comparison.

FUTURE RESEARCH DIRECTIONS

The research focus on the problem of Evaluating the Quality of DSLs, so far, has only been scratched very superficially. Although we have presented a way to systematize the evaluation process, so that we could fight its complexity, we need to go further and derive an integrated and effective set of tools to support this phase in a cost effective way. Further research is also required to prove that the user centered design process is a good way to reduce costs in DSL development – we need to organize case studies so that we collect more experimental evidences supporting that claim. We foresee new developments and an emerging community in this area in the near future.

CONCLUSION

Under the perspective of a Software Language Engineer, in order to experimentally evaluate a DSL, we need to know what are the criteria involved, understand the notion of Quality, and understand the evaluation process itself. This is usually complex, and a challenge with respect to reuse, because this is tailored to the specificity of the language under evaluation and its context.

In this chapter, we covered all the aspects mentioned before, and we brought some light to the systematic approach to do so. With general models of DSL's experimental evaluation such as the one we presented in this chapter, the Software Language Engineer is able to effectively reason about his experimental process and eventually detect flaws before it is applied and analyzed.

REFERENCES

Atkinson, C. & Kühne, T. (2003). Model-Driven Development: A Metamodeling Foundation. *IEEE Software, 20*(5), 36-41.

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

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

Barišić, A., Amaral, V., Goulão, M. & Barroca, B. (2011c). *Quality in Use of DSLs: Current Evaluation Methods*. in *3rd INForum - Simpósio de Informática (INForum2011)*, Coimbra, Portugal.

Basili, V. R. (1996). *The role of experimentation in software engineering: past, current, and future*. in *18th International Conference on Software Engineering (ICSE 1996)*. (pp. 442-449).

Basili, V. R. (2007). The Role of Controlled Experiments in Software Engineering Research. In V. R. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl & R. Selby (Eds.), *Empirical Software Engineering Issues. Critical Assessment and Future Directions* (pp. 33-37): Springer Berlin / Heidelberg.

Batory, D., Johnson, C., MacDonald, B. & Von Heeder, D. (2002). Achieving extensibility through product-lines and domain-specific languages: A case study. *ACM Transactions on Software Engineering and Methodology (TOSEM), 11*(2), 191-214.

Bellamy, R., John, B., Richards, J. & Thomas, J. (2010). *Using CogTool to model programming tasks*. in *2nd ACM SIGPLAN workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2010)*, Reno, Nevada, USA.

Bevan, N. (1999). Quality in use: meeting user needs for quality. *Journal of Systems and Software, 49*(1), 89-96.

Bevan, N. (2005). Cost benefits framework and case studies. *Cost-Justifying Usability: An Update for the Internet Age. Morgan Kaufmann*.

Bevan, N. (2009). Extending quality in use to provide a framework for usability measurement. *Human Centered Design*, 13-22.

Brooks, A. (1997). Meta Analysis -A Silver Bullet - for Meta-Analysts. *Empirical Software Engineering, 2*(4), 333-338.

Catarci, T. (2000). What happened when database researchers met usability. *Information Systems, 25*(3), 177-212.

Gabriel, P., Goulão, M. & Amaral, V. (2010). *Do Software Languages Engineers Evaluate their Languages?* in *XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), ISBN: 978-9978-325-10-0*, Cuenca, Ecuador. (pp. 149-162).

Goulão, M. (2008). *Component-Based Software Engineering: a Quantitative Approach.* PhD Dissertation, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, Portugal.

Goulão, M. & Abreu, F. B. (2007). *Modeling the Experimental Software Engineering Process*. in *6th International Conference on the Quality of Information and Communications Technology (QUATIC'2007)*, Lisbon, Portugal. (pp. 77-90).

Hermans, F., Pinzger, M. & Deursen, A. V. (2009). *Domain-Specific Languages in Practice: A User Study on the Success Factors*. in *12th International Conference on Model Driven Engineering Languages and Systems*, Denver, Colorado, USA. (pp. 423-437).

ISO9126. (2001). ISO/IEC 9126: Information Technology - Software Product Evaluation - Software Quality Characteristics and Metrics. Geneva, Switzerland: International Organization for Standardization.

ISO. (2001a). ISO/IEC 9126-1 Quality model.

ISO. (2001b). ISO/IEC 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability.

Jedlitschka, A., Ciolkowski, M. & Pfahl, D. (2008). Reporting Experiments in Software Engineering. In F. Shull, J. Singer & D. I. K. Sjøberg (Eds.), *Guide to advanced empirical software engineering* (Vol. 5971). London: Springer-Verlag.

