Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Software Languages Engineering: Experimental Evaluation

Pedro Hugo do Nascimento Gabriel

Orientador: Prof. Doutor Miguel Carlos Pacheco Afonso Goulão
Co-Orientador: Prof. Doutor Vasco Miguel Moreira do Amaral

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática.

Lisboa
(2010)

Student Id: 27227
Name: Pedro Hugo do Nascimento Gabriel

Dissertation Title:
Software Languages Engineering: Experimental Evaluation

Keywords:

- Domain-Specific Languages
- Experimental Software Engineering
- Quantitative Methods
- Qualitative Methods
- Threats to Validity
- Usability Engineering
- Systematic Literature Reviews
- Systematic Evaluation Methodology

Resumo

As Linguagens de Domínios Específicos (DSLs) são linguagens que através de notações e abstracções apropriadas, fornecem uma maior abstracção sobre um determinado problema de um domínio para uso mais restrito. A utilização de DSLs pretende contribuir para a melhoria da produtividade do seu utilizador, fiabilidade, facilidade de manutenção, validação e portabilidade quando comparada com a utilização de linguagens de programação comuns. No entanto, como qualquer produto de software, sem a passagem por todas as fases de construção da DSL, nomeadamente Análise do Domínio, Desenho, Implementação e Avaliação, algumas das alegadas vantagens das DSLs poderão ser impossíveis de alcançar com um nível de satisfação significativo, levando potencialmente, em alguns casos, à produção de linguagens inadequadas ou ineficientes. O foco desta dissertação incide precisamente sobre a fase de Avaliação.

De modo a caracterizar o compromisso actual da comunidade das DSLs com a fase de Avaliação, foi efectuada uma revisão sistemática bibliográfica publicada nos principais *fora* dedicados à investigação e desenvolvimento de DSLs. Nesta revisão foram analisados e catalogados artigos publicados entre 2001 e 2008, tendo-se observado uma reduzida preocupação com a fase de Avaliação. Uma das facetas mais relevantes que sobressaiu desta revisão sistemática foi a verificação da ausência de uma abordagem concreta à fase de avaliação, dificultando assim uma correcta aferição dos benefícios reais introduzidos pela utilização destas linguagens. Deste modo, o principal objectivo da dissertação consiste na proposta de uma metodologia para a avaliação sistemática de DSLs. De modo a alcançar este objectivo foi efectuado um levantamento das principais técnicas usadas no contexto da Engenharia de Software Experimental e Engenharia de Usabilidade, por forma a que a metodologia proposta combinasse boas práticas de ambas as áreas. A metodologia proposta foi validada com a sua utilização em diversos casos de estudo, em que a avaliação de DSLs foi feita de acordo com esta metodologia.

Abstract

Domain-Specific Languages (DSLs) are programming languages that offer, through appropriate notation and abstraction, still enough an expressive control over a particular problem domain for more restricted use. They are expected to contribute with an enhancement of productivity, reliability, maintainability and portability, when compared with General Purpose Programming Languages (GPLs). However, like in any Software Product without passing by all development stages namely Domain Analysis, Design, Implementation and Evaluation, some of the DSLs' alleged advantages may be impossible to be achieved with a significant level of satisfaction. This may lead to the production of inadequate or inefficient languages. This dissertation is focused on the Evaluation phase.

To characterize DSL community commitment concerning Evaluation, we conducted a systematic review. The review covered publications in the main *fora* dedicated to DSLs from 2001 to 2008, and allowed to analyse and classify papers with respect to the validation efforts conducted by DSLs' producers, where have been observed a reduced concern to this matter. Another important outcome that has been identified is the absence of a concrete approach to the evaluation of DSLs, which would allow a sound assessment of the actual improvements brought by the usage of DSLs. Therefore, the main goal of this dissertation concerns the production of a Systematic Evaluation Methodology for DSLs. To achieve this objective, has been carried out the major techniques used in Experimental Software Engineering and Usability Engineering context. The proposed methodology was validated with its use in several case studies, whereupon DSLs evaluation has been made in accordance with this methodology.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Software Languages Engineering aims at enhancing the software development process through the usage of Domain-Specific Languages (DSLs). A DSL is "*a programming language, or executable specification language, that offers, through appropriate notation and abstraction, expressive power focused on, and usually restricted to, a particular problem domain*" [4]. The main difference between DSLs and general-purpose programming languages (GPLs) is that DSLs place domain expressiveness first. This allegedly provides productivity and quality improvements, reduces maintenance work, and allows its use to a wider range of end-users (the Domain Experts), since new developers with less experience in programming can effectively develop features more easily [5-8].

Nevertheless, in order to achieve these improvements, on top of all rigorous development work to plan, design and implement the language, it is also necessary to assure that the DSL is adequate to the domain experts. This covers not only the language's correctness, but also language's Usability. It is in this last field, Language's Usability Evaluation, that we think there is a common shortcoming in DSL development projects, where much more must be done.

DSL's Usability Evaluation seems to be considered as waste of time and resources where decision makers seldom involve domain experts during DSL development. In practice, it is as if decision makers prefer to risk using or selling inadequate products, rather than spending resources in evaluating them. A software engineering study based on the usability of software products has characterized this situation by showing that 63% of large projects have overran the initial budget due to usability issues [9]. Although this study was not targeted to DSLs, its results support our concern that poor DSL usability evaluation is likely to lead to relevant economic losses.

We consider DSL usability a quality attribute of major importance for their wide acceptance. In this sense, the DSL community should start a paradigm shift that would lead current DSL development practices from craftsmanship to an Engineering activity.

For this purpose, the Language Evaluation phase should be considered as a key process activity during the DSL development.

1.2. Description and Context

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, and the study of these approaches; that is, the application of engineering to software [10]. Based on this assumption we argue that the DSL community must evolve to achieve the desired maturity and thus get wider acceptance for DSLs. In this sense, our objective consists in supporting this maturity improvement by providing a Systematic Evaluation Methodology for the construction of DSLs.

We borrow influences from two important software lifecycle models, the Waterfall [3] and Spiral [2]. A software development process should be defined in advance for each new product (although we acknowledge Agile Methods' different views on this subject).

The Waterfall Model includes five phases: Requirements Analysis, Design, Implementation, Testing (Validation), and Maintenance. In what concerns the testing phase, it is introduced in a final stage, which in some cases, depending on projects' magnitude, might prove to be late to get final results with lower budget, despite the existence of some retroactivity in each step, as shown in Figure 1.1.

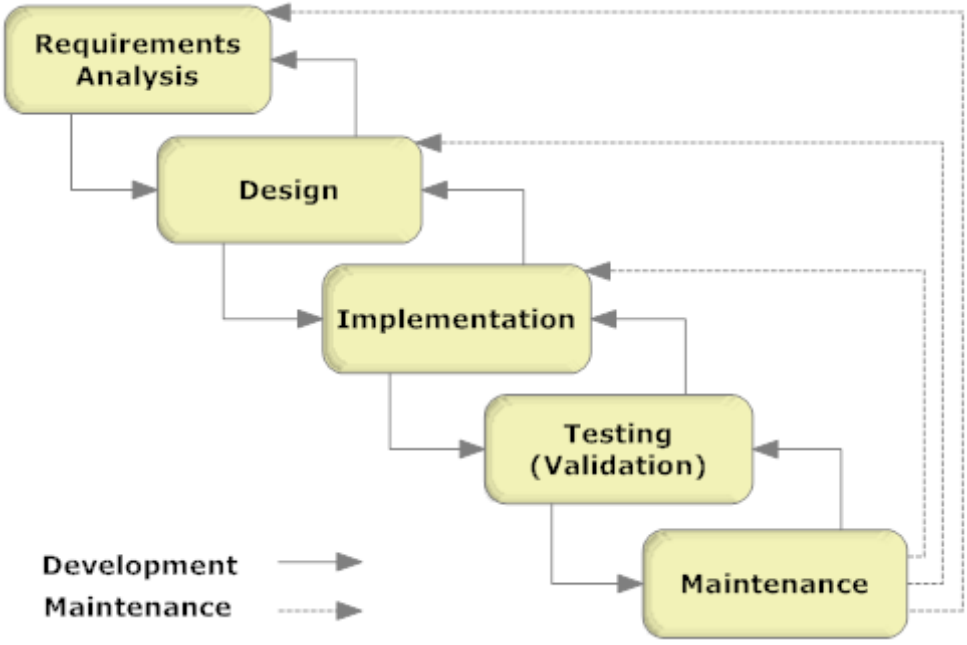

Figure 1.1 – Waterfall Model [3]

On the other hand, the Spiral Model is organized in four stages: Determination of the Objectives, Identification and Resolution of Risks, Development and Test, and, finally, the Planning of the next iteration. This model makes products validation more often, for each new iteration in the development process, since new tests are performed adjusting to iteration needs (Figure 1.2).

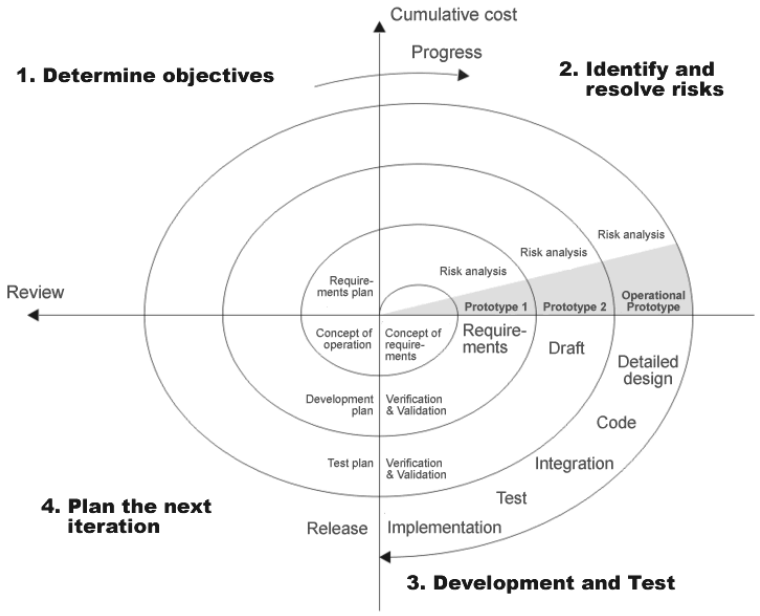

Figure 1.2 – Spiral Model [2]

Both models try to set up a formal development procedure that helps developers to establish their own objectives in each specified phase to improve the final product quality. Another relevant aspect is that both models give considerable importance to the Testing phase in contrast to what is observed in a typical DSL's development process, as is characterized in chapter 5 of this dissertation.

1.3. Scope of the problem

Observing the absence of a well-defined DSL evaluation phase, without clear procedures, assuring that the DSL is adequate to domain experts, as well as satisfies stakeholders' needs, led us to propose a methodology that could change this situation, through a Systematic Evaluation Methodology (chapter 6).

This methodology builds upon previous development models, such as the Waterfall and Spiral models, as well as usability methods and data collection methods from the Usability Engineering and Experimental Software Engineering. Other evaluation

approaches such as metamodel testing [11], or ontologies to assess DSLs [12, 13], are out of the scope of this work, although we recognize their importance to DSL validation.

As DSLs are developed for domain experts, only they can provide a true feedback of a DSL's final quality. Based on this rationale, we found imperative to have them involved in our methodology. In this sense, it is essential to recognize who will use the DSL and what is the domain experts' background in advance. This helps during domain analysis and experiment conduction, since DSL is developed in view of the users, as well as the evaluation material produced. In summary, it enables the achievement of a better and more adequate final product.

1.4. Main Contributions

In the context of the work described in this dissertation, we characterized the DSL community's commitment to evaluate their own DSLs. We conducted a systematic review on DSL evaluation. The review covered publications in the main *fora* dedicated to DSLs from 2001 to 2008 and allowed to analyse and classify papers with respect to the validation efforts conducted by DSLs' producers. Our initial concerns have been confirmed by this review supporting the usefulness of the second contribution of this dissertation: the proposal of a concrete evaluation approach for DSLs.

This Systematic Evaluation Methodology bridges this evaluation gap, where we had in mind factors, such as production costs, and domain experts' satisfaction and productivity. Any additional cost introduced by our methodology, is our sentiment that are retrievable given the better adjustment of the DSL to the domain experts needs.

Finally, the benefits and easiness of application of the Systematic Evaluation Methodology have been measured by comparing its usage against the current ad-hoc approach (chapter 7).

1.5. Document Structure

This document is organized as follows. Chapter 2 provides Domain-Specific Language benefits and current development process. Chapter 3 outlines the current state-of-the-art in Experimental Software Engineering. This includes a brief discussion on Quantitative, Qualitative and Mixed Methods, and the empirical methods validity

assessment. An overview of the usability techniques from the Usability Engineering according to dissertation objective appears in Chapter 4. Chapter 5 presents a Domain-Specific Languages Survey, to understand DSL community efforts to evaluate their own languages, published in *"XIII Congreso Iberoamericano en "Software Engineering" (CIbSE)* [14]. In Chapter 6 we describe the Systematic Evaluation Methodology. Chapter 7 presents the results of the case studies performed in the context of this dissertation to validate our methodology. Finally, in chapter 8 we present dissertation's conclusions, and future work.

2. Domain-Driven Development

Domain-Driven development is an approach to software development which relies on Domain-Specific Languages to raise the level of abstraction, while at the same time narrowing the design space [15]. In order to attain DSLs benefits in its plenitude, a mature domain knowledge must be gathered, otherwise significant losses will be achieved in too late amendments [4].

A Domain, also known as Application Domain, is a bounded field relative to the problem at hand, characterized by a set of concepts and terminology understood by practitioners [16]. In DSLs this domain is more restricted, compared to GPLs, but also more expressive and less generic, which helps to attain a superior level of domain experts' satisfaction, as well as increases flexibility, productivity, reliability and usability [17]. For example, a "Medical" domain is quite large and generic, thus we can "restrict" it to a sub-domain like "Medicines administration management to patients".

Nevertheless, despite all its proclaimed advantages, creating a DSL should be cost-effective; therefore, costs should never exceed the advantages of defining a DSL rather than using a GPL [18].

In section 2.1 we will discuss in more detail Domain-Specific Languages benefits compared to GPLs, and its development process. In section 2.2 we summarize.

2.1. Domain-Specific Languages

Domain-Specific Languages, also known as Micro-Languages [4], or Little Languages [4, 17], *are programming languages or executable specification languages that offer, through appropriate notation and abstraction, expressive power focused on, and usually restricted to, a particular problem domain* [4]. We can usually find them expressed as text or graphic diagrams; however, they can also use representations such as matrices, tables, forms or even trees [19, 20].

Industrial experiences have consistently reported remarkable productivity increases, 5 to 10 times higher than with current development approaches [21] (e.g. Nokia [22], EADS [23]). This is due to DSL's capacity to offer domain experts the domain

abstractions and semantics in a more readily apparent form, allowing experts to work directly with domain concepts [20, 24].

Beyond their valuable expressiveness, DSLs also offer substantial gains in terms of ease of use compared with GPLs, fostering a lower product complexity [25]. Therefore, DSLs' main benefits are [4, 17]:

- Enhanced productivity, reliability, maintainability and portability.
- Programs are concise and self-documenting.
- DSLs facilitate validation and optimization at the domain level.
- Specifications are easier to modify and modifications are easier to understand.
- Domain Experts can understand and validate specifications by themselves.

However, DSLs' usage also has its disadvantages, such as [4, 17]:

- Education costs of DSL end users.
- Difficulty in finding the proper domain or sub-domain for a DSL.
- Potential efficiency loss compared to General Purpose Languages (GPL).
- DSLs may be too expensive when used by a reduced number of domain experts.

DSLs' development process can be described as sequential, passing through four main phases, as depicted in Figure 2.1. The first one, the Domain Analysis, includes the identification of the problem domain by gathering all the underlying knowledge. In this phase, a meticulous understanding of domain experts' tasks and expertise guarantee a more concise and accurate design (see section 4.2.1 for further details).

The second phase is the Design. Here, a two phase job is performed. The Abstract Syntax definition, where it is established the legal relationships between the domain concepts of the DSL, usually made through a metamodel [26], and Concrete Syntax definition, which make clear how the established modeling concepts are represented by visual and/or textual elements [26].

The third phase is the Implementation, where the information from the metamodel is translated into a native programming language, typically using workbench tools such as MetaEdit [27], MetaEdit+ [28], GMF/EMF [29], GME [30], or Microsoft DSL Tools [31]. For instance, in a diagrammatic DSL, each symbol and its underlying relationships

produce certain fixed code. This automated generation saves time and improves consistency. Finally, the Evaluation, or Testing, phase aims to assess the final product, so that domain experts' productivity and satisfaction are meet with good quality standards.

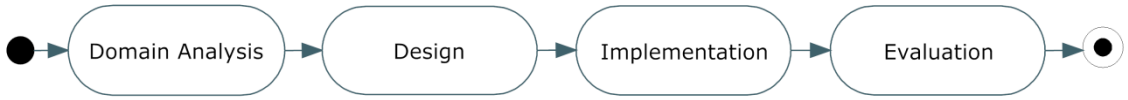

Figure 2.1 – DSL Development Phases [4, 6]

Reaching the end of this process does not mean the work is already done, since new features may need to be introduced in the language. In this sense, Figure 2.2 expresses this state of affairs by defining the DSL development process through a two level model: the Domain Engineering and Application Engineering levels. The Domain Engineering level reflects the work conceived for each DSL development phase, previously presented, while the Application Engineering illustrate the application of the attributes and the feasibility of enhancing the DSL according user requirements.

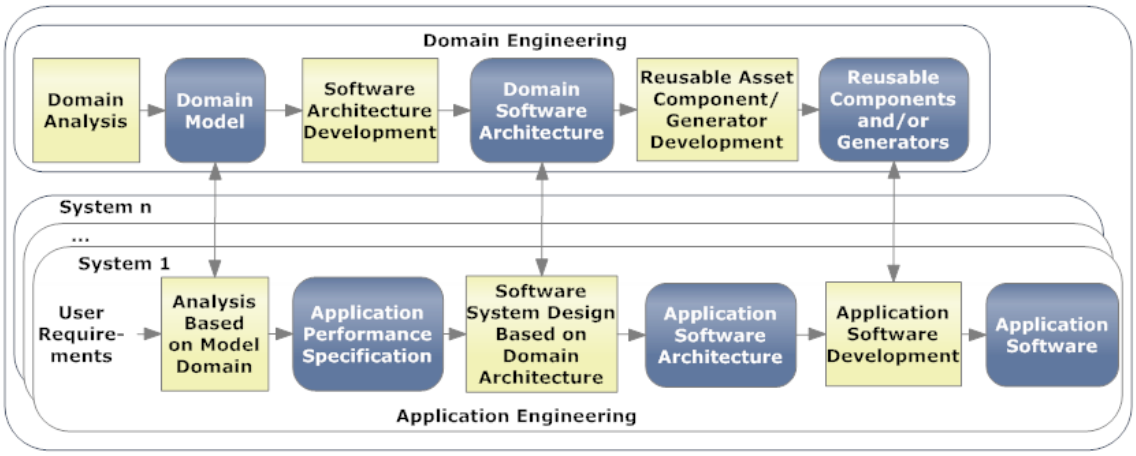

Figure 2.2 – DSLs' Development Process [1]

In our opinion, the Language Evaluation phase should be as important as any of its predecessors in the DSL development process. Nevertheless, among the DSL community (both in industry and academia) there appears to exist a low investment on this phase. This situation might lead to production of inadequate DSLs, wasting important financial and staff resources, and time in later improvements for not fulfilling stakeholder's needs. Consider, for instance, the maintenance costs associated with the

creation and usage of a DSL. Boehm modeled maintenance costs using the following equation [32]:

$$M = F * D * ACT$$

where F reflects an assigned weight to a maintenance factor, such as the type of system, the language used, or the maintainers' experience, among others ([17] lists 14 maintenance factors); D corresponds to applications initial development costs; finally, ACT (Annual Change Traffic) designates the fraction of code changed due to maintenance. After a period of DSL's usage, D will considerably decrease, as well as F, but ACT will continue the same in contrast to a system developed in a GPL. Domain expert involvement in maintenance would be valuable but also introduce costs.

2.2. Summary

Domain-Specific Languages focus on a specific issue from a domain, which comparatively to General Purpose Languages has considerable advantages. However, we cannot always rely on DSLs for reasons of efficiency loss compared to GPL and development costs that are not covered when used by a reduced number of domain experts. Therefore, in any new development, DSLs' advantages and disadvantages should be compared to GPL, to attain the best option concerning costs, expressiveness and stakeholders (Client and Domain experts) requirements fulfillment.

3. Experimental Software Engineering

Experimental evaluation in Software Engineering tries to assess Software Engineering claims through application of a set of experimental techniques. Since the scope of the dissertation focuses in the DSL's evaluation phase and we are interested in assessing DSL's usability, Experimental Software Engineering plays an important role in the Systematic Evaluation Methodology.

3.1. Empirical Methods

Developing a new software product is very challenging and, sometimes, the expected final result does not achieve the required reliability, productivity, etc. To help in the evaluation of these and other aspects of quality we can use solid and well structured experimental evaluation techniques. There are several experimental techniques borrowed from other scientific disciplines, which have been adapted to the validation of claims in the context of Software development. They can be classified into three major categories: Quantitative Methods, Qualitative Methods and Mixed Methods. When Qualitative and Quantitative analysis are combined, the retrieved outcome from the experiment is richer, since we are able to understand more easily the cause-effect of the issues, otherwise it might not be easy to infer a theory [33]. For example, we can achieve a statistical result without knowing the background cause for that.

In section 3.2 through 3.7 we discuss: the Quantitative, Qualitative and Mixed methods; the Threats to Validity; the Reliability/Replication of experiments; finally, we summarize this chapter.

3.2. Quantitative Methods

Quantitative Methods are based on the evaluation of measurable properties from real data, with the aim of supporting or refuting a hypothesis raised by the experimenter

[34]. Our discussion will focus on a particularly representative type of quantitative method: Experiments.

A Controlled Experiment is conducted to test a hypothesis or a set of hypotheses, which can be supported or refuted, but not proven, where the effect on the dependent variables is measured after the independent variables have been manipulated, in order to provide results with statistical validity and identify cause-effect relationships [33], [35], [36]. The impact that a specific selected method or tool, independent variable, has on an attribute of the dependent variable is called treatment [35], where the simplest experimental design has only two treatments (e.g. using a tool vs. not using it). Thus, it is feasible to test different solutions on an attribute of concern by applying each time a different treatment (i.e. a method or tool).

There are two sorts of experiments which are often performed in Software Engineering [36]:

- **Randomized Experiment** – An experiment in which units are assigned to receive the treatment, or an alternative condition, by random process, such as table of random numbers.
- **Quasi-Experiment** – An experiment in which units are not assigned to conditions randomly.

To perform such experiments, it is imperative to have a significant and well-defined sample of the population, so that inferences can be drawn. A thorough data analysis of the population should be held to identify potential anomalies in the data, such as outliers/extreme cases. These types of anomalies raise the risk of bias and consequently can reduce the external validity of results [34]. We will discuss validity threats in section 3.5. The identification of extreme or outlier cases is considered particularly valuable when the source of the collected data is unknown.

3.3. Qualitative Methods

Qualitative Methods focus on qualitative data obtained through observation, interviews, questionnaires, and so on, from a specific population. The data is then catalogued 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, they might help to explain the reasons for some relationships and results, which otherwise couldn't be well understood [34]. Qualitative data is also assumed, by many, to be subjective due to the lack of quantitative information. However, it can be made more expressive and less abstract [34], for example, through structured interviews and questionnaires to subjects. This allows the interviewer/observer to understand what the subject is thinking and the reason for his actions when performing a specific task. In sections 3.3.1 through 3.3.3, we will discuss the qualitative methods, data collection methods and coding qualitative data.

3.3.1. Qualitative Methods Description

When performing a qualitative analysis, experimenter has the possibility to choose the method that best fits to his goals. In section 3.3.1.1 through 3.3.1.3, we discuss three such methods: Ethnographies, Action Research and Case Study.

3.3.1.1. Ethnographies

This method is based on a field observation and studies how people from a particular community interact with each other. In Software Engineering, for example, one can use this approach to study the behavior of java programmers' community. Thus, ethnographic researchers are committed to producing well-structured assumptions that lead them to create local theories to improve their understanding of a particular process or product [33]. However, the lessons learned through this process, might be also useful to outsiders by providing insights in a particular context that can later lead to new hypotheses in other contexts.

3.3.1.2. Action Research

The key point of this method concerns the application of a feasible solution to solve a real-world issue and understand its consequent impact [1]. During an Action research process, researchers plan and conduct possible alternatives in the field with the purpose to improve the situation and then, assess the impact of those changes. Although practical and relatively low-cost, sometimes the desired changes may not be feasible to

apply because of internal organizational constraints, costing issues, ergonomics, etc. Another potential shortcoming of this approach concerns the external validity of findings using it.

Action research is vulnerable to biases of the researcher, particularly from those which are unaware of such biases, since the researcher participates in the area of study and simultaneously evaluates its results. It is, however an often viable surrogate for a stronger form of validation and an interesting option, particularly when external validity of the conclusions is not a priority (e.g. a company is typically more concerned with finding a solution that works well in its own context than with finding a solution that will also work well in other contexts, if finding the latter is less cost-effective for the company and the extra generalization effort brings no tangible benefits for the company).

3.3.1.3. Case Study

Probably the most used qualitative method among researchers. A Case Study can be defined as a field research, since particular subjects, such as an organization, a group of people, or systems are evaluated at point of time. In this sense, techniques of qualitative nature are used to get the data, such as the ones that will be presented in section 3.3.2, offering more and better knowledge about the object of study. In turn, a case study can be in one of the two stages [33]:

- **Exploratory Case Study** – this kind of case study is conducted mostly in the beginning of the research, with the objective of deriving new hypotheses and build theories.
- **Confirmatory Case Study** – this kind of study is used to confirm (or refute) the results obtained in previous case studies, so that an inference can be drawn.

A Case Study might seem similar with Action Research or even Ethnographies but what differentiates it from the others is the fact of being more robust and more feasible to apply to a wide range of cases. For instance, in Action Research, the researcher may bias the outcome of an experiment by acting according to his preconceptions on the expected outcome of the experiment [33], because he participates in the experiment. In Case Study, this in no way happens, because the researcher only conducts, but never participates in the experiment.

3.3.2. Data Collection Methods

When a Qualitative Method approach is chosen, two data collection methods can be used, together or separately, so that one can retrieve as much information as possible. In sections 3.3.2.1 and 3.3.2.2, we discuss those two data collection method: Subject Observation and Interviewing with or without Questionnaires.

3.3.2.1. Subject Observation

In general this kind of data collection focuses the necessity of observing the actions taken by participants, but in some cases it is not imperative [34] (e.g. a participant might perform an activity in a specific tool from which his actions can be retrieved after). In Software Engineering this is slightly different because information gathered through observation has its limits. Often, the most important and interesting information is inside the participant's head. There are some useful techniques to understand what they are thinking at each moment that help retrieving that kind of data [9, 34]:

- **Think aloud** – the participant articulates his thoughts, enabling the experimenter to understand his view of the system (e.g. during subject's keystrokes and mouse movements).
- **Constructive Interaction** – a thinking aloud variation, which involves two participants testing the system together. This allows a more natural dialogue between them, contributing with greater collected data detail for the experimenter.
- **Participant Actions** – records all planned or unplanned communications between the subject and his colleges.
- **Meetings Observations** – the observer must take measures to not be seen and get as much information as possible.

The inclusion of more observers is a measure that should be taken to increase the accuracy of data collection, because there are some cases where only one observer is incapable of retrieving all information from the participants' performed actions. Another alternative is to record a video of the participant's actions, which can then be analyzed off-line.

3.3.2.2. Interviews and Questionnaires

The interviewing process is performed in most cases along with observations and, when both are combined, the obtained results are richer. In terms of interviews they can be [34]:

- **Structured** – the interviewer has a defined set of questions to which the interviewee has to answer.
- **Unstructured** – the objective is to get as much information as possible on a broadly defined topic.
- **Semi-Structured** – is a mixture of the previous ones.

Questionnaires are another useful data collection method. A questionnaire is often performed by retrieving participant general information (e.g. age, gender, experience), using multiple choice questions, scalar questions, which allows the participant judging a statement according to a predefined numeric scale (e.g. agreement or disagreement 1-5), and open questions, where participants write their own considerations about the system.

3.3.2.3. Eye Tracking

Eye tracking is the process of measuring either the point of gaze or the motion of an eye relative to the head. This kind of data collection provides a means to understand the effect the visual elements in a system have to their users, and so identify which elements potentiate mistakes.

A number of methods for measuring users' eye movements can be used, but eye position extraction on video images seems the most popular variant [37]. During this process a wide range of data is achieved. However, is in its interpretation that dwells the hard work, since we have to associate each segment of data with the visual elements visualized by the users at the moment.

3.3.2.4. Log Analysis

When we are interested in capturing users' actions in an automated way of a particular system, for instance, determine which icons and menus have been selected, or

the most used system elements, among others, then the Log Analysis is the right choice. During this automated process a file with quantitative data is being filled. A major problem of this data collection method concerns the analysis of the data, since it only shows what users did but not why they did it. Therefore, it is recommended to combine it with other methods such as Interviews [9].

3.3.3. Coding Qualitative Data

After data collection has been made, using one of the previously discussed methods, it may be useful to extract values from qualitative data, to perform quantitative or meta-analysis. This process is called Coding. This kind of transformation is not always easy and might be subjective, so, it must be performed with caution [34].

3.4. Mixed Methods

There are some cases in which a method cannot be considered purely quantitative or qualitative and while it can be catalogued into one of the categories, we cannot guarantee that it does not use aspects of the other.

In most cases, researchers try to use the best of both worlds with the purpose of gathering more powerful results to consolidate their theory. The strategy behind this approach is using qualitative results to explain/characterize quantitative results from the study [33].

To perform such analysis the researcher must be familiar with both methods.

Of the methods discussed in the previous sections, Controlled Experiments are primarily Quantitative while Ethnographies, Action Research and Case Studies are primarily Qualitative [33] but they might use aspects of the other.

3.4.1. Survey Research

Survey Research has been considered in the mixed category as it includes a balanced combination of qualitative and quantitative characteristics. The conduction of survey research bases on gathering field data. It requires a clear point of departure, the research question, and a well-defined research protocol, to keep in mind the initial objectives, which afterwards will allow us to obtain a more solid theory or explanation

of some issue. A survey can be conducted with or without participants, but when they are included it is typical to use a set of questionnaires as well as interviews for collecting richer data [33]; as noted in section 3.3.2.2, it might also use quantitative methods so that a more powerful inference can be drawn.

3.5. Threats to Validity

Even carefully well-planned, experiments are always vulnerable to some form of validity threat. During an experiment, one or more factors might affect the obtained result thus constituting a validity threat. In some cases, these factors introduce undesirable effects, called biases, putting in risk the internal and external validity. There are some potential sources of biases, such as:

- Conduction of an experiment according the researcher's expectations.
- Misclassification of some parameters
- Wrong subject or paper rejection or acceptance
- Type of subjects
- Non-representativeness of the target population
- Low response rates
- Type of the participants used in the research

All previous research methods are vulnerable to this sort of threats, so it is important to be aware of them. They may compromise the generalization of results. Sections 3.5.1 through 3.5.4 present a brief description of the main threats.

3.5.1. Construct Validity

An experiment is built to support some concept or theory. Construct Validity concerns the generalization of the results of the experiment to the theory behind it [38]. Sometimes, those generalizations may be compromised, due to inconvenient threats which might be reflected in final results. Some common construct validity threats are:

- **Hypothesis guessing** – a pre-conceived idea of the hypothesis result exists, which may influence experimenter behavior, or that of the experiment's participant.
- **Experimenter expectancies** – conduction of the experiment according to the experimenter expectancies.

3.5.2. Internal Validity

Internal Validity is considered the validity of the study itself, with respect to the causal effect being studied. Like the previous validity type, Internal Validity is also threatened by several issues. Some of these inconvenient threats are caused by [38]:

- **Testing** – repetition of the test by the same subjects. This situation can provide different results each time, since subjects have a previous knowledge from the earlier test.
- **Instrumentation** – concerns the artifacts used to materialize the experiment, such as data collection forms (e.g. interview and observation form). When badly designed, instrumentation affects negatively the experiment.
- **Selection** – Depending on the type, number and performance of the subjects that take part in an experiment, results may be different. The method of primary studies selection is another aspect that also influence results.

3.5.3. External Validity

External Validity can be described as researcher's ability to generalize experiment results to software industrial practice [38]. Therefore, External Validity is somehow dependent of Internal Validity quality to get better experiment results generalization [39]. Possible threats of External Validity include:

- **Selection** – without a representative population and subject type, such as students, larger samples but reduce the probability of convince generalization, volunteers, more motivated and capable than others and professionals, easier to extrapolate to industry [33, 36].

- **Setting** – the absence of a very used tool among industry in the experimentation sample.
- **History** – the time or day of a conducted experiment (e.g. a performance questionnaire after a system crash).

3.5.4. Conclusion Validity

Conclusion Validity concerns the ability to extract the right conclusions about the relations between the treatment and experiment's outcome. This is vulnerable to the following validity threats:

- **Low Statistical Power** – reveals the true patter in the data. A low statistical power may provide mistaken conclusions.
- **Violated assumptions of statistical tests** – some statistical tests have certain assumptions (e.g. normal distributed sample), when they are not fulfilled erroneous conclusions are obtained.

3.6. Reliability/Replication

After the experiment has been conducted and results obtained, a concern that all researchers should have is the presentation of all steps taken, so that others can replicate it and confirm the reliability of the process, also referred as a "repeating study", or "replica" [33, 36].

Experimental replication is essential for validation for two main motives: when conducted by an independent team it is a form of external, independent validation; furthermore replicas can be designed to counter the effects of validity threats in the experiments they replicate. Although the replica will have its own threats, the combination of all replicas can mitigate those threats better than an isolated experiment.

3.7.Summary

An experiment can be conducted in many different ways. Some methods are more appropriated than others depending on the objective, scope and intentions of study.

Some may think that quantitative methods are better than qualitative and vice-versa, but the best of both worlds can lead us to better solutions where the weaknesses of one can be compensated by the strengths of the other, so we can argue that quantitative and qualitative methods complement each other [33].

Processes of experimentation can begin with qualitative methods through observation and/or interviewing where after the retrieving data and consistently coding it, the data can be used as input to statistical analysis [34]. During data codification some information may be lost. Qualitative data may be more expressive but it is harder to analyze objectively than quantitative data. In turn when quantitative methods are conducted there is the chance of some results not being well understood, so qualitative methods can help to overcome this situation [39].

Another important issue that must be considered with caution is the active mitigation of potential validity threats. In most cases the low portion of professionals involved as subjects in Software Engineering experiments reduces the ability to generalize results to other industrial contexts [34].

Finally, regardless from the method chosen in the experiment an hypothesis can never be proven, only supported or refuted.

4. Usability Engineering

Any new system must satisfy customer pre-established requirements. Often, these requirements are not fully met in the final product, due to interface miscommunication of specific features to end-users, increasing error-proneness and decreasing productivity. Therefore, performing a system Usability Inspection should be considered an important and desirable mechanism to counter this situation. In DSLs, this should be no different. A DSL with usability problems is likely to lead to productivity losses of DSL users and, consequently, to a potentially lower quality of the products built with that DSL. However, as we will discuss in chapter 5, usability is not among the major priorities of DSL builders.. In this chapter we will discuss Usability Engineering, a domain of Engineering from which there is many lessons to be learned concerning the development of a validation approach for DSLs. In sections 4.1 through 4.3 we provide a usability definition, discuss the usability lifecycle model and summarize by establishing the fundamental steps of the Usability Engineering.

The usability content presented in this section is based on Jakob Nielsen's work [9, 40].

4.1. Usability

Usability is a quality attribute based on users' and/or stakeholders' needs satisfaction by assessing how easy a system is to use, more generally corresponds to the user-friendless or fitness of a product. In ISO 9241-11 [41], usability is defined as "*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*". Normally, usability has several components can be assessed through five usability attributes:

- **Learnability** – the system should be easy to learn so that new users can begin effective interaction and achieve maximal performance.
- **Efficiency** – once the user has learned to use the system, a high level of productivity should be possible.

- **Memorability** – the system should be easy to remember after a period without being used.
- **Errors** – the system should have a low error rate so that users can perform their tasks normally.
- **Satisfaction** – the system should be pleasant to use.

These components allow any Usability Engineer to focus the fundamental aspects that every single system must have and afterwards performing much deeper analysis.

4.2. Usability Lifecycle

Every time a new product is requested, the development team must immediately start thinking the right route to reach the established customer requirements. During this journey punctual Usability Methods should be considered, if a full usability procedure is not possible to be taken. On the other hand, a clear number of test users and evaluators have to be defined as well, to be used among the usability methods. This practice will strengthen the final product correctness. Therefore, to accomplish this objective, Jakob Nielsen outlined eleven usability tasks, which are depicted in Figure 4.1, and will be described with further detail in section 4.2.1 through 4.2.11.

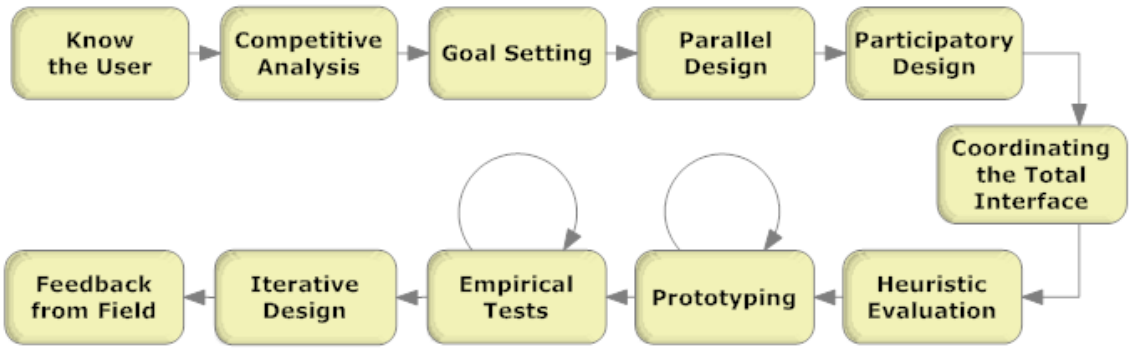

Figure 4.1 – Usability Lifecycle Model [9]

4.2.1. Know the User

This first stage starts by identifying the intended users and their future use of the system. An early definition of their capabilities should be considered in every new

system development, since they provide significant knowledge of the domain and feedback in the way that certain features should behave in the whole. In this sense, a visit to their workspace should be equated, to acquire more consciousness of their tasks. Similarly, the developer is capable to understand what is important and what is disposable, reducing the number of redundant or unnecessary features, as well as prioritizing between them. This process will carry out an interface more familiar and focused to end-users, since they are the real final users of the system, by increasing their productivity and satisfaction.

A significant aspect that should be taken into account during systems' usability development concerns users' capability to improve their efficiency, so it is advisable to use input of any previous experiences.

4.2.2. Competitive Analysis

A new system development may involve starting everything from scratch, which is expensive and exhausting, even to usability engineering. To improve this situation we can learn from the existing products developed in-house or in other companies. In most cases, competing products have passed through reasonable usability tests before arriving to market. Therefore, we can take advantage of this by putting users performing real tasks on them to understand how well their functionality and interaction techniques would be advisable to our new product. This allows having stronger basis to perform all usability lifecycle.

4.2.3. Goal Setting

During DSL development, Software Language Engineers are faced to choose one option over another. Typically, some of these options bring more revenue than others. Goal Setting can be useful in this context. This specifies, through a measurable scale, the minimum and target value to be achieved according to a specific task performed by the end-users. Figure 4.2, depicts an example of its application where, for a particular system, the reasonable number of users errors per hour should be two, although users make an average of 4.5 errors per hour. An early Goal Setting establishment for each desirable task will provide the developer with more guidance to choose the right option, as well as, provides with a more robust and faithful interface to end-users. For new

versions of existing systems or for systems that have clearly defined their competitors, the minimum acceptable usability rate should be equal to the current usability level. For those which are completely new and without any competition, usability goals are much harder to set.

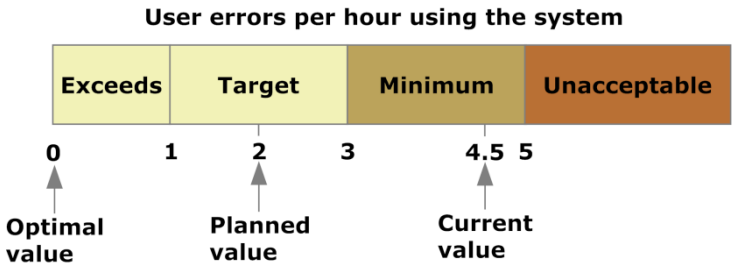

Figure 4.2 – Goal Setting example [9]

4.2.4. Parallel Design

The expected outcome of a system can be achieved through adoption of different design processes. Usually it starts by a designer establishing system requirements and respective functional structure. However, a better result can be accomplished by using several designers in parallel by doing the same procedures. This can be a good option for novel systems, where little guidance is available. As this leads to several alternative definitions of the same system, the best design can be combined with others, providing stronger basis for the remainder of the project. This process should be done with designers working independently, before any proper implementation has been carried out. In Figure 4.3 we present an instance of a parallel design.

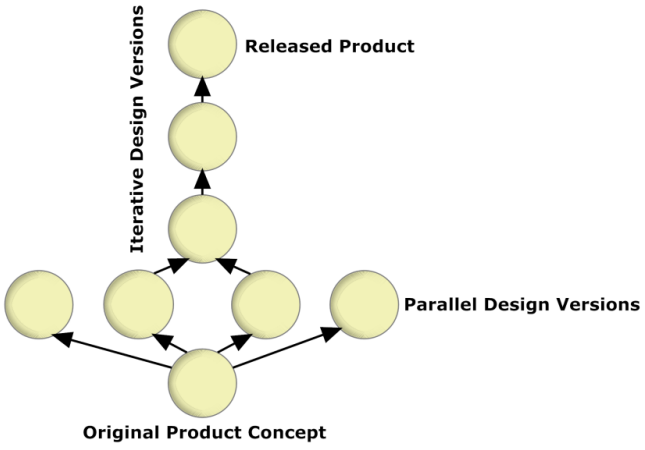

Figure 4.3 - Parallel Design [1]

4.2.5. Participatory Design

A first study of users' capabilities and future tasks has already been performed by Knowing the User, discussed in section 4.2.1. However, we can benefit much more from their presence in the design process, for instance, through regular meetings between users and designers. During these meetings they can provide with new fresh ideas, criticize existing options, which may not reveal as productive as the initial intention, and suggest solutions for improvement of specific features. Despite the considerable advantages of users' introduction in design process, after some period of time they become less representative, as soon as they understand the proposed system structure. So a periodical refresh of the pool of users used in development process should be performed. However, there is a trade-off in refreshing the pool of users: this procedure has a negative short-term impact, since changing users involves spending extra time explaining the project all over again.

4.2.6. Coordinating the Total Interface

During all product development process everyone should be informed with the latest changes, improvements and trends, so that all efforts are conducted in the right direction. In this sense, among DSLs, the software language engineers should communicate the newly introduced feature(s) to the evaluators, so that they could produce, or adequate, the material for the experiment, and evaluators should provide them with the changes that must be carried out after being conducted the experiment, promoting consistency. This consistency shouldn't be measured at a single point in time, but over successive releases, so that each new release is consistent with the previous one.