Kelly, S. & Tolvanen, J.-P. (2000). *Visual domain-specific modelling: benefits and experiences of using metaCASE tools*. in *International Workshop on Model Engineering, at ECOOP'2000*.

Kieburtz, R. B., McKinney, L., Bell, J. M., Hook, J., Kotov, A., Lewis, J., Oliva, D. P., Sheard, T., Smith, I. & Walton, L. (1996). *A Software Engineering Experiment in Software Component Generation*. in *18th International Conference on Software Engineering (ICSE'1996)*, Berlin, Germany. (pp. 542-552).

Kleppe, A. G. (2009). *Software language engineering: creating domain-specific languages using metamodels*: Addison-Wesley.

Kosar, T., Mernik, M. & Carver, J. (2012). Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering, 17*(3), 276-304.

Kosar, T., Oliveira, N., Mernik, M., Pereira, M. J. V., Črepinšek, M., Cruz, D. & Henriques, P. R. (2010). Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. *Computer Science and Information Systems, 7*(2), 247-264.

Marcus, A. (2004). The ROI of Usability. In Bias & Mayhew (Eds.), *Cost-Justifying Usability*: North- Holland: Elsevier.

Maroco, J. (2003). *Análise Estatística - Com Utilização do SPSS* (2nd ed.). Lisbon: Edições Sílabo.

Mernik, M., Heering, J. & Sloane, A. M. (2005). When and How to Develop Domain-Specific Languages. *ACM Computing Surveys, 37*(4), 316-344.

MetaCase. (2007a). EADS Case Study, http://www.metacase.com/papers/MetaEditinEADS.pdf.

MetaCase. (2007b). Nokia Case Study, http://www.metacase.com/papers/MetaEditinNokia.pdf.

Miller, J. (2000). Applying Meta-Analytical Procedures to Software Engineering Experiments. *Journal of Systems and Software, 54*(11), 29-39.

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

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

Nielsen, J. (1993). *Usability Engineering*: Academic Press.

Nielsen, J. & Gilutz, S. (2003). Usability Return on Investment (4th ed.). Nielsen Norman Group.

Nielsen, J. & Molich, R. (1990). *Heuristic evaluation of user interfaces*. in *SIGCHI conference on Human factors in computing systems: Empowering people (CHI'90)*, Seattle, Washington, United States. (pp. 249-256).

Petrie, H. & Bevan, N. (2009). The evaluation of accessibility, usability and user experience. In C. Stephanidis (Ed.), *The Universal Access Handbook*: CRC Press.

Phang, K. Y., Foster, J. S., Hicks, M. & Sazawal, V. (2009). *Triaging Checklists: a Substitute for a PhD in Static Analysis*. in *1st ACM SIGPLAN wokshop on evaluation and Usability of Programming Languages and Tools (PLATEAU 2009)*.

Prechelt, L. (2000). An Empirical Comparison of Seven Programming Languages. *IEEE Computer, 33*(10), 23-29.

Reisner, P. (1988). Query languages *Handbook of Human-Computer Interaction* (pp. 257-280). Amsterdam, The Netherlands: North-Holland.

Sjøberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N.-K. & Rekdal, A. (2005). A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering, 31*(9), 733-753.

Weiss, D. M. & Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*: Addison Wesley Longman, Inc.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. (1999). *Experimentation in Software Engineering: An Introduction* (Vol. 6): Kluwer Academic Publishers.

ADDITIONAL READING SECTION

Benestad, H. C., Arisholm, E., & Sjøberg, D. I. K. (2005). How to Recruit Professionals as Subjects in Software Engineering Experiments. Paper presented at the Information Systems Research in Scandinavia (IRIS), Kristiansand, Norway.

Cao, L., Ramesh, B., & Rossi, M. (2009). *Are Domain-Specific Models Easier to Maintain than UML Models?* IEEE Software, 26(4), 19-21.

Cook, S., Jones, G., Kent, S., & Wils, A. C. (2007). *Domain-Specific Development with Visual Studio DSL Tools*: Addison-Wesley Professional.

Deursen, A. V., & Klint, P. (1998). Little Languages: Little Maintenance? *Journal of Software Maintenance: Research and Practice, 10*(2), 75-92.

Deursen, A. V., Klint, P., & Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 35(6), 26-36.