4.2.7. Heuristic Evaluation

The Heuristic Evaluation for Jakob Nielsen is a usability evaluation method to find usability problems in a user interface design. This evaluation process involves a small set of evaluators examining and judging interface compliance according with fairly broad usability principles, referred as "Heuristics". In widespread Software Engineering

this process is usually referenced as checklist based evaluation. The ten Heuristics for this purpose are:

1. **Match between system and the real world** – the system should speak the users' language, with words, phrases and concepts familiar to the user. Follow real-world conventions.
2. **Consistency and Standards** – similar things should look and act similar, as well as different things should look different, letting users unconfused.
3. **Help and Documentation** – help and documentation should be provided with easy search and understanding.
4. **User control and Freedom** – the user should not feel trapped in his tasks. An "emergency exit" is advised. Support undo and redo features.
5. **Visibility of system status** – users should be informed about what is happening in the system through appropriate feedback.
6. **Flexibility and Efficiency of use** – shortcuts should be included to speed up users' frequent operations.
7. **Error Prevention** – do not give users the opportunity to make errors. Eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
8. **Recognition Rather than Recall** – minimize user's memory load by making objects, actions, and options visible. Users should not have to remember information from one part of the dialogue to another.
9. **Help users Recognize, Diagnose, and Recover from errors** – error messages should be intuitive and in users' language.
10. **Aesthetic and Minimalist Design** – users should be faced with good graphic design, and simple dialogues.

Evaluators' capability to find usability problems is not an exact science and varies from one session (or evaluator) to another. However, since we know that each new evaluator tends to find different issues, we can achieve relatively better final results by incorporating a minimum of three and a maximum of five evaluators, Nielsen suggests. The exact number depends on the cost-benefit analysis. On the other hand, sessions

should not last more than two hours. Longer and complicated evaluations must be divided into smaller sessions, each one focusing on a specific part of the interface. During this process each individual evaluator should inspect the interface more than once by himself. Evaluators should only be allowed to change ideas among them after performing their individual assessments. This procedure produces unbiased evaluations from each evaluator. Then, the respective output corresponds to a detailed list of usability problems, where afterwards, each usability problem found should be classified according to their severity priority, as discussed in section 4.2.9.

This approach is especially valuable: when time and resources are short and makes it possible to develop heuristics to specific classes of products. However, it is deeply dependent on evaluator's experience, although this effect can be mitigated by using evaluators with different expertise in the evaluation team.

4.2.8. Prototyping

When developing a new system, regardless of it being completely new or a new system version, usability engineers recommend not starting a full-scale implementation. This mitigates the risk of having to perform large changes or even starting all over again. Both would lead to an extra time expenditure and project final costs' increasement. Thus, usability engineers often support the construction of system prototypes. In user centered interfaces, this can be achieved by creating a prototype from the intended interface using paper mock-ups or a small computational version of the system. Afterwards, they can be used for evaluation several times. In this sense, there are two prototyping approaches to consider: Vertical Prototyping, which cuts the number of features and includes a detailed functionality definition but only for a few features which are selected for testing; and Horizontal Prototyping, which reduces the level of detailed of functionalities, but allows testing the entire user interface. The latter approach is less feasible.

4.2.9. Empirical Tests

After sufficient features of the system are implemented, the system is put into a real world test. Therefore, a pool of users must be gathered. A typical protocol for the empirical evaluation of usability defines five users as a sufficient number for each

round. Since several rounds might be useful to find more interface design problems, developers can modify the system between each round. The number of usability problems to be found can be approximated by the following expression, where *i* is the number of test users, N the total number of usability problems in the interface and λ the probability of a single test user finding any problem, which varies from project to project.

$$\textbf{Usability\_Problems\_Found}(i) = N\,(1 - (1 - \lambda)^{i})$$

A usability test should be performed in four stages:

- **Preparation** – makes sure that is everything prepared to start the test, including materials, instructions, questionnaires, room.
- **Introduction** – a brief explanation of test purpose is given by the experimenter.
- **Running the test** – users should not be helped unless there is a clear issue in the system observed in previous test users that makes them get stuck and unhappy with the situation.
- **Debriefing** – a semi-structured conversation with each test user where they are asked to fill a questionnaire.

During test running, beyond the useful Subject Observation (section 3.3.2.1), we can collect statistics about system usage by having the computer collecting users' actions, Log Analysis (section 3.3.2.4). This allows developers to track the most used features, users' failures, etc, as a complementary analysis.

After each usability test, it is time to analyse the outcome values. Frequently, some revealed usability problems are much more severe than other. Therefore, a prioritization among them is performed by the evaluators after determining which heuristics have been violated, since it is not feasible to solve all problems at once. In Table 4.1 we present a single rating scale for usability severity.

Severity	Description
0	This is not a usability problem at all.
1	Cosmetic Problem – need to be fixed unless extra time is available on project.
2	Minor Usability Problem – fixing this should be given low priority.
3	Major Usability Problem – important to fix, so should be given high priority.
4	Imperative to fix this before product can be released.

Table 4.1 – Severity Scale [9]

4.2.10. Iterative Design

Each new product development process requires well-defined production guidelines, so that customer requirements and domain comprehension are accurately achieved. However, it is not always feasible to retrieve all domain information at once, since we only notice the importance of some attributes in later stages of the project, taking us back a few times in our development, and new attributes appear with new pretensions of the customer. In most cases new iteration designs are performed to counter these situations. Nevertheless, these new iterations may interfere with other previously implemented components, introducing new usability problems. Therefore, a list cataloguing all changes performed and respective reasons is advisable so that usability principles are not sacrified to attain a minor objective.

4.2.11. Feedback from Field

At the end, the whole usability engineering process performed in the system is revealed publicly. However, there is still much work ahead. A post-development analysis of the system should be performed, by gathering usability data in the field, through end-users experimentation and feedback, for the next version and future products. In this sense, a release can be viewed has a prototype, since several assumptions can be retrieved for future products.

4.3. Summary

Each stage of Nielsen's usability lifecycle model provides with singular aspects to system's quality improvement and users' satisfaction. Nevertheless, it is not always possible to perform full system usability engineering, passing through all the usability methods described in this chapter, due to budget constraints or development deadlines. Even when this happens it is highly recommended to rely on more than one usability method.

5. A Domain Specific Languages Survey

5.1. Introduction

Domain-Specific Languages have an important role in Software Languages Engineering (SLE), since they are said to bring important benefits in productivity, time-to-market responsiveness, and training time when compared to General Purpose Languages (GPLs) [42]. 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. To attain these considerable gains, a typical development process has to be followed. The Domain Analysis starts it, in order to elicit the domain concepts from the pre-establish customer requirements. The language Design follows it, through previous concepts relationship establishment. Then, the language is implemented by constructing the library, typically using a workbench tool for the purpose. And finally, the Evaluation phase assures that the DSL is adequate to the end-user (the Domain Expert).

As with any other software product, we need to assure that the DSL is adequate to the end-user. This covers not only the language's correctness, but also quality attributes, such as language's usability, the maintainability of the produced systems, or the productivity of the developers using the DSL. Deursen *et al.* [17], corroborate the importance that DSL's Usability has on their acceptance and success.

We think there is a serious gap in what language Evaluation should be. In this sense, we present a systematic review to 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 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 literature. The review presented in this chapter aims to fill in this gap. Ultimately, we aim to raise community's awareness to the problem of poor validation of DSLs. This chapter reports on a survey that quantitatively characterizes the description of experimental validation of DSLs in

papers published in top venues from 2001 to 2008. Therefore, from a total of 246 inspected articles, 36 have been successfully selected [18, 43-77].

The followed Systematic Experimental Review Methodology [39], helped us to establish the research questions and a review protocol, which left us tightly connected with the predefined research parameters. This methodology inhibits researchers from contaminating the review with their expectations, reducing thus the likelihood of bias introduction. This contrasts with ad-hoc literature surveys, which are much more vulnerable to researcher's biases. This detail is important for the context of this dissertation. As our purpose is to propose a methodology for evaluating DSLs, we first need to assess the current state of evaluation of DSLs. An ad-hoc review on the topic would provide weaker evidence concerning the lack of proper evaluation in the current state of practice of DSL development. By conducting a systematic review on this topic, we make our survey repeatable and more auditable. This is common practice in other sciences, such as medicine, where evidence-based research has a longer tradition than in Software Engineering.

This survey has been published in *"XIII Congreso Iberoamericano en "Software Engineering" (CIbSE)* [14].

The chapter is organized as follows. In section 5.2 we present the research protocol followed in this systematic review of the current state of practice in SLE. In section 5.3 we present the selection method with the inclusion and exclusion criteria in this review. Section 5.4 depicts the Data Analysis, with the articles that has performed Quantitative and Qualitative analysis, the Subjects' involved in DSL's development and/or evaluation, as well as the Usability criteria, and articles' Replication. Section 5.5 discusses the feasible threats to survey validity and how they were mitigated. Section 5.6 summarizes this chapter.

5.2. Research Questions

Our main motivation was to determine DSL community commitment to the extent of 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 the 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 from the proposed DSLs?

- Is there any evidence that the developed DSL is easy to use and corresponds to end-users needs?

In order to facilitate our characterization of DSLs state of practice 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 any experimentation conducted for the assessment of the DSL?

RQ3: Does the paper report the inclusion of end-users in the assessment of a DSL?

RQ4: Does the paper report any sort of usability evaluation?

RQ5: Does the paper report the DSL development process with some detail?

5.3. Review Methods

Paper selection was performed in two steps. For the first step a direct inspection of paper abstracts and conclusions has been followed, to identify papers covering our research questions. If any doubt remained with respect to the paper's eligibility, we selected it for further analysis. This enabled a more systematic and rapid filtering of articles. In the second step, we followed an in-depth analysis of each of the reviewed papers. To facilitate paper selection, we defined strict paper inclusion criteria, namely: (1) the paper reported on the development of at least one DSL; (2) the paper reported on the experimental evaluation of DSLs; or (3) the paper reported on specific techniques of DSLs Usability evaluation. In what concerns our third criterion we had no success while applying it.

All selected and discarded papers have been inspected from 15 of the most important scientific publications. 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)*, *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, and 10 workshops, *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, *OOPSLA 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.

Table 5.1 presents an overview of the selected papers. We grouped the publications in two categories: publications at least partially targeted to DSLs versus generic Software Engineering publications. 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.

Selection	Publication	Available articles	Inspected articles	Selected articles	Selection percentage
Specific	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%
General	VL/HCC	141	16	2	12.5%
	MODELS	200	4	1	25,0%
	ICSE	42[1]	6	2	33.3%
	TSE	32[2]	2	1	50.0%
	LDTA	10	2	1	50.0%
	Total	763	246	36	14.6%

Table 5.1 – Selected papers

[1]This value corresponds to the number of articles obtained through an advanced search where research keywords have been "Domain Specific Language" from ICSE from 2001 to 2008 in ACM Portal web site, due to considerable amount of articles in the respective conference between the defined date scopes.

[2] This value corresponds to the number of articles obtained through an advanced search where research keywords have been "Domain Specific Language", "Domain Specific Modeling Language", "DSL" and "DSM" in IEEE Transactions on Software Engineering web site, due to considerable amount of articles in the respective journal between the defined date scopes.

The papers from ICSE and TSE have been retrieved with our text-based search, rather than analyzing all the published papers, during the whole time frame. In VL/HCC, MODELS, and LDTA, due to conferences scope being less focused on DSL domain, papers' title served as a mean of selection. In what concerns the DSML conference there is a variation between the available and inspected articles, because 2 of the papers were written in German and were, therefore, discarded from further analysis.

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. Figure 5.1 represents the distribution of these publications during the analyzed period. As we can observe, several of publications considered in our review had a single edition. The most notable exceptions are OOPSLA-DSM and VL/HCC, among the more focused venues, and both of the selected general Software Engineering publications, TSE and ICSE. 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.

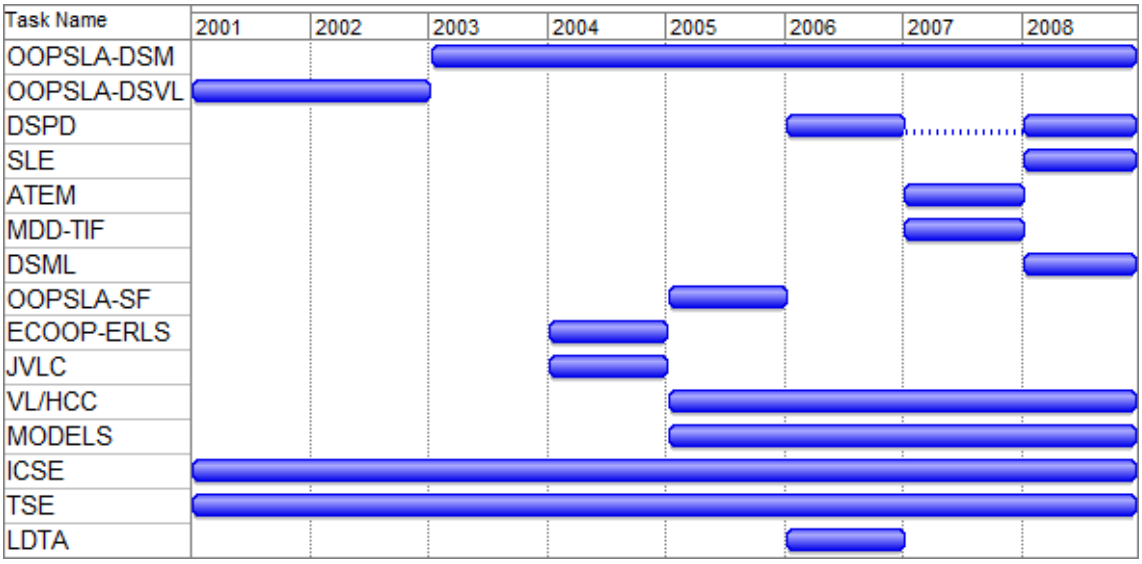

Figure 5.1 – Reviewed publications time frame

During the systematic review many papers addressed DSLs. However, not all fulfilled the research parameters. Some of them have been discarded because they concerned the development of frameworks for DSLs, the application of DSL attributes for other development domains, for instance, code generation and metamodels, or announced case studies where no scientific evidence was provided.

Our survey target was especially publications with a strong concentration on discussions on DSLs, and their creation, although, we are aware that DSLs are widely disseminated through whole software industry. Therefore, it is fair to assume that there are several DSLs addressed in different journals, conferences, and workshops, according to their domain.

5.4. Data Analysis

In this section, we report the obtained results from the selected papers under the defined parameters in section 5.3.

RQ1: Does the paper report the development of a DSL?

In what concerns this subject, a considerable percentage of the total selected papers reports the development of a DSL, 91.7%, to satisfy a specific demand in the real world, while others were presented as proof of concept, targeted to improve a specific domain in software production (e.g. a DSL for Interactive Television applications [50, 62, 63], a DSL for Interoperability between Object-Oriented and mainframe systems, [54]). In turn, the 3 papers which did not reported DSL development were selected for presenting a quantitative analysis to assess domain-specific modeling techniques [72], a qualitative analysis of collected experiences during DSMLs development [76], and usage of usability techniques for DSLs assessment [18]. Table 5.2 depicts the number of selected papers reporting DSL development.

Category	Number of Articles	Percentage
Development of DSLs	**33**	91.66%
Without Developing a DSL	**3**	
Quantitative Experimentation	1	2.78%
Qualitative Experimentation	1	2.78%
Usability Techniques	1	2.78%
Total	**36**	100%

Table 5.2 – Development of DSLs

RQ2: Does the paper report any experimentation conducted for the assessment of the DSL?

After identifying papers reporting DSL development, we were interested to know how many of them have performed any sort of experimental evaluation. In this sense, we grouped these parameters of analysis in two groups: Experimentation Kind, quantitative and qualitative experimentation, and Experimentation Material Kind, Industrial Level and Ad-hoc/Toy Example:

- **Quantitative Experimentation** – papers with quantitative evaluation.
- **Qualitative Experimentation** – papers with qualitative evaluation. This attribute has been divided into subgroups, participant observation, interviewing or not defined to get more understanding.
- **Industrial Level** – papers reporting DSLs tested in industry.
- **Ad-hoc/Toy Example** – papers reporting a DSL implementation as proof of concept.

A Quantitative Method is based on the evaluation of measurable property (or properties) from real data, with the aim of supporting or refuting a hypothesis raised by the experimenter. Qualitative Methods focus on qualitative data obtained through observation, interviews, questionnaires, etc., from a specific population. The data is then catalogued 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, they might help to explain the reasons for some relationships and results, which otherwise couldn't be well understood [34]. In this sense, we obtain the following results, depicted in Table 5.3.

Experimentation kind	Total Articles
Quantitative	3
Qualitative	2
Unknown	21
Without Experimentation	10

Table 5.3 – Quantitative and Qualitative Experimentation

As some of the papers claim performing some sort of experimentation, but without further relevant information to the reader according the kind of evaluation, we add the category Unknown. In turn, some of them report no experimental evaluation, Without Experimentation.

The first noticeable information concerns the few papers performing experimentation validation of DSLs, five in a universe of 36 selected papers. In what concerns quantitative experimentation, only 3 articles has performed it, [65, 67, 76], but without a reliable evaluation process. Merilinna *et al.* [65], report on a comparison between two different approaches, the "traditional software implementation" versus using a DSML. In Zeng *et al.* [67], a DSL dataflow analyzer has been performed to analyze programs as part of the compilation process, where the experimental evaluation focused on a comparison in lines of code (LOC) between the "traditional" method and DSLs generated code. Finally in Bettin *et al.* [76], once again a LOC comparison has been done, but this time between "traditional software development", UML based software development and DSL software development based. An atomic model element was introduced to measure the effort of production, but no further evaluation was conducted to give wider scientific evidence.

The two papers reporting qualitative experimentation are [64, 72]. In Luoma *et al.* [72], a 20 industrial project research using DSMs and MetaEdit+ has been performed. The qualitative data has been 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. On the other hand, Correal *et al.* [64], assess DSM techniques targeted to the definition and improvement of software process models within a software development company, Industrial Level.

In contrast, 10 papers do not report the implementation of any kind of experimental evaluation of DSLs [18, 45, 49, 52, 55, 56, 61, 68, 69, 74].

The remaining 21 papers [43, 44, 46-48, 50, 51, 53, 54, 57-60, 62, 63, 66, 70, 71, 73, 75, 77], report the usage of ad-hoc or toy examples as mean of proofing their concept. However, from five papers performing either a quantitative or qualitative validation, three [65, 67, 76] provide no details concerning the kind of examples used in their evaluation, and two [64, 72] declare to use industry-level examples. In turn, [72] goes further and claim to have obtained their information at industrial level, but does not provide details on the particular evaluation. Table 5.4 summarizes this information.

Experimentation material kind	Total Articles
Ad-hoc/Toy Example	21
Industrial Level	2
Unknown	3
Without Experimentation	10

Table 5.4 – Ad-hoc/Toy Example and Industrial Level Experimentation

RQ3: Does the paper report the inclusion of end-users in the assessment of a DSL?

When developing a new system, regardless of being completely new or a new system version is important to make an exhaustive study of 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 [9]. This observation is applicable to software users in general, and to DSL users in particular. In order to characterize DSL users who participate in a DSL evaluation, each paper was inspected concerning three categories:

- **Industrial or Specialized personnel** – articles reporting subjects with expertise in the domain. He doesn't necessarily need to have knowledge about DSLs, in general.
- **Academic** – usage of students as a surrogate for real end-users of a DSL.
- **Not defined** – when were revealed the usage of subjects but not the profile.

From the 36 selected papers only 5 have explicitly reported the usage of subjects. Three of them reported using domain experts, including seismologists [51], and other specialized developers [67, 72]. The remaining two papers didn't specify subjects type in the evaluation of the DSL [55, 64]. Once again all available data was not carefully disclosed. Table 5.5 depicts the retrieved information.

Domain Experts	Total Articles
Industrial or Specialized personnel	3
Academic	0
Not defined	2
Unknown	31

Table 5.5 – Domain Experts usage

RQ4: Does the paper report any sort of usability evaluation?

Usability is a quality attribute based on users' and/or stakeholders' needs satisfaction, and concerns how easy a system is to use. In the context of our survey, it was imperative to assess the extent to which DSLs were tested for usability, and whether they fulfill the end user's needs. Thus, we identify three categories for this:

- **Usability Techniques** – the papers report a set of techniques that allow DSLs becoming more accurate to the end users.
- **Ad-hoc** – the paper reports an ad-hoc approach to improving DSLs' usage without a detailed rationale.
- **No Usability Assumption** – The paper provides no information about usability evaluation.

80.6% of the total selected articles revealed no concern in measuring DSL's usability, while 19.4% consider that some options might improve it. In terms of those who spared some time in this issue, 14.3% belong to Usability Techniques category, while the remaining 85.7% correspond to Ad-hoc. Table 5.6 summarizes this information.

The paper that used Usability Techniques has provided a questionnaire to assess a general purpose language, which was afterwards adjusted to the DSLs' context [18]. Papers in category of Ad-hoc focused on visual issues pointed out by subjects, such as layout ideas [55], usage of familiar icons and commands [53], interactive dialogs to increase users performance [44], and the impact that an iterative development with subjects during production has in usability [51]. Finally, [74] has developed three Domain Specific Visual Languages, each one with an intended target user group, where have been undertaken usability trials, without specifying the exact procedure. Consistency and error-proneness have been also compared.

Usability	Total Articles
Usability Techniques	1
Ad-hoc	6
No Usability Assumption	29

Table 5.6 – Usability Techniques reported

RQ5: Does the paper report the DSL development process with some detail?

To understand DSL community commitment concerning evaluation replication, we asked the following question "Does the article report the DSL development process with some detail?". This question helps us characterizing the extent to which authors provide details about the DSLs whose development was detailed in the papers. So, for each paper, we looked for details on DSL construction. From 33 papers reporting a DSL development, 16 provide some in-depth details on how those DSLs are built [18, 46, 48, 50, 51, 54, 55, 58, 59, 62, 63, 65, 67, 70, 72, 73]. The presence of a metamodel was not imperative, for this classification, but in some cases proved to be a good help explaining the developed DSL. The number of DSLs reporting the construction details over the years is depicted in Table 5.7.

Year	Total Articles	Reference to steps
2008	9	4
2007	10	7
2006	7	2
2005	1	1
2004	3	2
2003	1	0
2002	3	0
2001	2	0
Total	36	16

Table 5.7 – Number of articles with reference to the steps taken by year

We can observe that a major percentage of articles reporting DSL development process with further detail occurred in the most recent half of the time considered in this survey.

5.5. Threats to Survey Validity

Even when carefully planned, surveys are always vulnerable to some sort of validity threats, similarly to what concerns experiments, as discussed in section 3.5. However, these threats can be mitigated. In this sense, we have followed a systematic review [39], to avoid biasing the results of this survey with our expectations. Although we were very conservative in our selection, 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 concerns the misclassification of papers. This can happen when the reviewers misunderstand some important information about the paper and classify it in the wrong category. Nevertheless, we mitigated this threat by creating objective criteria to classify the surveyed papers, thus minimizing subjectiveness in the data collection.

A shortcoming in the reviewed work that corresponds to a threat to survey validity is 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 to industry. Most of the publications scrutinized in this review are workshops. Therefore, it may be the case that the predominance of work in progress papers in such venues increases the relative frequency of insufficiently validated claims.

The lack of detail on the surveyed experimentation reports implies that we often do not know who the subjects were involved in the process. This is a threat, as we do not know the extent to which domain experts were really involved in this process, in most cases.

5.6. Summary

We found a low level of experimentation in the surveyed papers. Only about 14% of the papers report to have followed a quantitative or a qualitative evaluation of the DSL, and yet 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 validation of claims supported through experimentation [78, 79]. The proposal of a roadmap for the validation of DSLs could mitigate this shortcoming of current practice. A widely accepted methodology for DSLs validation is our objective, presented in chapter 6.

Most of the publications scrutinized in this review are workshops. Therefore, it may be the case that the kind of chosen examples are in line with what we typically find in workshops, i.e., the presentation of work in progress papers 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. However, the focus of the selected workshops is centered on DSL issues.

In summary, we can characterize that DSL community does not systematically report on the realization of any sort of experimental validation of the languages it builds. Therefore, one of the present challenges to the community is to foster the systematic evaluation of the languages as part of the standard of practice in the development process.

6. Systematic Evaluation Methodology

6.1. Introduction

Domain-Specific Languages are becoming widely used by a growing number of vendors [80]. Industrial experiences have consistently reported remarkable productivity increases by a factor of 5 to 10 times higher than with current development approaches [21].

Nevertheless, in the middle of such promising opportunity, through our Domain-Specific Languages survey, chapter 5, we verified that, in general, DSL producers still neglect the evaluation of their languages, either due to the absence of a concrete evaluation methodology or to a poor perception of DSL usefulness in the development. Therefore, we try to mitigate this problem by providing the Software Language Engineer with a Systematic Evaluation Methodology that can guide him during the evaluation process.

We argue for a systematic methodology, in the sense that provides repeatable procedures, enabling the developer to have a controlled universe built-in with a pre-defined *modus operandi* for each technique, reducing the development time, error rate, experimentation biases and, consequently, the production costs.

DSLs development process is established through the analysis of the problem domain, followed by the design, domain application, and implementation, library construction, and finally ends up with the evaluation, or final testing to assess the final product (as discussed in chapter 2). During this process several iterations are made and several versions of the language are produced until a satisfactory one is achieved. Typically, software language engineers with different degrees of language expertise are involved [81, 82]. Ideally, Domain Experts should also participate in this process as well, since they are a decisive factor for language's acceptance and success [17].

A Verification and Validation of the DSL must be performed. Ensuring that software correctly implements specific functions and satisfies its specification is the mission of the Verification, while determining if the system satisfies customer

requirements is a Validation task. Here our main concern is the Validation, in order that the DSL meet domain experts' expectations and desires, by increasing their satisfaction and productivity.

The remainder of this chapter is organized as follows: Section 6.2 presents the motivation of our work by evincing Systematic Evaluation Methodology foundations. Section 6.3 describes which stakeholders should be involved in the evaluation process. The achieved evaluation procedures can be found in Section 6.4. Section 6.5 summarizes this chapter.

6.2. Motivation

Domain-Specific Languages development is a discipline within Software Languages Engineering. Its application domain is more restricted than GPLs development. Even so, they share several challenges.

In GPLs, customer requirements are usually captured using a generic specification language (e.g. UML). In DSLs, this is often defined in form of a Metamodel with a workbench tool for this purpose [26]. If the metamodel is not self-consistent, easily and correctly read, processed and assimilated, the communication instance fails and brings more difficulties [82], for present and future Domain Experts.

The extreme situation to avoid is to have a DSL engineer doing everything by himself, ignoring other people's expertise, because, without all stakeholders' commitment, the language is unlikely to achieve a good result [19].

On the other hand, some significant problems only arise after the entire development process has been followed, making the language difficult to evolve, understand and use to its full potential, as well as error-prone [83].

In order to counter systematic problems proliferation in languages building, we can borrow approaches from Experimental Software Engineering and Usability Engineering. Some of these approaches, especially in Usability Engineering, are based on iterative designs and usability methods. This allows software language engineers to find errors in advance and reduce future drastic and costly changes in the produced DSLs. On the other hand, with Experimental Software Engineering techniques we are able to retrieve domain experts' impressions of the language and if possible compare them with previous versions of the DSL or previous languages that have already been tested, in in-house or in other organizations in order to understand their significance.

Two leading researchers, make strong claims concerning the importance of using several methods during the Evaluation phase. Jakob Nielsen in Usability Engineering:

> "*Several studies have shown that usability inspection methods are able to find many usability problems that are overlooked by user testing but that user testing also finds some problems that overlooked by inspection, meaning that the best results can often be achieved by combining several methods.*" [84]

and Barbara Kitchenham in Experimental Software Engineering:

> "*…we do not expect a specific method/tool to be the best in all circumstances.*" [85]

Our Systematic Evaluation Methodology builds on this notion of combining several evaluation methods that already exist. Therefore, the choice of the evaluation methods, as well as the evaluation process uses some particular aspects of:

- **Production Costs** – Time and budget constraints, which have an effective repercussion on the Validation Techniques and on the number of software language engineers, evaluators and domain experts used in the evaluation process.
- **Effectiveness** – How well the produced DSL meet stakeholders' objectives: if modeling concepts' are well represented by visual and/or textual elements; and language's comprehensibility, this factor is influenced by domain experts' previous knowledge on: (1) Language Expertise, previous expertise with modeling languages in general, (2) Domain Expertise, previous expertise with similar DSL domains, (3) Problem Size, size of the domain to be modeled, and (4) Type of Task, difficulty of each task, since some tasks are more easily implemented than others.
- **Satisfaction** – The user satisfaction while performing specified tasks. This is influenced by the language intuitiveness - ease of use and ease of learning -, interaction methods, and closeness - proximity between domain expert's mental representation of the DSL and DSL's capacity to satisfy their intentions.

- **Productivity** – How proficient the domain experts performing their specification tasks with the DSL. Whenever possible domain experts' results are compared with previous versions of the language and/or competitive languages.

Given milestones for the Systematic Evaluation Methodology, a roadmap will be provided to be followed by the Software Language Engineers and Evaluators during DSLs' development process.

6.3. Stakeholders

In order to achieve a language well suited for the future users, we identified three classes of actors, each of them with different background and knowledge, as part of the DSL evaluation process: Domain Experts, the end-users of the language, Software Language Engineers, who are responsible for establishing the language, and Evaluators, in charge of setting the evaluation parameters, such as define the number of domain experts to involve in the experiment, produce the evaluation material for domain experts, and examine visual and/or textual elements of the DSL in order to find inaccuracies. Communication among all of these actors is primordial to ensure that each one plays their role efficiently. In Figure 6.1 we depict each actor and associate them to the development phases in which they participate.

In what concerns the first phase of the DSL development process, the Domain Analysis, we identified the domain experts and software language engineers as playing a role in this activity. The domain experts provide their daily tasks, natural capabilities and desires, in which software language engineers take to make DSL planning. This interchange of information will benefit the final DSL, in the way that will help to determine its characteristics, and therefore achieve a more adjusted language to their users' needs, expectations and desires. Domain experts' satisfaction should be one of the main concerns to increase usability standards (as discussed in section 4.2.1).

During Design, software language engineers and evaluators share opinions and envision the future language. Software language engineers are responsible for language definition, whereas evaluators, if necessary, suggest new development directions based on their evaluations of previous versions of the DSL or previous languages that have already been tested.

The Implementation phase concerns DSLs library construction. This is outside our evaluation methodology scope. Nevertheless, the software language engineers are responsible for this task, as depicted in Figure 6.1.

Finally, the Evaluation phase takes place. At this point we considered two primary classes of actors, the domain experts and evaluators. Nevertheless, we are aware that in some cases evaluators' usage may become difficult or impossible due to language development constraints (e.g. budget constraints, or available personnel). In this case software language engineers become part of the evaluation process by performing evaluators' tasks. The domain experts serve as a mean of evaluation for the evaluators or software language engineers depending on each case.

In any case, we still support the presence of an evaluator (or team of evaluators), since a seasoned evaluator is likely to have superior evaluation expertise, and thus is capable to find more inconsistencies and avoid bias during the experiment conduction. This is somewhat similar to having independent quality assurance teams (e.g. testing teams) in "normal" software development.

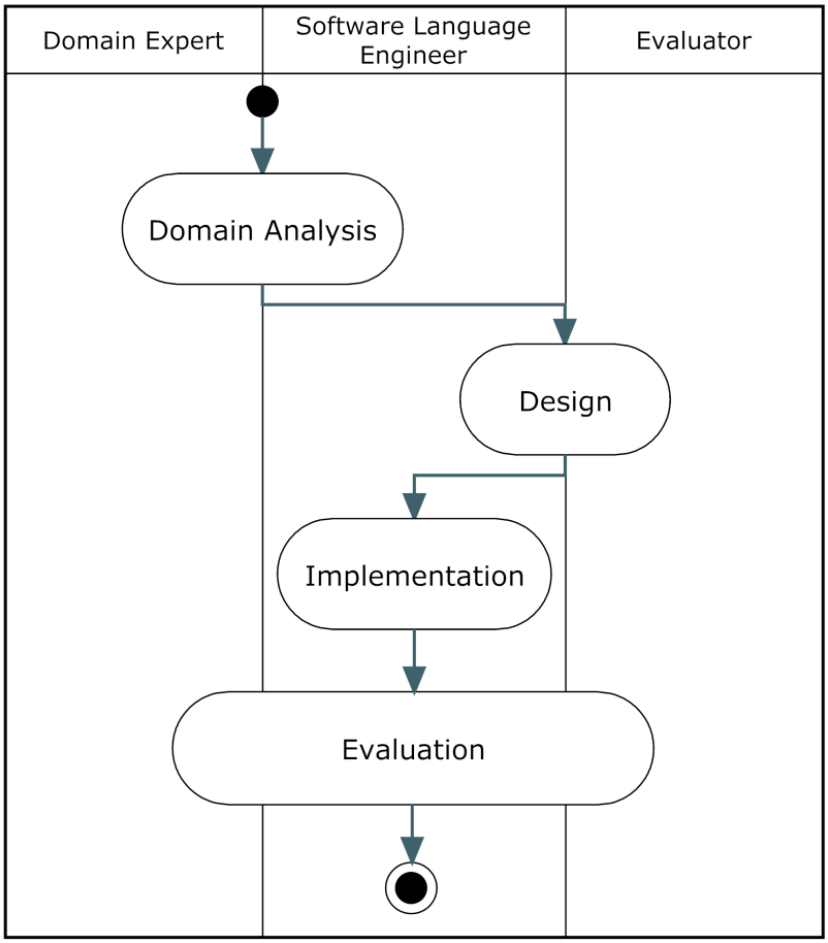

Figure 6.1 – Stakeholders in DSL's development process

6.4. Evaluation Methodology

Regardless of the particular target domain, any DSL should be easy to learn and remember, and useful. This means that it should contain easy-to-use functions. It should also bring efficiency, so that the target domain experts can achieve higher productivity standards, and ensure the satisfaction of their users. Verification, Validation and Testing of the language can help achieving these goals.

Given this and the assumptions in section 6.2, we build our Systematic Evaluation Methodology upon a set of Validation Techniques belonging to specific phases of a typical DSL development process. These techniques are based on Usability Methods from Usability Engineering, discussed in chapter 4, and Data Collection Methods from Experimental Software Engineering, discussed in section 3.3.2. In Figure 6.2, we present a synopsis of the validation techniques adopted during the evaluation of the language, as proposed in our methodology.

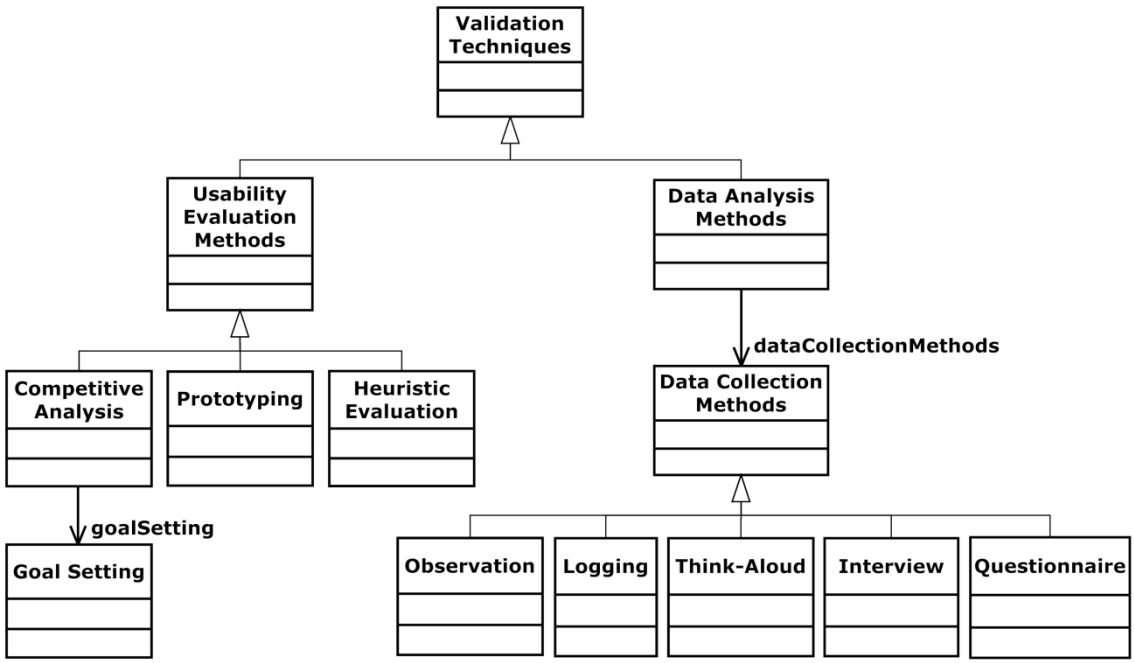

Figure 6.2 – Validation Techniques

The Validation Techniques selected to our methodology are based on the following assumptions: the costs of applying the technique, its capacity to determine domain experts' satisfaction, and productivity, and DSL effectiveness.

Some Usability Methods have been extended to fulfill particular DSL usability criteria, which is the case of the Heuristic Evaluation. Data Collection Methods have

been introduced to obtain useful insights from the domain experts, so that evaluators can provide software language engineers with the right directions to the next step of language development.

In our opinion, two usability methods, Competitive Analysis and Goal Setting, can be used during Domain Analysis. Both will help establishing the baseline for comparison with the results obtained from the language assessment. Regarding competitive analysis, similar languages (DSLs or GPLs) produced in-house, or by a third-party, allow to retrieve development procedures as well as evaluation indicators (e.g. number of errors, task duration, etc). These indicators will serve as basis to the new language's requirements, by helping in the Goal Settings phase.

In the Design phase, we think the remaining techniques should be applied through an iterative process between the development and evaluation of the DSL. This process allows a deeper relationship between design, test and redesign, repeated as often as necessary, to strengthen language Usability [86]. This iterative process is also supported by several articles about DSL construction, such as [5, 87-90]. Nicholas *et al.* [20], outside DSL scope, address this situation by building a model as an ongoing process.

Nevertheless, we are aware that in some cases software language engineers would still prefer to evaluate their language only in a final stage of production. In this sense, our evaluation methodology also satisfies this situation, which matches the particular case of a single iteration of the iterative evaluation process.

We did not assign any validation technique to the implementation phase, as the validation of the implementation is beyond the scope of our work.

In the last phase, the Evaluation, we establish the procedures to prepare, conduct and analyze the experiment. For those who follow our previous recommendations, a competitive analysis and an iterative design, they will benefit from that, since they are able to take advantage of the already produced evaluation material by the evaluators, for the domain experts to experiment the DSL.

In sections 6.4.1 through 6.4.4, we characterize each method and its application, regarding the respective phase of DSL development process.

6.4.1. Domain Analysis

At this moment, before establishing any evaluation procedure, the software language engineer must be aware of: (1) customer demands, to define the crucial and

optional DSL features, (2) the target domain experts, their experience and future use of the language, and (3) customer's time and budget constraints. This last point is of great importance for the evaluator as well, in the sense that it helps him to determine the validation procedures to follow, since some of them are more time demanding and expensive than others [91].

The evaluator should be also concerned with what has been done in-house and/or in other organizations' products that can be used as an asset to the new DSL evaluation. This leads to the selection of our first validation technique, the Competitive Analysis.

Competitive Analysis

Competitive Analysis is helpful for defining a baseline for our evaluation purposes. It uses in-house, or third-party languages (DSLs or GPLs) as a baseline, and then assesses the DSL using that baseline.

Acquiring third-party languages information might become a complex task. In most cases, an extra work has to be done in order to measure the usability factors of interest to compare with the targeted DSL, and get language development details. For that reason, it is often more feasible to perform competitive analysis with in-house targeted DSL, since most languages are not completely new and share more commonalities than idiosyncrasies. This is in line with Product Line Engineering (PLE) [92], and appears to be a recent DSL development trend, since PLE has been contributing to DSL development process. The rationale is to improve the reusability of DSL core assets with SPL techniques [93]. For instance, in Karsai *et al.* [94], the authors promote the reuse of existing language definitions as much as possible, saying that: "*taking the definition of a language as a starter to develop a new one is better than creating a language from scratch. Both the concrete and the abstract syntax will benefit from this form of reuse*".

In summary, DSLs or GPLs with similar domains and development characteristics as the targeted DSL should be analyzed, since much information can be obtained from competing languages and thus considered in DSL's development and evaluation. Languages outside/with similar DSL's target domain and similar/different development characteristics' should not be discarded, as valuable information may be obtained as well. In turn, languages with different domains and development characteristics' from the target DSL are disposable. Table 6.1 summarizes this information.

Competitive Analysis (GPLs or DSLs)	Analyze	Discard
Similar Domain & Similar Development Characteristics	√	
Similar Domain & Different Development Characteristics	√	
Different Domain & Similar Development Characteristics	√	
Different Domain & Different Development Characteristics		×

Table 6.1 – Languages to consider in Competitive Analysis

Competitive Analysis serves two purposes in our Systematic Evaluation Methodology.

The first one corresponds to the identification of good development practices to be introduced in the development of the new DSL. The collectable attributes range from: a simple tool adoption, by identifying its advantages and disadvantages, to development procedures, such as the required number of evaluators and domain experts, and the techniques used and its benefits to the final product. In Table 6.2 we present these attributes with further detail.

Collectable Attributes	Considerations
Tool	The tool used to produce the DSL has a large impact on the perception of characteristics of models, their creation and use [19]. If more than one alternative workbench tool exists, we can compare their advantages and disadvantages of adopting one tool over another. Since many tool details only become apparent with their usage, testing each tool is helpful, before deciding which one to use.
Number of Evaluators	The selection of the ideal number of evaluators to be used is not an exact science, although we are aware of typical upper and lower limits, five and three evaluators respectively [9]. With similar competitive languages we may become more accurate on the right number of evaluators to use in the target DSL, since more commonalities than idiosyncrasies are shared.
Number of Domain Experts	Similarly to the number of evaluators the number of domain experts is not easy to define. Therefore, having a baseline for comparison will help in this task.
Validation Techniques	Observing in action the early presented Usability Techniques and Data Collection Methods in competitive

	languages, helps DSL producers to perform a more accurate decision concerning which ones best fit the desired purpose of the target DSL.

Table 6.2 – Collectable Attributes

The second purpose concerns the establishment of the base values of the evaluation elements: the Goal Settings. These values serve as baseline for comparison with the results obtained through domain experts' involvement in the target DSL to measure its usability. Cao et. al [8], provide an example that shows the importance of such results comparison in order to understand their real significance. In their research they compare maintenance tasks using Domain-Specific Models (DSM) and UML Models. Their findings suggest that DSM saves user's time understanding implementation or language issues, granting them more time to model the solution.

In Table 6.3 we present the Goal Settings we found more important to take into consideration during DSL evaluation. It combines inputs from [9, 37, 95-100]. The number of errors made by the users of the language are evinced in [9], their success or failure while performing the tasks in [95, 99], the time needed to perform the task in [37, 96-98], task completion in [37], help request during the experiment in [9], domain experts' satisfaction about the language in [98, 100], and users mental effort with the language in [37]. The list is not exhaustive, in the sense that other goals may be found useful, depending on the DSL scope and objectives. Likewise, this represents a set of alternatives, where for each DSL only the Goal Settings of major interest should be chosen. The provided definitions for each goal setting are possible examples of their usage. However other solutions may be established.