GME. (2007). GME: Generic Modeling Environment. Vanderbilt University.

Gray, J., Rossi, M., & Tolvanen, J.-P. (2004). Preface. *Journal of Visual Languages and Computing, Elsevier, 15*, 207-209.

Guizzardi, G., Ferreira Pires, L., & van Sinderen, M. (2005). *An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages*. Paper presented at the Model Driven Engineering Languages and Systems (MoDELS'2005), Montego Bay, Jamaica, 691-705.

Johnson, P. (1992). *Human computer interaction: psychology, task analysis, and software engineering*: McGraw-Hill.

Kelly, S., Lyytinen, K., & Rossi, M. (1996). *MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment.* Paper presented at the 8th International Conference on Advanced Information Systems Engineering, CAiSE'96.

Kelly, S., & Pohjonen, R. (2009). *Worst Practices for Domain-Specific Modeling*. IEEE Software, 26(4), 22-29.

Kitchenham, B. A., Dybå, T., & Jørgensen, M. (2004). *Evidence-based Software Engineering*. Paper presented at the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, Scotland, 273-281.

Kitchenham. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering: Keele University and Durham University Joint Report.

Kitchenham, B. A., Al-Khilidar, H., Babar, M. A., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., & Zhu, L. (2008). Evaluating guidelines for reporting empirical software engineering studies. Empirical Software Engineering, 13(1), 97-121.

Kosar, T., Martínez López, P. E., Barrientos, P. A. & Mernik, M. (2008). A preliminary study on various implementation approaches of domain-specific language. Information and Software Technology, 50(5), 390-405.

Luoma, J., Kelly, S., & Tolvanen, J.-P. (2004). *Defining Domain-Specific Modeling Languages: Collected Experiences*. in *OOPSLA Workshop on Domain-Specific Modeling*, Vancouver, British Columbia, Canada.

Moore, W., Dean, D., Gerber, A., Wagenknecht, G., & Vanderheyden, P. (2004). *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. IBM Redbooks.

Seffah, A., Donyaee, M., Kline, R. B., & Padda, H. K. (2006). *Usability measurement and metrics: A consolidated model*. Software Quality Journal, *14*(2), 159-178.

Smolander, K., Tahvanainen, V.-P., & Marttiin, P. (1991). *MetaEdit - A flexible graphical environment for methodology modelling.* in *International Conference on Advanced Information Systems Engineering, CAISE'91*.

White, J., Hill, J. H., Tambe, S., Gokhale, A., & Schmidt, D. C. (2009). *Improving Domain-Specific Language Reuse With Software Product Line Techniques*. IEEE Software, *26*(4), 47-53.

KEY TERMS & DEFINITIONS

Language: a theoretical object (a model) that describes the allowed terms and how to compose them into the sentences involved in a given communication.

Semiotics: the study of the structure and meaning of languages. It is a part of linguistics that studies the dependencies and influences among the following parts: Pragmatics, Syntax, and Semantics.

Syntax: defines what signs can be used in a language, and how those signs can be composed to form sentences.

Semantics: defines the meaning of the sentences of a language. In the case of DSLs, we are interested in languages which have computational meaning, where its semantics is specified by stating how the sentences in such kind of languages can be logically interpreted by a machine.

Contexts of Use: the set of users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used' (ISO, 2001a). It is one of the characteristics that we can use to evaluate a product's usability. In fact, we can use this characteristic to pragmatically distinguish between different products: in DSLs, different languages may have different Contexts of Use (Atkinson & Kühne, 2003). Moreover, if they have different Contexts of Use, then we can infer that the users of those languages

(the humans) most likely will have different knowledge sets, each one with a minimum amount of ontological concepts required in order to actually be able to use each language.

Usability: the extent to which a product can be used by specified users to achieve specified goals – "Goal Quality". It has to be evaluated through the Quality in Use that is perceived by the user during actual utilization of a product in real Context of Use. Achieving Quality in Use is dependent on achieving the necessary External quality, which in turn is dependent on achieving the necessary Internal quality (ISO9126, 2001).

Effectiveness: Usability characteristic that determines the accuracy with which a developer completes language sentences.

Efficiency: Usability characteristic which tells us what level of effectiveness is achieved at the expense of various resources, such as mental and physical effort, time or financial cost, commonly measured in the sense of time spent to complete a sentence.

Satisfaction: Usability characteristic which captures freedom from inconveniences and positive attitude towards the use of a product (in the context of DSLs, the use of a language).