Targets for Goal Settings	Definition
Number of Errors	The average number of users' errors while performing a task. In some cases it may be useful determine the number of users' errors while performing a task during a specified period of time (e.g. number of users' errors per hour).
Task Success or Failure	A percentage of the domain experts that successfully performed the task and of those who were unable to accomplish it.
Task Duration	The average duration that domain experts spend to

	complete the task.
Task Completion	The percentage of a task completion by the domain experts. This percentage might be determined with respect to a pre-defined duration assigned to the task.
Help Request	The average number of help requests that domain experts perform to accomplish the task. When establishing this value, it is important to understand if in competitive languages a specific duration for the task has been assigned, otherwise we may compare results with different characteristics.
User Satisfaction	Domain experts ease of use, learning and comprehension of the DSL. This can be measured by a questionnaire where the questions for that purpose are answered through a Likert scale.
Mental Effort	The necessary mental effort that a domain expert has spend to perform a specific task. This can be measured by a questionnaire where the question for that purpose is answered through a Likert scale.

Table 6.3 – Goal Settings

6.4.2. Design

After performing the Domain Analysis it is time to Design the future language according the pre-established domain concepts. In this sense, the Abstract and Concrete Syntax have to be defined.

In what concerns the Abstract Syntax, the modelling concepts' and their legal relationships are established. On the other hand, the Concrete Syntax makes clear how the established modelling concepts are represented by visual and/or textual elements.

The abstract and concrete syntax are key to the success of the DSL. Therefore, a poor definition and evaluation of the entities relationships and visual and textual elements, will force the domain experts to use an inadequate language. In this sense, we argue for an iterative design between development and evaluation of the Abstract and Concrete Syntax, as the one depicted in Figure 6.3. However, for those who still prefer to conduct a single evaluation, this is also achievable with a single iteration of the iterative evaluation model.

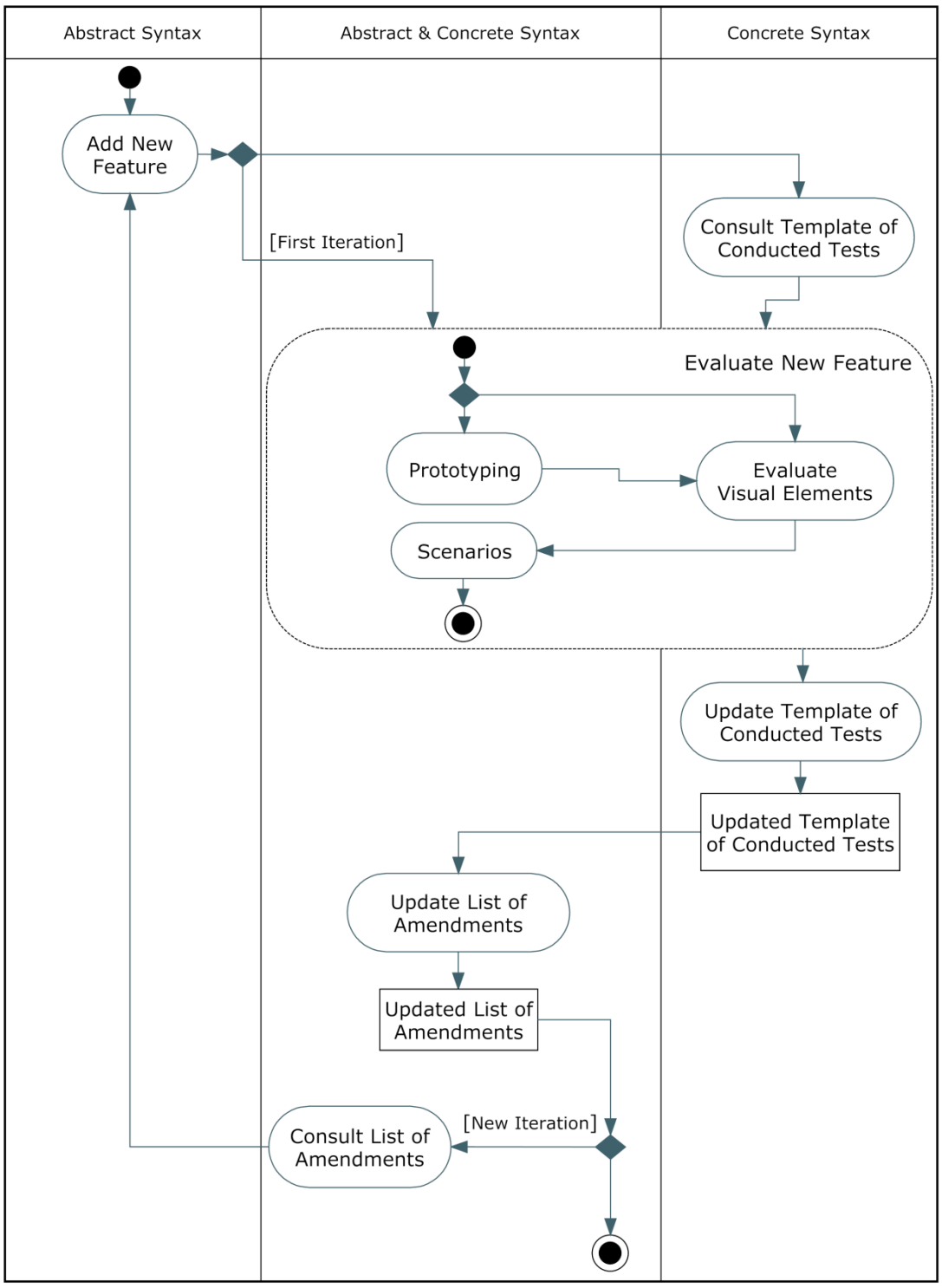

Figure 6.3 – Iterative Evaluation Model

The process should start by modelling a small subset of the language and then test it. If the result from the experiment is not satisfactory, the necessary changes should be performed and stored, as well as the conducted tests, in order to be used in future for comparison. Then the process continues until the desired language is accomplished. This kind of evaluation process is also supported by [88, 89].

Each procedure presented in Figure 6.3 will be described in this section with further detail. In turn, the necessary steps to prepare, conduct and analyze the experiment will be the same as the evaluation process described in section 6.4.4.

Prototyping

Developing a new DSL is a hard and demanding work. Domain concepts and customer requirements usually change has language evolves, and with that several adjustments have to be done. Regarding a short-term DSL development, i.e., a language that is under development and has not yet used for any circumstance by its intended users, the sooner these adjustments are carried out, the less time and resources are wasted unnecessarily.

Prototyping is a useful approach in this context. A preliminary version of the intended language is created, unveiling certain details that would be otherwise revealed in later stages with higher associated costs. Steven *et al.* [19], state that software language engineers often view language creation as a waterfall process, neglecting its iterative nature and the need for prototyping, which can result from spacing development milestones too far apart.

Prototyping is associated to both Abstract and Concrete syntaxes, as depicted in Figure 6.3, since they are responsible for language concepts' communication to the domain experts. In this sense, we believe that prototyping is capable to: (1) give an earlier feedback, (2) anticipate the future errors and conflicts; (3) facilitate previewing language's complexity; and (4) help recognizing which integrity constraints should be established, for instance: missing rules, associations, constraints, and cardinality.

Using paper-mocks for prototyping can be a choice, as discussed in session 4.2.8. Another alternative is producing a computational version of the language. In DSLs' scope this could be valuable for a start up project. However, in subsequent versions of the same language, we consider that both approaches may introduce unproductive labour, since the base of the project has already been established and assessed. This statement is somehow sustained by Jakob Nielsen [9], who claims that in early stages of the design where functional prototypes are not yet available, paper mock-ups or simply a few screen designs can be used to prompt user discussion.

Visual Elements

After establishing a new feature, to a start up DSL or to a previous DSL version, the first evaluation procedure to be followed concerns language's visual and/or textual elements, Figure 6.3. But, before setting up our validation process we will make some observations regarding this subject.

The first thing that any software language engineer should be aware, when establishing the visual elements, concerns the extent of use of his DSL, namely whether it is Local or International [101]. This issue plays an important role in how the vocabulary and symbols should communicate the entities.

In a Local environment, restricted vocabulary and symbol metaphors would easily work. However, in an International environment the restricted vocabulary and symbol metaphors solution may not be suitable, introducing a wide range of misunderstandings to domain experts. In this sense, the usage of widely recognizable symbols and text is a good opportunity to overcome this issue. However, the costs associated with introducing this (desirable) redundancy may be too high, when we are only interested to use the DSL within a local context.

When performing the evaluation of a DSL, built for an international community of domain experts, it is useful to gather a sample of multicultural domain experts, in order to guarantee that the visual elements are potentially recognized everywhere.

In spite of the recognized importance of good looking and familiar symbols that are easy to read, remember, understand and use in the DSL, symbol selection does not receive as much attention as it should, as referred in [19]. A choice of symbols with lack of detail can cause ambiguity, and lack of usability [102]. Common shortcomings that should be overcome in symbols used in DSLs include:

- **Complex Bitmaps** – the more detail included, the more difficult it will be to understand the symbols' intension, therefore decreasing usability [102-104].
- **Poorly scaled Bitmaps** – this occurs particularly with aspect-ratio changes and can make symbols hard to understand [19].
- **Subtle distinctions** – too subtle distinctions between symbols may lead domain experts to commit a wide range of mistakes by choosing the wrong symbol [103].

- **Photographic Representations** – the usage of photographs to communicate the concepts. Alan Blackwell [104], has shown through an experiment that his participants did not recognise the photographic images from which they were faced with in a data flow.

In what concerns the textual elements, the DSL should provide terms that are easily understood by any potential end user, avoiding too much mental effort. For instance, using a large number of acronyms may burden the user with unnecessary complexity, particularly if those acronyms are not commonly used in the user's jargon. However, if the DSL target users are quite experienced this may not be a huge problem, as long as they are familiar with the acronyms. Nevertheless, new users with lower expertise may be involved in the future and their performance may suffer from the lack of clarity brought by inadequate acronyms.

Our first validation process builds on the criteria discussed in this section concerning text and symbols, as well as on the heuristics proposed by [40] (already discussed in section 4.2.7), and in Karsai *et al.*'s DSL design guidelines for Concrete Syntax [94].

This validation process consists on having a pool of Evaluators, with a strong background on usability evaluation, appraising the Concrete Syntax in search for potential usability problems. The detailed list of usability problems found by each evaluator should be discussed with the others in order to achieve a consensus in what must be changed. We can regard this process as a special kind of software review [105], targeted to detecting usability problems. If such evaluators do not exist, another element of the company, except the one(s) who developed the feature and set the visual elements, should perform the evaluator's role. Thus, the heuristics that should always be preserved are:

H1. Match domain experts' language – the DSL should speak domain experts' language with one or two words by entity. Follow real-world conventions to wide acceptance.

H2. Error Prevention – Similar words and symbols should be avoided between dissimilar entities.

H3. Minimize domain experts' memory load – avoid too many technical terms and acronyms to identify symbols and entities.

H4. Default values – Symbols with dynamic graphical behaviour should provide default values for input fields.

H5. Help and documentation – help and documentation should be provided with easy search and understanding.

H6. Aesthetic and Minimalist Design – domain experts should be faced with good symbols expressing each entity purpose.

This pre-validation will "clean" the Concrete Syntax from visual and/or text inaccuracies introduced during DSL development process, responsible for diminishing language's usability. Likewise, the pre-validation allows domain experts to find major structural problems during the experimentation phase.

Scenarios

The next step of our evaluation process concerns collecting domain experts' impressions about DSL's new feature correctness. Regarding this, we try to measure DSL capacity to satisfy domain experts' expectations and desires, as well as increase their productivity standards. In order to achieve these assumptions we established a validation process based on Scenarios, as shown in Figure 6.3.

A Scenario describes a sequence of actions that a domain expert should perform by himself on the model, in order to understand if the expected task is achieved in its fullness or something lead him to stop in the middle. If the domain expert cannot achieve his desires this may suggest that the model contains an error or something is not sufficiently perceptive. This last case can be caused by remaining undetected visual elements issues from the previously validation, *Visual Elements* section.

The usage of scenarios to evaluate models is a widespread practice, depicted in Nielsen [9], Nicholas *et al.* [20], Kamandi *et al.* [37], where instead of Scenario they call it Task, and in Richard *et al.* [99], and Jiménez *et al.* [98]. In what concerns DSLs, scenarios are also used as depicted in [98, 99]. In both cases scenarios are part of a set of validation procedures to evaluate their own languages.

A scenario may be Closed, or Open. A Closed Scenario provides a sequence of detailed actions. An Open Scenario provides only the starting and end points, but not a detailed description of any intermediate steps (Figure 6.4).

In Open Scenarios, domain experts are faced with more freedom during tasks execution. Therefore different solutions may be obtained by different domain experts due to language expressiveness. We believe that this Open Scenarios allows detecting missing attributes, such as: (1) missing constraints, (2) cardinality entities, (3) missing rules, and (4) missing associations. In this sense, we propose a major usage of Open Scenarios in detriment of Closed Scenarios during domain experts Examination, as discussed in section 6.4.4.

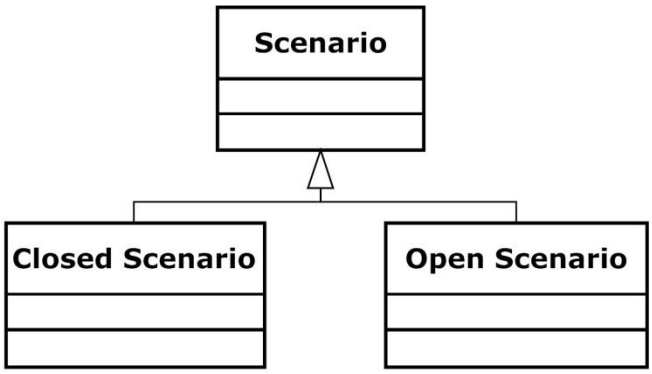

Figure 6.4 – Scenarios

This validation technique can be combined with Data Collection Methods (discussed in session 3.3.2). In this sense, we identified five methods which are useful in this context: (1) Observation, (2), Think-Aloud (3) Log Analysis, (4) Interviewing, and (5) Questionnaire. The first three allow evaluators understand domain experts' actions as the scenario is performed, while the last two are useful for post analysis. A combination of these methods can and should be considered by the evaluators, in order to achieve more significant conclusions of domain experts' actions.

Regarding Observation and Think-aloud, two measurements can be taken: assign at least one evaluator per domain expert while they perform the scenario; or doing it automatically by recording domain experts' session and analyze it in the future. Both approaches allow us to understand domain experts' actions and feelings, where otherwise slightly details would be forgotten.

In what concerns the first option, one evaluator per domain expert, it might represent an overload when too many evaluators are required, which organizations usually do not have. In this sense, two aspects should be retained: less domain experts performing the scenario at the same time requires less evaluators, but more time would be required; more evaluators allows more domain experts evaluated in less time, but more resources are needed.

The second option, record domain experts' session, may overtake the drawback of having too many evaluators. However, it is important take into consideration that this approach will require more time to analyze domain experts' actions and some of the users would not feel very pleased to give his thoughts to a recorder without a human presence by his side.

The Log Analysis, reported to be used by [99], is another suitable data collection method, since it permits to retrieve a lot of statistical data, such as: (1) scenario duration, (2) created and removed entities, (3) documentation access, and (4) entered and removed values. However, it is strongly dependent on the DSL tool to provide this functionality. If a log analysis is feasible, it should be combined with domain expert interviews as characterized by [9], otherwise results interpretation may be extremely difficult and inconclusive, since it only reveals what was done, but not why.

In the end an Interview and/or Questionnaire should be performed, to assess domain experts' feelings about the feature produced for the DSL. This process has been followed by [95, 97, 98]. In what concerns the questionnaire, in [97] the authors strongly recommend engineers to perform a questionnaire as part of their design effort.

Nevertheless, we think that conducting not only a final questionnaire (Table 6.4) but also intermediate ones for each scenario (Table 6.5), is a good opportunity to understand more effectively what went right and what went wrong. These questionnaires, are built on [37, 97, 100, 106], and serve as a start for any new DSL evaluation, since some other feasible questions which are dependent of DSL's scope that can be introduced. The areas of focus in the questionnaires should include:

- **Domain Expert Background** – perceive domain experts' programming skills and experience with DSL modelling tools.
- **Learnability** – the ease of assimilation of the domain concepts and DSL modelling tool functionality.
- **Familiarity** – whether the concrete syntax offers recognizable symbols and text that can be easily understood by the domain expert.
- **Ease of Use** – domain expert impression of the performed scenario, which in turn reflects the ease of use of the DSL.
- **Effectiveness** – the capability of the DSL to enable domain experts to achieve specified tasks with accuracy and completeness.

- **Expressiveness** – how compact and restrictive is the DSL to express our intentions.
- **General Impressions** – overall domain expert impression.

On the other hand, each question in both Final questionnaire and Scenario's questionnaire has been identified at least with one usability factor:

- **Memory Load** – the amount of information that the domain expert needs to memorize in order to perform a specified task.
- **Understanding** – whether the language concepts are easily perceived by the domain expert.
- **Intuitiveness** – Language's capacity to enable domain expert to automatically recognize what and how to achieve a specified task, through a previously learning of concepts.
- **User Guidance** – whether a helpful documentation or supervisor explanation is provided with enough detail.
- **Readability** – whether language's symbols and/or text elements can be easily understood.
- **Attractiveness** – whether language's symbols are good-looking to domain experts.
- **Error-Proneness** – whether the language avoids the domain expert to make mistakes.
- **Controllability** – whether the language give domain experts the sense of control of the environment.
- **Changeability** – whether the language is easily modified by the domain experts.
- **Operability** – the necessary amount of effort to operate and control the language.
- **Accuracy** – Language's capability to provide correct results.
- **Minimal Action** – Language's capability to help domain experts achieve their tasks in a minimum number of steps.

- **Likeability** – Domain experts' perceptions, feelings, and opinions of the language.

The final questionnaire (Table 6.4) is structured in three columns, containing the question itself (Question), the usability factor the question seeks to answer (Factor), and the identifier of the question (ID). In turn, each question belongs to one of the previously presented area of focus (Domain Experts Background, Learnability, Familiarity, Ease of Use, Effectiveness, Expressiveness, and General Impressions).

Factor	ID	Question
Domain Experts Background		
Previous Experience with Programming	B1	Did you have software programming skills? (Yes, No) How do you classify yourself? (Advanced, Experienced, Average, Beginner, Inexperienced) How many years of programming experience do you have?
Preview Experience with DSL tool	B2	How often did you use a DSL tool? (Very often, Often, Sometimes, Seldom, Never) How long have you used it? Have you enjoyed it? (Yes, No) What for?
Learnability		
Understanding	L1	How do you classify the learning concepts? (Very Easy, Easy, Normal, Difficult, Very difficult)
User Guidance	L2	How useful were the provided examples? (Very Good, Good, Satisfactory, Poor, Very Poor)
Familiarity		
Attractiveness	F1	How do you classify the symbols representing the concepts? (Very Good, Good, Satisfactory, Bad, Very Bad) Which of them did you find inadequate?
Readability	F2	How do you identify the text representing the concepts? (Very Good, Good, Satisfactory, Bad, Very Bad) Which of them did you find inadequate?
Readability	F3	How often did you make mistakes due to symbols similarity? (Very often, Often, Sometimes, Seldom, Never)
Readability	F4	How often did you make mistakes due to ambiguous vocabulary? (Very often, Often, Sometimes, Seldom, Never)
Ease of Use		
Controllability	U1	What do you think of the DSL tool? (Very Good, Good, Satisfactory, Bad, Very Bad)
Changeability	U2	How did you feel performing changes? (Very Easy, Easy,

		Normal , Difficult, Very difficult)
Operability	U3	How physically demanding was performing the scenario? (Undemanding, Simple, Regular, Tough, Severe)
Effectiveness		
Accuracy	EF1	Does the outcome reflect what you were expecting? (Totally, Close, Normal, Hardly, Very far)
Expressiveness		
Operability	EX1	What percentage of code you needed to add after DSLs code generation?
Controllability	EX2	How often did you feel unable to express what you intended? (Very often, Often, Sometimes, Seldom, Never) Where did you feel more difficulties?
General Impressions		
Likeability	G1	What is your overall assessment of the DSL? (Very Good, Good, Satisfactory, Poor, Very Poor)
Likeability	G2	What changes or additions do you propose to the model?
Likeability	G3	Do you feel the new system as a value-added compared to the previous one? (Yes, No) Why?
Likeability	G4	Do you feel more productive than with the previous system? (Yes, No) Why?

Table 6.4 – Final Questionnaire

Scenario's questionnaire (Table 6.5) structure is the same as presented in the final questionnaire. The only difference concerns the identifiers assigned to each question. Here we continued to follow the numbering inside each area of focus in order to facilitate questions distinction.

The objective of this questionnaire is to help the evaluators determine more effectively what went right and what went wrong with the domain experts during the tasks they were told to perform, and therefore, achieve more accurate conclusions. These intermediate questionnaires may also serve as mean to perform questions about particular concepts of the language focused in the developed scenarios.

Factor	ID	Question
Learnability		
Intuitiveness User Guidance	L3	How often did you find the need to consult the documentation? (Very often, Often, Sometimes, Seldom, Never)
User Guidance	L4	How often did you perform questions to the supervisor? (Very

		often, Sometimes, Seldom, Never)
Ease of Use		
Controllability	U4	How confident did you feel during scenario execution? (Very confident, Confident, Normal, Insecure, Very insecure)
Error-Proneness	U5	How often did you find trapped or confused during the scenario? (Very often, Often, Sometimes, Seldom, Never)
Memory Load	U6	How mentally demanding was the scenario? (Very Simple, Simple, Regular, Difficult, Very Difficult) What did you feel more difficult to reason/perform?
Effectiveness		
Accuracy	EF2	How do you feel about the correctness of the performed scenario? (Very Good, Good, Satisfactory, Poor, Very Poor)
Expressiveness		
Minimal Action Operability	EX3	How compact did you find the accomplished scenario? (Very Simple, Simple, Regular, Difficult, Very Difficult)
General Impressions		
Likeability	G2	What changes or additions do you propose to the model?

Table 6.5 – Scenario's Questionnaire

Conducted Tests

At this point, DSL's visual elements and domain concepts have already been inspected. During this process data collection methods were used to retrieve domain experts' impressions and results from feature evaluation. In our opinion, supported by [20, 81] where historical record of model testing is seen as good practice, domain experts' impressions, as well as their results have extreme value for next iterations, versions or even new DSLs. In this sense, we consider it is important to record this information for future consultation (Figure 6.3). This historical record will serve: (1) as basis to establish the Goal Settings for a new DSL; (2) as a basis for comparison, to measure domain experts' results significance; (3) as support to establish new tests; and (4) to preview future directions of usability analysis of the language. In summary makes the DSL more accessible to experimentation.

Based on previous assumptions, we defined a template, presented in Appendix 1, to store the results of each group of domain experts. A group is defined as a set of individuals with similar characteristics, for instance, a group of domain experts based on their programming skills, age, experience, etc. This division allows us to perceive difficulties and comfortability within a group, as well as, understand if the same issue remains in different groups. Thus avoids the misinterpretation of the overall results.

The template contains four major areas of interest: Domain Experts Classification, with domain experts' background (e.g. experience, programming skills, etc) and type (e.g. Programmers, Students, etc); the Tool area, contains the name of the workbench tool used in the project; Scenarios' Results, stores domain experts results and their own perception of their performance for each implemented scenario; and General Impressions, where domain experts impressions of the overall DSL are retrieved (Figure 6.5).

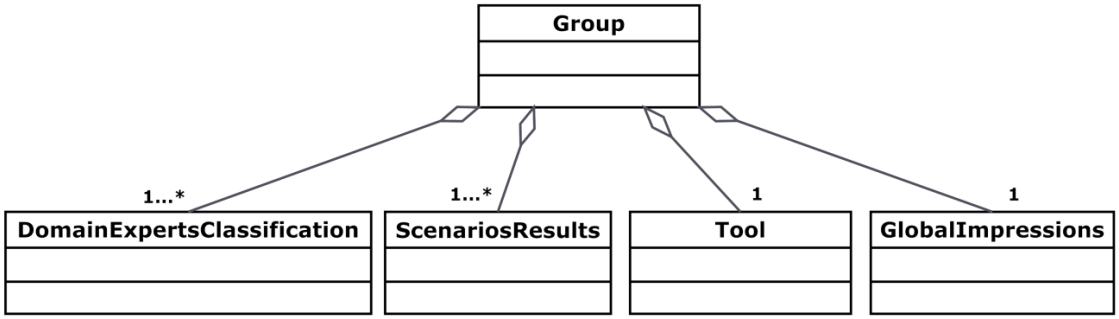

Figure 6.5 – Content of the template with the conducted tests

Amendments

Domain experts have tested and judged DSL's feature in several areas defined by the evaluators. Their tests and results have already been packaged for future uses. Then comes the moment to conduct the necessary amendments according the results obtain through language evaluation, in order to eliminate the remaining flaws and thus increase DSL usability for their users (Figure 6.3).

Once again, we propose in our evaluation approach, supported by [20, 81] where historical record of model transformations and modifications of the language is seen as good practice, to record the amendments performed to the DSL after conducting feature evaluation. We see this approach a valuable mechanism to assist next iterations, versions and even new DSLs to: (1) prevent conflicts between features, since software language engineers are aware of previous approaches that led to conflicts; (2) preview the most common mistakes to be avoided during DSL development; and (3) provide new software language engineers with better documentation, so that they can learn from the experience gained in previous iterations. In summary makes DSL's maintenance easier.

Hence, we strongly recommend the annotation of the amendments, so that not only the current DSL but the future DSLs can benefit with what have been learned in the past. Regarding this factor, we perceive that organizations may be capable to produce a list of good development practices, leading them to establish higher usability standards over time.

6.4.3. Implementation

The DSL is then translated to code. Each textual element, symbol and model relationships will produce certain fixed code to a specific programming language.

This phase is outside our evaluation methodology scope, since it is based on programming skills instead of usability factors. However, we recognize its importance to domain experts overall satisfaction with the DSL. A DSL capable to generate the full extent of the domain concepts will be embraced with more "enthusiasm" than a DSL where it is necessary to write code to achieve the objectives.

6.4.4. Evaluation

The DSL is finally developed. During this process, if our process has been followed, an iterative evaluation has been incorporated within the DSL development process, involving domain experts to experience and judge it. Therefore, at this stage if all steps were strictly fulfilled and every single component of the model tested with sufficient detail, then we think that the DSL is potentially prepared to provide significant satisfaction, productivity and effectiveness to domain experts. However, we should consider that some pending issues may have escaped the software language engineers'or evaluators' attention, and more important, some software language engineers would still prefer to evaluate their DSL only in a final stage of production. In this sense, a final evaluation involving the entire DSL components is always advisable.

Our next step in our systematic evaluation methodology was to identify the procedures to prepare, conduct and analyze the experiment. For this purpose, Figure 6.6 presents an overview of the steps to be taken. The process should be preceded by a pre-validation of language's Concrete Syntax, based on the heuristics presented in *Visual Elements* section. However, if all visual and/or text elements have already been

inspected in previous evaluations, this pre-validation may no longer be necessary. Each of the activities will be discussed in the next sub-sections.

In our model, the deliverables and their relationships are represented with UML 2.0 class diagrams. Some of the activities carried out during the evaluation process have a direct impact on deliverables, or are fed by deliverables produced earlier. In this context, we use three stereotypes to establish the relations between activities and deliverables: `<<read>>`, used when the contents of the deliverable feed an activity; `<<write>>`, used when an activity produces a deliverable; and `<<update>>`, used when an activity updates a deliverable. These stereotypes are not part of the standard UML 2.0 metamodel, but were used in order to increase the expressiveness of our model.

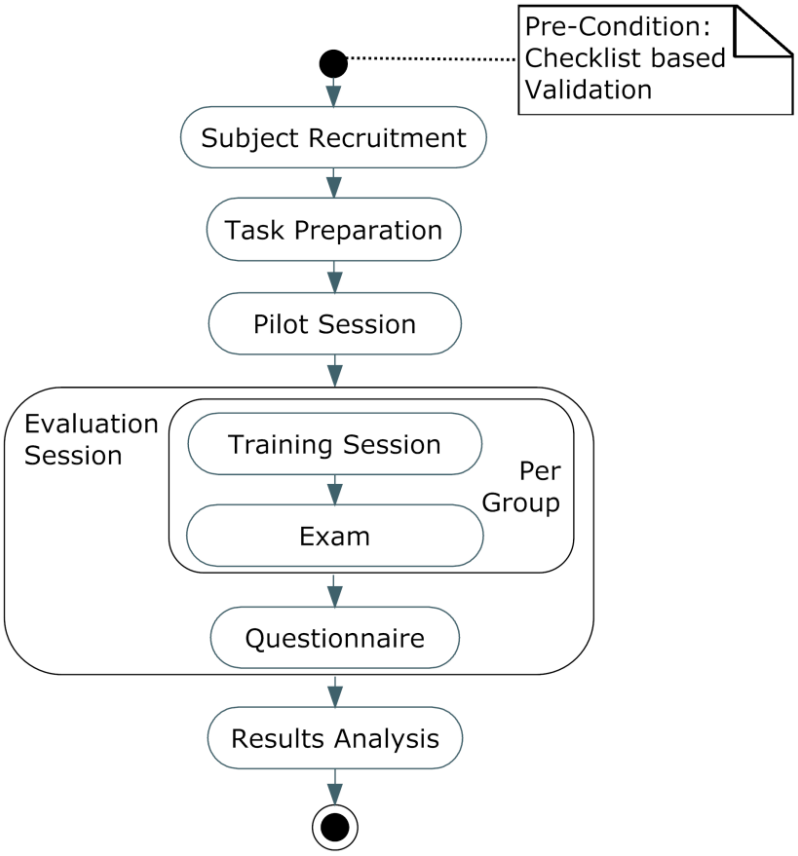

Figure 6.6 – DSL Evaluation Process

Subject Recruitment

The first step is to clearly define the domain experts that will experiment the DSL and group them according similar characteristics (Figure 6.7). Whenever possible,

evaluators should select the same domain experts that will use the DSL. This will provide the most real understanding of how prepared the DSL is to their users. However, this not always happens and students usually take their place [98]. On the other hand, grouping domain experts allows us perceive difficulties and comfortability within each group, which in turn can be compared between groups in order to achieve stronger conclusions, and thus avoiding misinterpretation of the overall results.

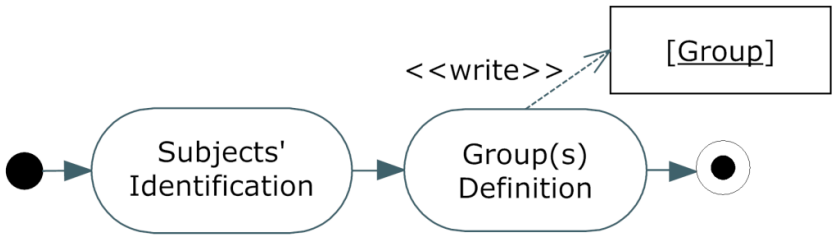

Figure 6.7 – Subject Recruitment

This group division can be performed based on several internal company assumptions. For instance, a possible division can be based on domain experts' prior education and experience, as depicted in Figure 6.8.

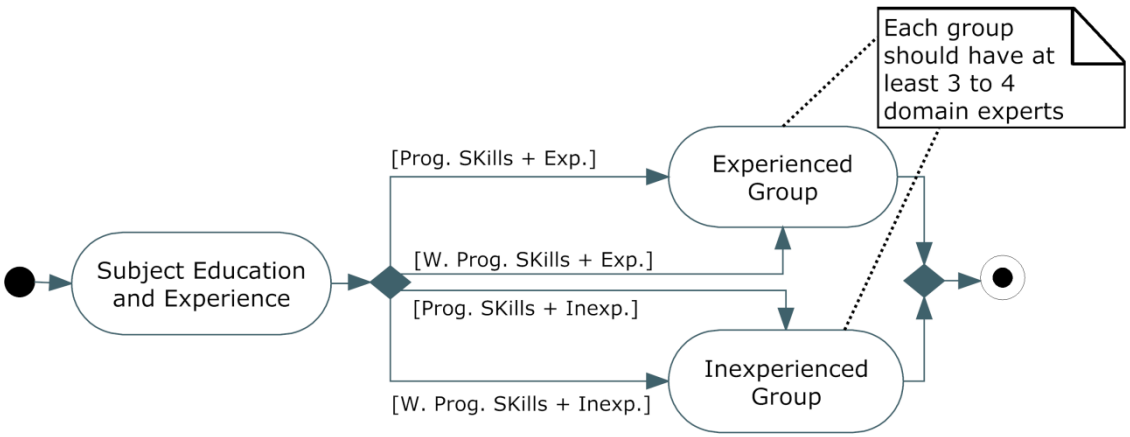

Figure 6.8 – Subject Recruitment Example

Another important issue concerns the number of domain experts to use in the experiment. This topic is enclosed in much controversy since there is not a consensus on the right size of the sample. However, we are aware that the sample size is closely related to the power of the statistical test [38].

Here, our approach to determine the number of domain experts to involve in the experiment is based on a previous formula, presented in session 4.2.9, that is capable to

tell us the amount of usability problems that a particular number of users find in the language. The recovered formula is:

$$\text{Usability\_Problems\_Found}(i) = N\left(1 - (1 - \lambda)^{i}\right)$$

i – The number of test users
N – The total number of usability problems in the interface
λ – The probability of a single test user finding any problem

On this formula we can focus our attention in $\mathbf{1 - (1 - \lambda)^{i}}$. Here **λ** is typically 31% [107], where 5 domain experts will find 85% of the usability problems and 15 domain experts find 99%. At a first glance, it seems that involving 15 domain experts at a single test is the solution, but according to Jakob Nielsen work [107], this is not the case. He states that an iterative evaluation based on three tests with 5 users each is always preferable. This assumption comes in line with our iterative evaluation model, since a feature is implemented, tested, changed, and if necessary, the cycle is repeated once again.

When more than one group of disparate domain experts is defined, has presented in Figure 6.8, the required number of domain experts to experiment the DSL change. In this sense, based on Nielsen [107], with two groups of users, 3 to 4 users should be selected to each group, whilst with three or more groups, at least 3 users should be selected in order to ensure that behavior diversity within the group is covered. Table 6.6 summarizes this information.

Number of Groups		**Number of Domain Experts**
Single Group	Single Test	15
	Iterative Evaluation	Three tests with 5
Two Groups		3 to 4
Three or More Groups		At least 3

Table 6.6 – Number of Domain Experts

Before establishing the number of groups and the number of domain experts for each group, two aspects of major importance must be considered. The first one concerns to domain experts capability to learn with previous experiences. In this context, when three tests with 5 users each is conducted, users from one test should not be used in the

next, unless that is the purpose of the experiment to measure their evolution. The second issue concerns the number of domain experts defined for each group. Since different groups may have different number of users, when this happens we must compare the results between groups with care in order to not misinterpret the results. In this sense, whenever possible define groups with equal domain experts so that a truthful extent of the results can be measured.

In our approach we tried to instantiate only the required attributes, so that resources, time and budget constraints were not wasted. However, we are aware that the number of domain experts established lack of statistical significance, but as presented in Jakob Nielsen work [107], more domain experts in the experiment will not introduce major benefits.

Task Preparation

Then arises the moment to prepare the material for domain experts' exam (Figure 6.9). This process starts by defining the Open and Closed Scenarios. The number of scenarios and their extent depends on what is looked to evaluate. However, they should be sufficient to assure that the attributes to be assessed are effectively used by the domain experts.

Afterwards, takes place a set of tasks in parallel concerning different areas of interest. Updating the already previously presented questionnaires (Table 6.4 and Table 6.5) is one of them. Here new questions can be introduced to adjust it to DSL reality. Then, if necessary, prepare a Structured, Unstructured, or Semi-Structured interview (session 3.3.2.2) to get more impressions of domain experts. Tutorial creation is another important step. Here, Closed and/or Open scenarios can be presented to domain experts to experience the DSL. However, the amount of the information provided and how it is presented should be carefully selected to not influence the exam results. In turn, we think the tutorial should have references to documentation, so that the next time domain experts need to consult it they know where to look for, reducing the amount of time searching in the documentation. Finally, a consent form might be useful in some cases, to inform domain experts of the uses to be made of the data to be collected during the experiment, as followed by Kieburtz *et al.* [99].

The final step concerns the preparation of the data collection tools. We regard this step as optional, as it is not always feasible, or necessary. Nevertheless, we have

identified two possible mechanisms to retrieve domain experts' actions during the exam. They are through Recording Material and/or Log Analysis. These mechanisms can be truly helpful if domain experts are not restricted to a single place, i.e., when we have domain experts spread over the world performing the exam.

In what concerns the iterative evaluation model, much of the developed work to prepare the evaluation can and should be reused between iterations. The scenarios and tutorial only need to be readjusted in order to satisfy new attributes. The questionnaire will not suffer major changes as well as the consent form, since the core of interest has already been established. The interview should be rearranged to focus on the new feature attributes. Thereby, this process will assess not only the new feature but also all previous ones.

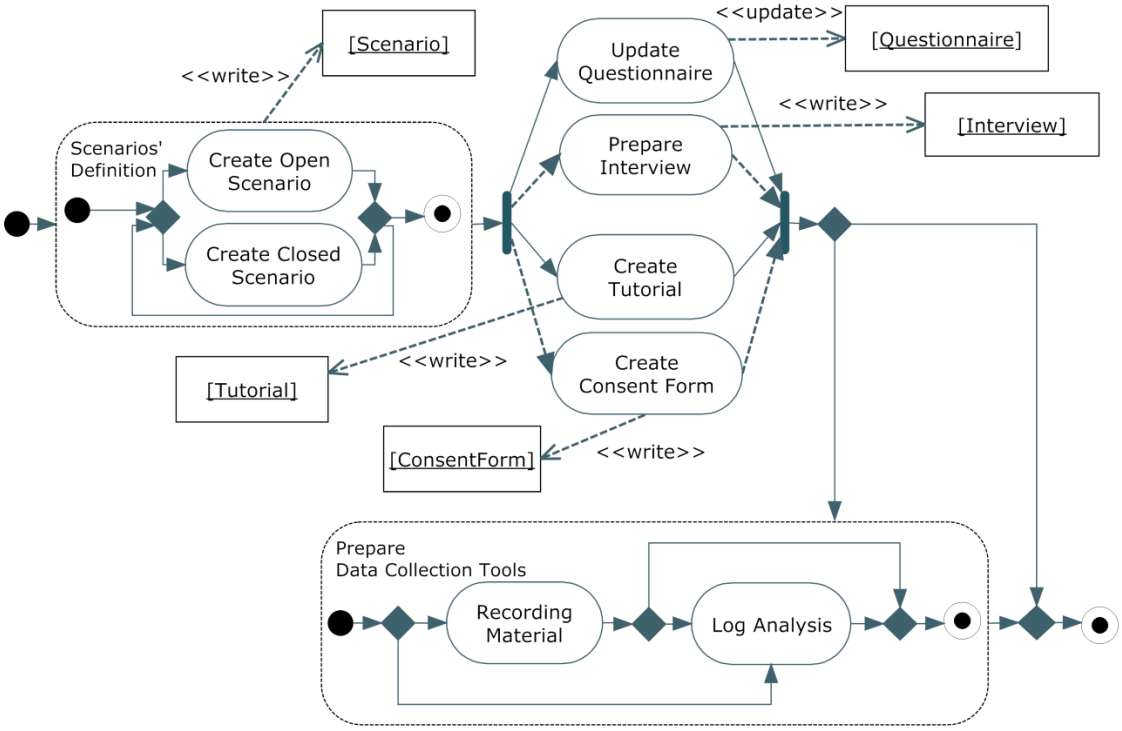

Figure 6.9 – Task Preparation

Pilot Session

This phase intends to simulate the exam, to guarantee that all previous produced material and lab conditions are ready to be used by the domain experts at the exam (Figure 6.10). For this purpose, a different person from the one who produced the material should assess it under the same conditions as domain experts will face in the exam, so that any inappropriate aspect can be easily identified and readjusted.

Nevertheless, if this is not followed, then the person who produced the material will find fewer inconsistencies, since he is too focused on the subject.

The process starts by defining and establishing the environment whereupon the exam will be conducted. If an in-house exam takes place, then it should be as natural as possible wherewith domain experts are used to in their workplace, in order to avoid any kind of pressure. On the other hand, if domain experts will be spread over different places, then the necessary means to support them should be created.

Then starts material review process according to the environment already pre-established. The subject or subjects assigned to this task, initiate the process by performing the tutorial looking for inconsistencies and/or misunderstandings. Then, if any data collection tool has been defined, it is time to test it and put it into operation for the next stage. At this point, each scenario and its respective questionnaire are reviewed. These intermediate questionnaires (Table 6.5) in most cases may suffer few changes and yet will provide great insights of DSL usability. The final step concerns reviewing the updates made to the final questionnaire (Table 6.4), as well as to the interview if it has been previously defined.

In the iterative evaluation model we can take advantage from our previous experiences, in terms of the right environment to provide to the domain experts, the most appropriate data collection tools to use and type of questions to be made to its users.

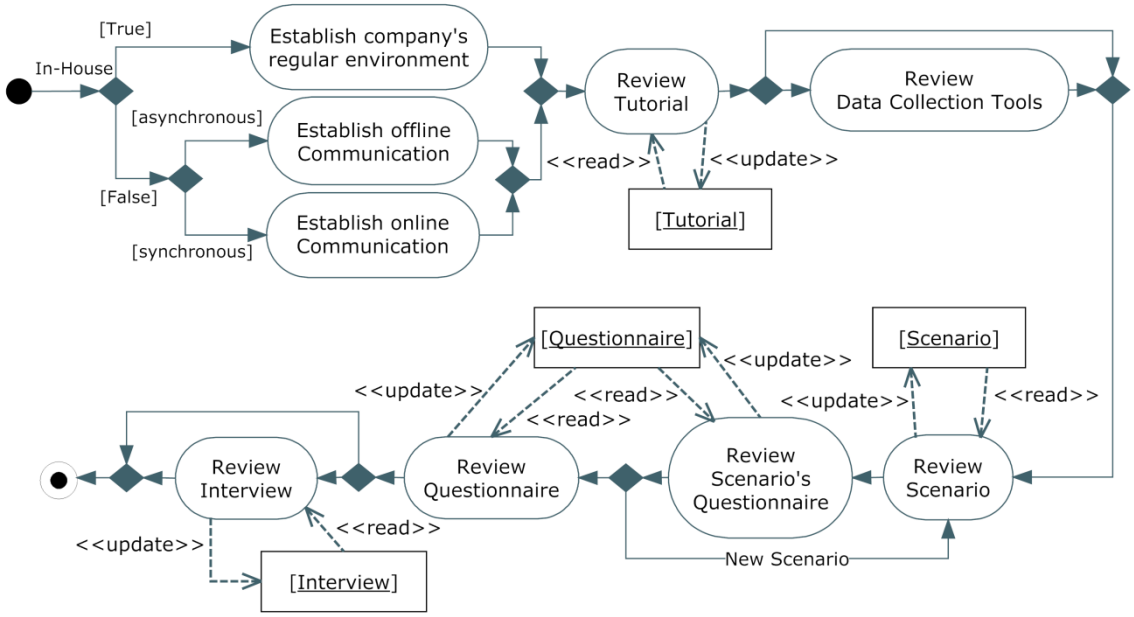

Figure 6.10 – Pilot Session

Training Session

The training session will help domain experts get used with the DSL, and prepare them to what they will do in the exam. Figure 6.11 expresses what we think ought to be done.

Training Session should start by informing each domain expert the uses to be made of the data that will be collected over the course of the experiment. In some cases it might be also useful to supply a consent form to be signed for all participants. Then, it should be established the environment experienced in *Pilot Session*. If necessary, explain how the data collection tools work, so that domain experts get used to and feel more comfortable during the exam. Finally, each participant performs the tutorial produced in *Task Preparation* and assessed in *Pilot Session*, so he gets aware of DSL terminology and its respective behavior.

The iterative evaluation model will not experience any transformation or gain compared to a single evaluation.

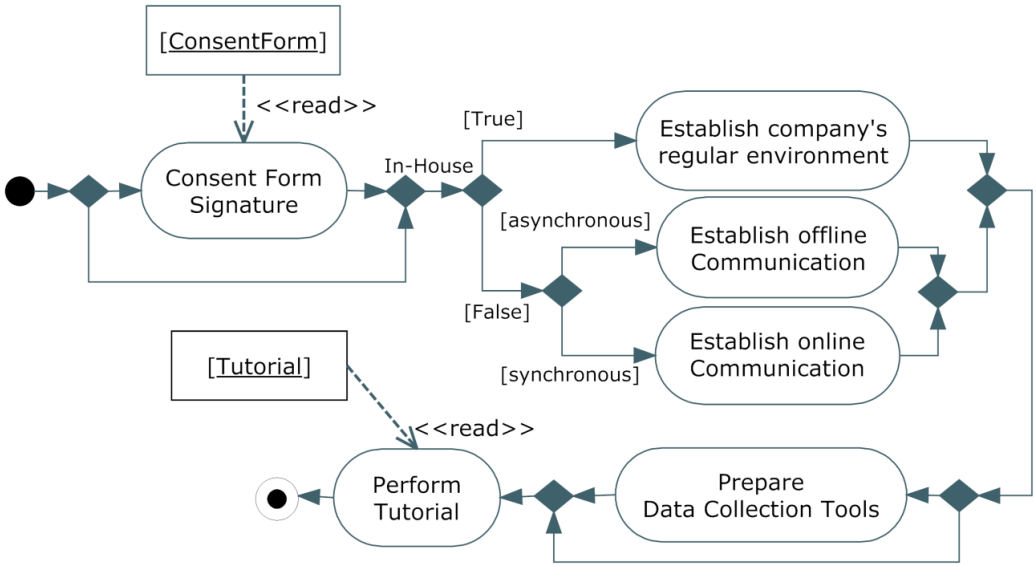

Figure 6.11 – Training Session

Exam

After all previous work to set up the experiment foundations, arises the moment to start the examination process (Figure 6.12). Subjects must be once again established with the optimal environment to perform the exam. We focus this aspect quite frequently since it is of fundamental importance to reduce domain experts' nervousness

and sensation of pressure, so that in the end we are not faced with unwanted biased results.

Any data collection tool previously defined should then be established to retrieve domain experts' actions. At this point, some assumptions must be very clear. One of them concerns the extent of help provided to domain experts. If a domain expert's group receive any help the other groups should receive the same treatment as well. Another assumption concerns the amount of interaction between the domain expert and the evaluator. As presented in *Scenarios* section, Observation and Think-aloud are two feasible approaches, but nevertheless can distract domain experts and bias the results if not done automatically (recording domain experts' session).

Only then, it is the right moment to initiate the exam. Here, for each scenario, domain experts should have a respective questionnaire to fill (scenario's questionnaire, Table 6.5). This will allow the evaluator to understand what went wrong with the attributes that have been evaluated in the underlying scenario. Then, the final questionnaire takes place to get domain experts final impressions of the DSL. In the end, if necessary and already defined in previous stages of the evaluation process, an interview should be performed.

In what respects the iterative evaluation model, this process remains the same between iterations as well as the amount of effort to conduct the exam.

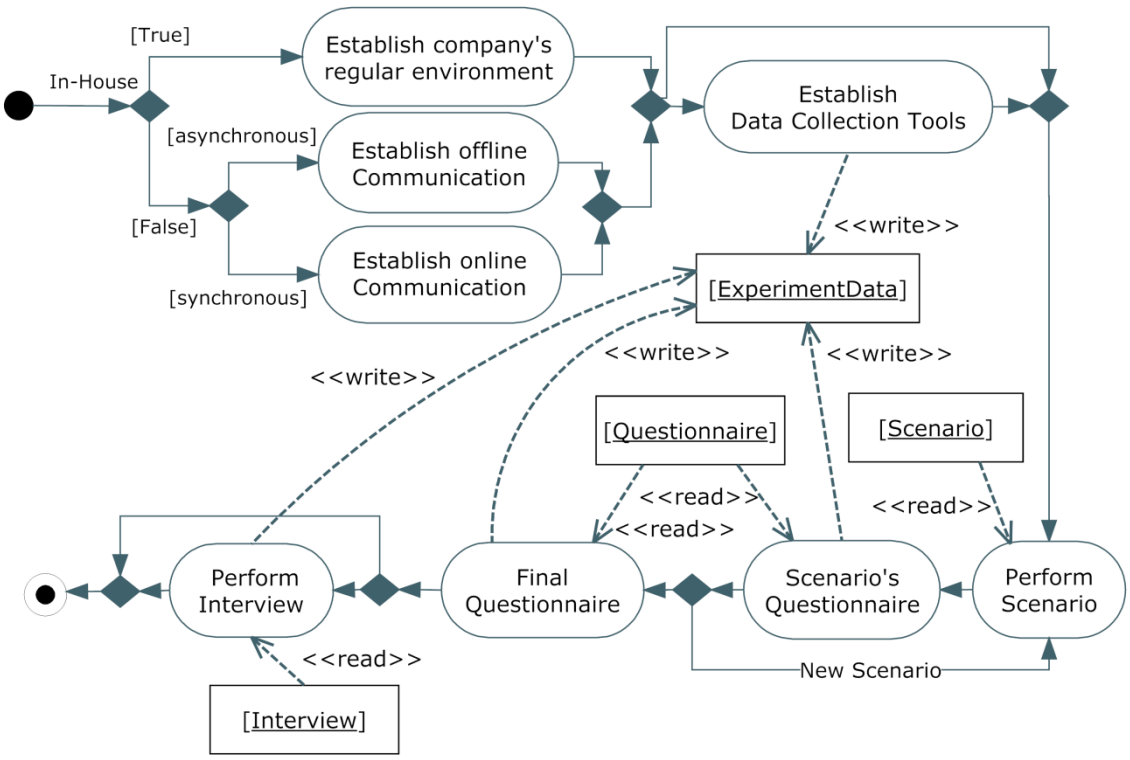

Figure 6.12 – Exam

Results Analysis

The final phase of our DSL evaluation process concerns the analysis of domain experts' results from the exam. The process is summarized in Figure 6.13. For each group of domain experts their results and impressions from the intermediate and final questionnaire should be stored in Appendix 1. Meanwhile, any threat found during the experiment should be meticulously analyzed in order to understand their impact on results. Only after, should start interpretation of results. During this process we identified two major objectives. Compare Results between Groups is the first one. Here domain experts' results from each group for a particular scenario should be compared with the other groups. This allows the evaluator to understand if a specific issue is restricted to a single group or occurs in the others. If a widespread problem takes place then it is compulsory to be changed. The second objective concerns the comparison of the results with previous versions. As previously mentioned in *Competitive Analysis* section, this helps better understand domain experts' results significance.

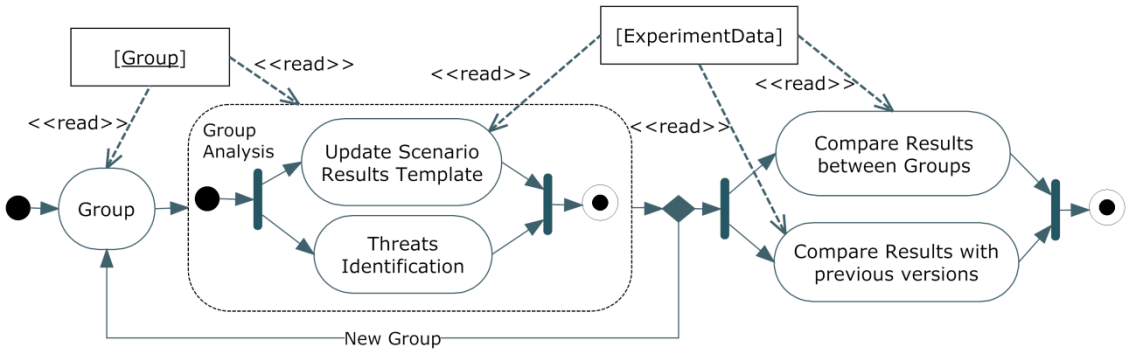

Figure 6.13 – Results Analysis

6.5. Summary

In this chapter we provided a Systematic Evaluation Methodology to assess Domain-Specific Languages usability. For this purpose we began by defining the actors involved in language's assessment, as well as their tasks.

Our next step focused on the establishment of the evaluation measures to be taken in each phase of DSL's development process. In this context, at the Domain Analysis phase we sought to define the measures to set up a baseline for comparison with the results obtained from language's assessment. For that purpose we specified the

languages to be considered in comparison, as well as, the elements of comparison, for instance, number of errors, user satisfaction, etc.

In the Design phase we produced an iterative evaluation model, where we outlined the measures for both Abstract and Concrete Syntaxes. In this process Software Language Engineer(s) and/or Evaluator(s) start by modelling a small subset of the language and then test it. For this purpose we considered prototyping as first evaluation method, since it is a preliminary version of the intended language, and unveils certain details that would be otherwise revealed in later stages. Then DSL's Visual Elements inspection follows it. For this purpose we provided a set of six heuristics to become this process more easily followed by Evaluators. We also specified which common shortcomings should be avoided when establishing the symbols for DSL's feature attributes. The next step concerns collecting domain experts' impressions about DSL's new feature correctness through Scenarios. Here we specified which measures should be taken into account when developing a scenario, which data collection methods can be used, and provide a questionnaire in order to assess domain experts' feelings.

We consider both conducted tests and amendments performed to the language should be stored in order to be used in future for comparison. For this purpose we provided a template to store the results of each group of domain experts.

Although our methodology has been especially designed for an iterative development, it is also applicable only in the final stage of the development process.

At the Implementation phase we did not assign any evaluation mechanism in the sense that it was outside the scope of our work, although, we are aware of its importance to domain experts overall satisfaction of the DSL.

In the last phase, the Evaluation, we established the procedures to prepare, conduct and analyze the experiment. Here we defined: how to group the domain experts assigned to the experiment, as well as their number; how to prepare the material for domain experts' exam, and how to evaluate such produced material and by whom; how to conduct domain experts' training session, so that they get used with the DSL; how to conduct domain experts' exam and what should be done in order to avoid biasing the results; and how to analyze the experiment results.

Our methodology has been built on the notion of combining several usability methods and data collection methods from Usability Engineering and Experimental Software Engineering, in order to help the evaluator assess the language. The choice of the evaluation methods, as well as the evaluation process has in consideration organization's budget constraints, and domain experts' satisfaction.

When producing a DSL every stakeholder should remain in his mind that despite all efforts to conduct the most thorough experiment there is always something that escapes and the workbench tool has its limits, where in extreme cases may influence negatively our pretensions, by dictating language development procedures [19].

7. Systematic Evaluation Methodology Validation

7.1. Introduction

In the previous chapter we provided a methodology to help DSL producers to overcome the documented low evaluation, by providing an evaluation roadmap to follow. In this chapter we try to characterize our Systematic Evaluation Methodology in terms of easiness of use and capability to improve DSLs' usability standards compared to previous state of practice.

To measure and understand the full extent of the benefits of our approach, we conducted a software engineering case study in which DSL producers that assessed their languages according to our methodology has been compared with those who have used an ad-hoc evaluation approach. With the results of this study, we seek to answer the following research question:

- Does a Systematic Evaluation Methodology brings effective advantages to your DSL?

The remainder of this chapter is structured as follows. Section 7.2 presents the questionnaire and DSL producers involved in the experiment. In section 7.3 we discuss the results of the experiment. Section 7.4 discusses the feasible threats to experiment validity and how they were mitigated. Section 7.5 summarizes this chapter.

7.2. Experiment Design

In order to answer the previous research question, we need to measure the success of our Systematic Evaluation Methodology criteria. For this purpose, we began by establishing two groups of DSL producers, one for those who evaluated their DSLs based on our methodology and another for those who used an ad-hoc evaluation approach. In both cases, DSL producers only assessed their languages after completing

the development process. In the end of their language evaluation we asked both groups to answer a questionnaire over the internet.

During their evaluation process, we never gave any kind of advice, but we did clarify any doubts about our methodology. We think that this process helped us avoiding the contamination of the results according to our desires or expectations.

7.2.1. Subjects

Eight subjects were engaged to our experiment. Four of them used our methodology. They developed: a DSL for ubiquitous devices, a DSL for languages composition, between "I*" and "KAOS", a DSL tool for transformations, and the last subject did not actually develop a DSL, but used two DSLs to perform transformation rules between "I*" and "KAOS". The other four subjects based on an ad-hoc evaluation approach developed: a DSL to specify "I*" language rules, a DSL for queries optimization, a DSL to specify "KAOS" language rules, and a DSL to specify applications of augmented reality. In both cases subjects were MSc Students from DI/FCT.

The four subjects, who used our methodology, have followed a single iteration of our iterative evaluation model presented in Figure 6.3 of section 6.4.2. This has happened since they could only assess their languages after the development process has been fully completed.

In order to explain our evaluation methodology, presented in section 6.4, we carried out a debriefing session. Here, we gave the details about what should be developed and established in each stage of the evaluation process (Figure 6.6), the actors that should be involved and their tasks (section 6.3), and the number of domain experts that we recommend for each group. The material of our methodology, scenario's questionnaire (Table 6.5), final questionnaire (Table 6.4) and template to store domain experts' results (Appendix 1) was also made available to them.

7.2.2. Questionnaire

A questionnaire has been answered by both groups (Table 7.1). Every question relates to one of the two main areas. The first one, questions Q1 to Q12, concerns the procedures that DSL producers have made to set up the experiment. Here, we tried to

identify their experimental foundations and advantages from using them. For those who have followed our methodology we tried to understand the extent to which they used of our evaluation methodology. The second area of interest, questions Q13 to Q17, concerns DSL producers' general impressions. This time, for each group of DSL producers we sought to understand their satisfaction level with the evaluation process followed.

Subjects' responses have been collected in two ways: through a Likert scale and open answers. We used both, in order to get more knowledge about certain options they have made during language's assessment.

ID	Question
Experiment	
Q1	Have you used the checklist based validation? Yes/No If you answered No, explain why.
Q2	How many changes have you done to the DSLs Concrete Syntax based on the checklist based validation?
Q3	Have you followed the entire Evaluation Process? Yes/No If you answered No, which of the following steps did you not use? Subject Recruitment Task Preparation Pilot Session Training Session Exam Scenario's Questionnaire Final Questionnaire Result Analysis Why did you not use them? (Provide a reason for each unused step)
Q4	How many groups of domain experts did you create? (E.g. Experienced Group, Student Group, etc.)
Q5	How many domain experts did you assign to each group?
Q6	How many scenarios have you defined?
Q7	Did you add new questions or update the existing ones from the provided questionnaire? Yes/No Identify the new questions and the updated ones, and give a reason *Why* you needed it.
Q8	Did you remove any question? Yes/No Identify the ID of the removed questions, and give a reason *Why* you removed it.
Q9	How did you set up experiment's environment? In-house Environment Asynchronous Communication (offline communication, e.g. email) Which one?

	Synchronous Communication (online communication, e.g. VOIP) Which one?
Q10	Did you establish any Data Collection Tools (e.g. Recording, Log Analysis)? Yes/No Which tools? (If you answered Yes)
Q11	How much effort (Man-Hours) did it take to set up the experiment until domain experts' Exam?
Q12	How many changes have you performed to the DSL after the experiment?
General Impressions	
Q13	How useful did you find the checklist based validation? (Very Good, Good, Satisfactory, Bad, Very Bad)
Q14	How demanding did you find establishing the experiment? (Undemanding, Simple, Regular, Tough, Severe)
Q15	Which step(s) of the Evaluation Process did you find more challenging to follow? Subject Recruitment Task Preparation Pilot Session Training Session Exam Scenario's Questionnaire Final Questionnaire Result Analysis Why? (Please express your feelings for each selected one)
Q16	Did you feel lost in *What* and *How* to do, to establish the experiment? (Very often, Often, Sometimes, Seldom, Never)

Table 7.1 – Questionnaire

7.3. Results of the Case Study

In this section we present the results of our software engineering case study grouped by three main areas: DSL Producers' Decisions (section 7.3.1), where we present DSL producers' decisions to establish the experiment and changes they found necessary to perform to their languages in the end of the evaluation process; Checklist Based Validation (section 7.3.2), here we focus our attention on the pre-validation of languages' Concrete Syntax carried out by DSL producers that used our evaluation methodology; and Evaluation Process (section 7.3.3), where for both groups of DSL producers we explore their impressions about the evaluation process they have followed.

7.3.1. DSL Producers' Decisions

To understand the similarities and differences between both groups of DSL producers, in terms of their decisions to establish the experiment and changes performed in the end of the evaluation process was our first goal. For this purpose, for each row of Table 7.2 we present the number of domain experts within each group created by the DSL producer, the number of scenarios developed, their effort to set up the experiment, and the number of changes made to the language. The effort to set up the experiment has been measured in Man-Hour, since this unit accounts the effective work performed by all people involved in the process [108].

Evaluation Type	N Domain Experts per Group			N Scenarios	Effort (Man-Hour) to set up the experiment	Changes
	Group 1	Group 2	Group 3			
Systematic Evaluation Methodology	7	–	–	2	14	2
	5	5	–	2	8	1
	6	3	–	1	12	4
	5	5	–	1	7	22
Ad-hoc Evaluation	10	–	–	3	6	1
	2	2	2	3	6	0
	5	–	–	8	15	10
	5	–	–	2	2	1

Table 7.2 – Questions Q4, Q5, Q6, Q11, Q12

Regarding the number of domain experts identified to involve in DSLs' evaluation, presented in *Subject Recruitment's* section, three DSL producers based on our methodology were capable to fulfil our recommendations. The remaining DSL producer was unable to reach our recommendations due to insufficiency of available personnel to assess his language. On the other hand, none of the DSL producers who followed an ad-hoc evaluation has assigned enough domain experts to meet the requirements of our methodology.

Only based on the findings about the number of domain experts to involve in DSLs' evaluation and their capacity to find inconsistencies in a language, we can assert that DSL producers, who used our methodology, find more inconsistencies in their language than those who have followed the ad-hoc evaluation. Based on the results of our DSL

producers, we verify that our expectations come true, since, in general, more changes have been performed by those who have followed our methodology.

Scenarios created by our DSL producers were another of our concerns. This is due to scenarios being directly related to the extent of language evaluated. As described in *Scenarios*' section, more scenarios do not necessarily mean more language evaluated or even more errors found, if enough efforts are not made to guarantee that all attributes are effectively used by their domain experts during the evaluation session. In our case study we can notice that our statement confirms, since a larger number of scenarios, developed by DSL producers based on an ad-hoc evaluation, did not contribute with more inconsistencies found and changes performed.

DSL producers using our methodology spent slightly more effort to set up the experiment than those who followed an ad-hoc evaluation. This effort made by both groups is directly related with the scenarios produced, and their measures to conduct and analyze the experiment. We can notice that the superior effort spent by those who followed our methodology, combined with our recommendations about the number of domain experts to involve in language's evaluation, and clarifications about how to produce Scenarios, made possible for DSL producers to find more inconsistencies in their languages. In this sense, these results lead us to believe that our guidelines helped the DSL producers in producing and establishing the evaluation material.

Another point of interest was the kind of environment established by DSL producers and the mechanisms they have used to communicate with his domain experts (Figure 7.1). Regarding this subject we notice great similarities between both groups of DSL producers, which strengthen our previous considerations.

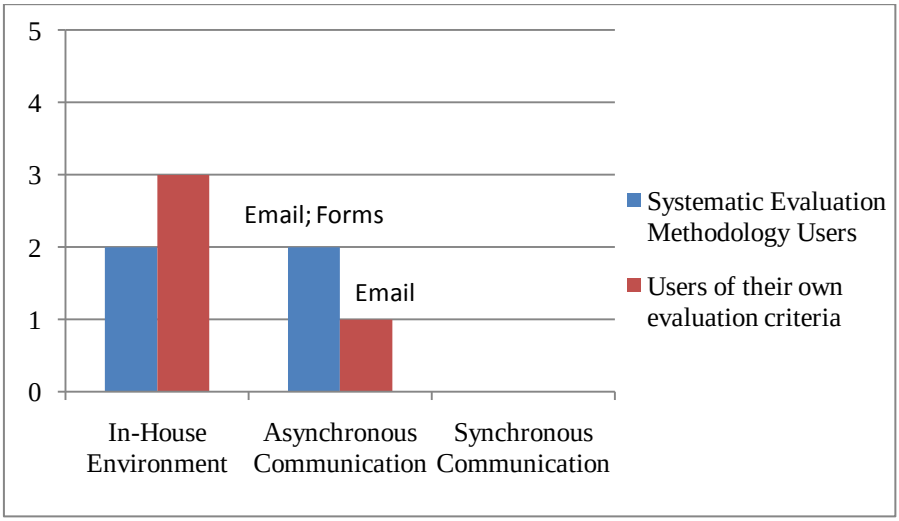

Figure 7.1 – Question 8. How did you set up experiment's environment?

When asked if they have used and/or established any Data Collection Tool, all of them promptly answered to have not used it.

7.3.2. Checklist Based Validation

Regarding this element of validation to assess languages' Concrete Syntax, we can conclude that it brought considerable advantages to those who have used it. From the four DSL producers involved, three of them guarantee to have followed the checklist based validation and said to have been a good or very good option concerning its usefulness (Table 7.2).

In the case of two DSL producers it led to carry out four changes for one of them and two changes for the other. One did not find any inconsistency, and the other DSL producer have not followed it since he has not developed a DSL, but used two DSLs to perform his work.

Despite we are aware that results does not present statistical significance, they are encouraging with respect to these evaluation criteria being truly valuable to find inaccuracies even before involving the domain experts in the case study, leading us to consider that with a larger sample similar results will be found.

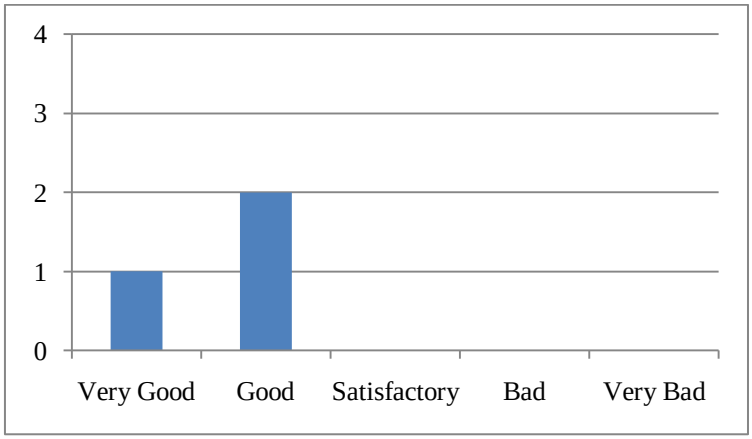

Figure 7.2 – Question Q13. How useful did you find the checklist based validation?

7.3.3. Evaluation Process

Here, for both groups of DSL producers, we present their impressions about the evaluation process they have followed. For those DSL producers based on our

evaluation methodology we also have focused on their choices and what they have used from our methodology to evaluate their languages.

From the four DSL producers who have followed our methodology, none have used all the resources that we made available, and some steps were considered more challenging than others (Figure 7.3). Nevertheless, all of them reveal a very good impression about the evaluation process that we present.

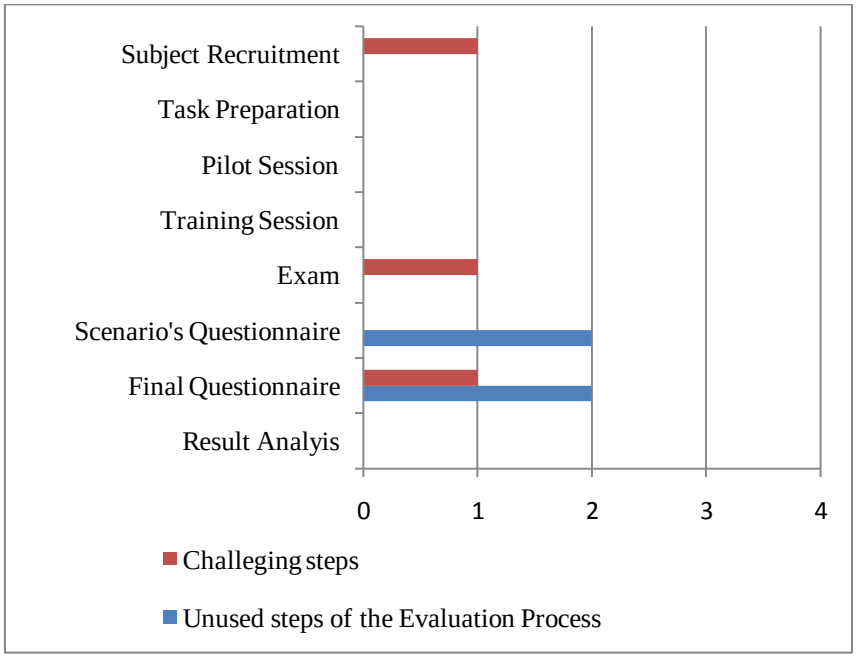

Figure 7.3 – Questions Q3 and Q15

The scenario's questionnaire and final questionnaire, that we made available, have been used by two DSL producers. The other two DSL producers have not used them since another questionnaire had already been approved by their master thesis supervisors, in order they could compare the results among them.

Both DSL producers who used our questionnaires decided to merge them. The reason that one of them gave for doing this was: "*Due to low availability of users, we chose to use only one final questionnaire*". When asked about the questions they have updated or removed, one answered to have removed questions L4 and EX1 from Table 6.4 and Table 6.5, and the other made small adjustments to suit the questionnaire to his DSL needs, but did not specify those changes in detail.

According to both DSL producers' answers we notice that some decisions are dependent on the objectives, and capacity to find enough domain experts to evaluate the language.

When all four DSL producers were asked about the most challenging steps of the evaluation process, one reported to be the Subject Recruitment/Exam, where he made the following statement: "*They are both related, and the problem is always the user availability. In subject recruitment it is not always easy to find the candidates to perform a good evaluation*", and other said to be the Final Questionnaire, since he had to give domain experts a prior training so they could be able to answer the questionnaire.

As result of their answers we confirm that none of them has felt truly difficulties to set up our recommendations, but with custom adjustments in order to fulfil their own objectives. This statement can be consolidated by Figure 7.4. This time we asked them about how demanding they found establishing the experiment, where most of them found it simple. On the other hand, DSL producers based on an ad-hoc evaluation found it relatively more demanding.

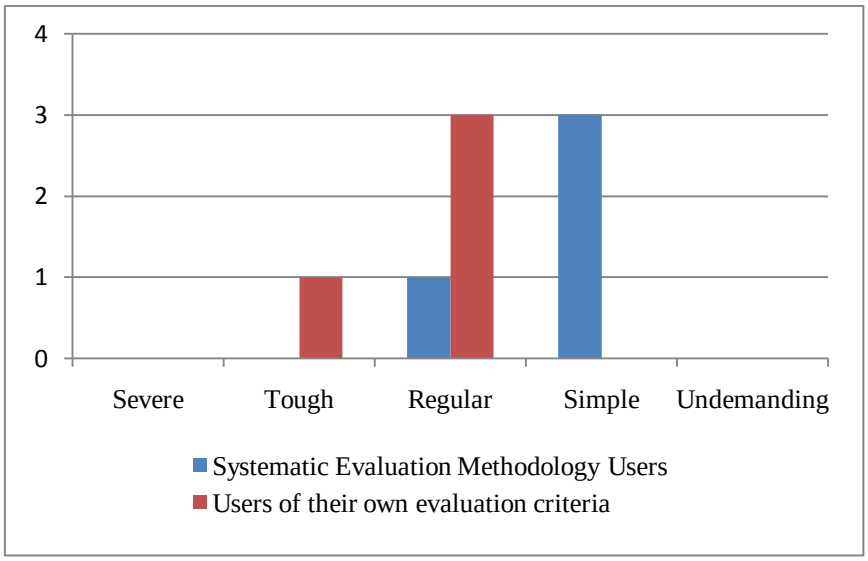

Figure 7.4 – Question Q14. How demanding did you find establishing the experiment?

A final question has been made to both groups of DSL producers in order to understand if, in any time, they felt lost in *What* and *How* to do to establish the experiment. Based on their answers, depicted Figure 7.5, we observe that most DSL producers who followed our methodology said they never felt lost, whereas the other group is divided between seldom, sometimes and often. These results reinforce our confidence that our efforts to establish a roadmap that could be easily understood

were successfully achieved. In this sense, we believe to have provided a helpful mechanism to DSL producers evaluate their languages.

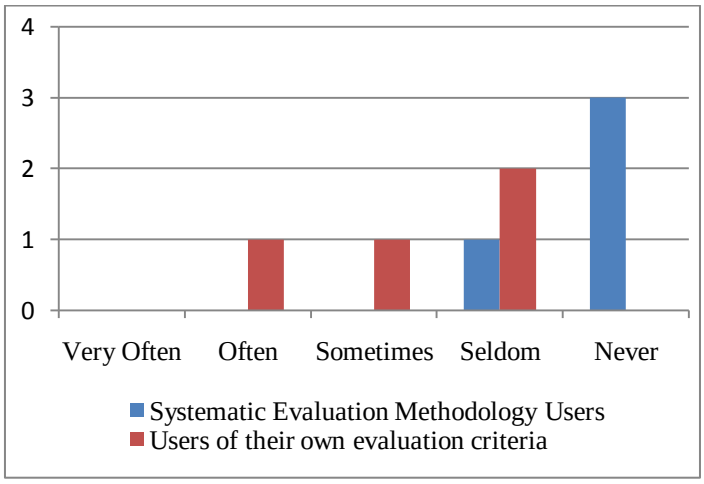

Figure 7.5 – Question Q16. Did you feel lost in *WHAT* and *HOW* to do, to establish the experiment?

7.4. Threats to Validity

When conducting a survey, any sharing of information between subjects should be avoided, otherwise answers of a respondent may be influenced by other replies [109]. In order to mitigate this issue we made sure that no respondent had access to the responses of the others. Moreover, some subjects did not know the other elements involved in the experiment, which made less likely that they could share opinions about the survey.

Regarding the questionnaire presented to our DSL producers, it was first built and then reviewed by my supervisors, and only after that, subjects had the opportunity to answer it. This helped to assure a further content validity. Every time we asked our DSL producers to estimate the amount of time spent in their evaluation tasks we were aware that effort information tends to be imprecise. However, we believe the results are sufficiently reliable, based on our knowledge on their experiments. Furthermore, we have no reason to think imprecisions would favour one of the alternatives over the other.

Regarding the methods to manipulate the data, they were very straightforward, and did not introduce any kind of menace to survey validity.

7.5. Summary

The goal of this case study was to understand the applicability and advantages of our Systematic Evaluation Methodology against previous state-of-affairs, to assess Domain-Specific Languages. For this purpose, we defined two groups of DSL producers. In both cases subjects were MSc Students from DI/FCT. One group has evaluated their languages based on our methodology and the other group followed an ad-hoc evaluation approach.

In order to assess DSL producers' impressions and understand the extent of use they have made from our methodology we asked them to answer a questionnaire over the internet. Here we focused our attention in three main areas: DSL producers' decisions, checklist based validation, and general impressions about the evaluation process they have followed.

Our software engineering case study has produced promising results showing that our evaluation process was easily followed and DSL producers found themselves much more confortable in *What* and *How* to do at each moment of the evaluation of their languages, comparatively to those who have used an ad-hoc evaluation. The checklist based validation presented in our methodology has shown good results as well. Here, the group of DSL producers using our methodology was capable to find significant inconsistencies. DSL producers based on our methodology found more inconsistencies than those based on an-hoc evaluation.

We also noted how that some evaluation approaches are truly dependent on the extent that each one gives to the experiment. An example of it was scenario's questionnaire suppression by two DSL producers.

Some evaluation criteria presented in our Systematic Evaluation Methodology have not been able to be experienced in this case study. An instance of that concerns the iterative evaluation that we presented in section 6.4.2. To have followed it, it would have been necessary more time and resources (e.g. DSL producers), which in this dissertation we did not have.

Based on our study results we believe that we are capable to answer our research question: "Does a Systematic Evaluation Methodology brings effective advantages to your DSL?", by saying that these early tests on our methodology point to tangible benefits to those who use it to attain a language with higher usability standards, and so increase DSL proclaimed productivity levels [21].

8. Conclusions

8.1. Summary

In this dissertation we have shown that the DSL community does not systematically report on the realization of any sort of experimental validation of the languages it builds, as discussed in chapter 5. In practice, it is as if decision makers prefer to risk using or selling inadequate DSLs to their end-users (the Domain Experts), rather than spending resources evaluating them.

Regarding these facts, in chapter 6, we propose a Systematic Evaluation Methodology to mitigate this problem that can be easily followed by Software Language Engineers and/or Evaluators, and thus improve current DSL development practices from craftsmanship to an Engineering activity. For this purpose, we began by doing a review of existing techniques from Experimental Software Engineering, discussed in chapter 3, and Usability Engineering, discussed in chapter 4. From them we borrowed data collection methods and usability methods, respectively.

Our next step was to develop the methodology itself. In this sense, for each phase of DSL's development process we specified the evaluation procedures to be taken. In the Domain Analysis phase we sought to define the measures to set up a baseline for comparison with the results obtained from language assessment. For this purpose we specified the languages (GPLs and/or DSLs) to be considered in comparison, as well as, the elements of comparison, for example, number of errors, user satisfaction, etc.

In the Design phase we produced an iterative evaluation model, where we outlined the measures for both Abstract and Concrete Syntaxes. In this process Software Language Engineer(s) and/or Evaluator(s) start by modelling a small subset of the language and then test it. For this purpose we considered prototyping as first evaluation method, since it is a preliminary version of the intended language, and unveils certain details that would be otherwise revealed in later stages. Then DSL's Visual Elements inspection follows it. For this purpose we provided a set of six heuristics to become this process more easily followed by Evaluators. We also specified which common shortcomings should be avoided when establishing the symbols for DSL's feature

attributes. The next step concerns collecting domain experts' impressions about DSL's new feature correctness through Scenarios. Here we specified which measures should be taken into account when developing a scenario, which data collection methods can be used, and provide a questionnaire in order to assess domain experts' feelings.

We consider both conducted tests and amendments performed to the language should be stored in order to be used in future for comparison. For this purpose we provided a template to store the results of each group of domain experts.

Although our methodology has been especially designed for an iterative development, it is also applicable only in the final stage of the development process.

At the Implementation phase we did not assign any evaluation mechanism in the sense that it was outside the scope of our work, although, we are aware of its importance to domain experts overall satisfaction of the DSL.

In the last phase, the Evaluation, we established the procedures to prepare, conduct and analyze the experiment. Here we defined: how to group the domain experts assigned to the experiment, as well as their number; how to prepare the material for domain experts' exam, and how to evaluate such produced material and by whom; how to conduct domain experts' training session, so that they get used with the DSL; how to conduct domain experts' exam and what should be done in order to avoid biasing the results; and how to analyze the experiment results.

Then, in order to validate our Systematic Evaluation Methodology we carried out a Software Engineering case study, discussed in chapter 7. For this purpose, we defined two groups of DSL producers. In both cases subjects were MSc Students from DI/FCT. One group has evaluated their languages based on our methodology and the other group followed an ad-hoc evaluation approach. The Application Domain of the studied DSLs are based on "KAOS" and "I*" transformations, "I*" language rules, ubiquitous devices, augmented reality and queries optimization.

In order to assess DSL producers' impressions and understand the extent of use they have made from our methodology we asked them to answer a questionnaire over the internet. Here we focused our attention in three main areas: DSL producers' decisions, checklist based validation and general impressions about the evaluation process they have followed.

Our questionnaire produced some interesting results. One of these results concerns to the number of inconsistencies that both groups of DSL producers were able to find in their own languages. The DSL producers who followed our methodology found more inconsistencies than those following an ad-hoc evaluation.

The checklist based validation presented in our methodology has shown good results as well. Here, the group of DSL producers using our methodology was capable to find significant inconsistencies. When we asked both groups if, in any time, they felt lost in *What* and *How* to do to establish the experiment, almost all of those who followed our methodology, except one, answered to never felt lost. In contrast, the answers from participants of the ad-hoc evaluation group included instances of "seldom", "sometimes", and "often".

From this software engineering case study we were also able to confirm that some evaluation approaches are truly dependent on the extent that each one gives to the experiment.

In summary, despite we have experienced our Systematic Evaluation Methodology with a relatively small sample of academia DSL producers', a common practice and already noticed as an interesting alternative to carry out pilot studies in related areas [36, 99, 110], it shows promising results.

8.2. Future Work

We identify several areas for future work. One direction is to conduct a similar survey as presented in chapter 7, but this time at an industry level. A second direction is to experiment the iterative evaluation process during the DSL development process in both academia and industrial level. The challenge here will be to find academia subjects with enough experience and skills in developing DSLs and an industrial partner that is willing to collaborate in such surveys.

It is our belief that these opportunities will be useful, not only to corroborate the results obtained from our previous software engineering case study to validate our methodology, but also to the DSL community, since it might highlight with more precision the impact that our evaluation procedures have on: DSL development costs, domain experts' productivity and satisfaction, and amount of time and evaluators to establish the experiment.

Academia and industrial level experiments may also have an important role in order to find future increments to our methodology. The collectable attributes from previous versions of the language and/or competitive languages is one case. Here new base values for comparison with the new DSL, the Goal Settings, may be found useful. Both scenario's questionnaire and final questionnaire may also benefit from it, since new

relevant questions may be found to retrieve domain experts' thoughts. The list of common shortcomings to avoid when establishing the symbols for DSL's feature attributes may grow up as new common mistakes are being noticed. In the end, the template to store the results of each group of domain experts may be updated in order to satisfy in a better way other software language engineers or evaluators desires.

A third direction is to assess the applicability of our Systematic Evaluation Methodology, with the necessary adjustments, outside DSL scope, in particular in GPLs, in order to understand if they would benefit from our work regarding the step-by-step evaluation process presented in section 6.4.4, and so increase their usability standards.

Appendix 1 – Group Results for each Scenario and their General Impressions

Domain Experts Classification		Tool
ID		
Background		
Type		

Scenarios' Results

Scenario	Domain Expert	Success	Completion (%)	**N** Errors	Help		Duration (HH/MM)	Ease of Use			Expressiveness	Effectiveness
					Doc	Sup		Mental Effort	Confused	Confident	Compact	Correctness
1												
2												

	Related Questions
N Errors	F3 and F4 gives Feedback
Help(Documentation, Supervisor)	L3, L4
Mental Effort	U6
Trapped/Confused	U5
Tool	U1. U3 physical effort
Confident	U4
Change	U2
Symbols	F1
Textual	F2
Overall	G1
Compact	EX3
Total/Partial	EX1
UEI(unable to express intension)	EX2
Correctness	EF2
Expectation	EF1

General Impressions								
Satisfaction						Expressiveness		Effectiveness
Domain Expert	Tool	Change	Symbols	Textual	Overall	Total/Partial	UEI	Expectation

Scenario Description	
Scenario 1	
Scenario 2	
Notes	

Bibliography

[1] S. Thibault, “Domain-Specific Languages: Conception, Implementation and Application,” University of Rennes, France, 1998.

[2] B. W. Bohem, “A spiral model for software development and enhancement,” *Computer*, vol. 21, no. 5, pp. 61-72, May, 1988.

[3] W. W. Royce, “Managing the Development of Large Software Systems,” *Technical Papers of Western Electronic Show and Convention*, August, 1970.

[4] A. van Deursen, P. Klint, and J. Visser, “Domain-Specific Languages: An Annotated Bibliography,” *SIGPLAN Notices*, vol. 35, no. 6, pp. 26-36, 2000.

[5] J. Hammond. "Boosting productivity with domain-specific modelling," 2008; http://www.metacase.com/papers/mte-boosting.pdf.

[6] M. Mernik, J. Heering, and A. M. Sloane, “When and How to Develop Domain-Specific Languages,” *ACM Computing Surveys*, vol. 37, no. 4, pp. 316-344, December, 2005.

[7] J. L. Hammond. "Improving productivity and quality with domain-specific modeling," 2008; http://www.metacase.com/papers/ESDE-apr08.pdf.

[8] L. Cao, B. Ramesh, and M. Rossi, “Are Domain-Specific Models Easier to Maintain Than UML Models?,” *IEEE Software*, vol. 26, pp. 19-21, 2009.

[9] J. Nielsen, *Usability Engineering*: AP PROFESSIONAL, 1993.

[10] A. Abran, and J. W. Moore, *Guide to the Software Engineering Body of Knowledge*, 2004.

[11] D. A. Sadielek, and S. Weißleder, “Towards Automated Testing of Abstract Syntax Specifications of Domain-Specific Modeling Languages,” in Workshop on Domain-specific Modeling Languages, Humboldt-Universität zu Berlin, Germany, 2008.

[12] G. Guizzardi, L. F. Pires, and M. v. Sinderen, “Ontology-based evaluation and design of domain-specific visual modeling languages,” in International Conference on Information Systems Development (ISD), Karlstad, Sweden, 2005.

[13] R. Tairas, M. Mernik, and J. Gray, “Using Ontologies in the Domain Analysis of Domain-Specific Languages,” in Workshop on Transformation and Weaving Ontologies in Model-Driven Engineering (TWOMDE), Toulouse, France, 2008, pp. 332-342.

[14] P. Gabriel, M. Goulão, and V. Amaral, “Do Software Languages Engineers Evaluate their Languages?,” in XII Congreso Iberoamericano en "Software Engineering", Cuenca, Ecuador, 2010.

[15] J. Gray, M. Rossi, and J.-P. Tolvanen, “Preface,” *Journal of Visual Languages and Computing*, vol. 15, pp. 207-209, June-August, 2004.

[16] A. J. Sánchez-Ruíz, M. Saeki, B. Langlois *et al.*, “Domain-Specific Software Development Terminology: Do We All Speak the Same Language?,” in OOPSLA Workshop on Domain-Specific Modeling, Jacksonville, Florida, USA, 2006.

[17] A. V. Deursen, and P. Klint, “Little Languages: Little Maintenance?,” *Journal of Software Maintenance: Research and Practice*, vol. 10, no. 2, pp. 75-92, March-April, 1998.

[18] Ø. Haugen, and P. Mohagheghi, “A Multi-dimensional Framework for Characterizing Domain Specific Languages,” in OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada, 2007.

[19] S. Kelly, and R. Pohjonen, “Worst Practices for Domain-Specific Modeling,” *IEEE Software*, vol. 26, pp. 22-29, July/August, 2009.

[20] N. A. Allen, C. A. Shaffer, and L. T. Watson, “Building modeling tools that support verification, validation, and testing for the domain expert,” in WSC '05: Proceedings of the 37th conference on Winter simulation, Orlando, Florida, 2005, pp. 419-426.

[21] S. Kelly, and J.-P. Tolvanen, *Domain-specific modeling: Enabling Full Code Generation*: John Wiley & Sons, 2008.

[22] MetaCase. "Nokia Case Study," 2007; http://www.metacase.com/papers/MetaEdit_in_Nokia.pdf.

[23] MetaCase. "EADS Case Study," 2007; http://www.metacase.com/papers/MetaEdit_in_EADS.pdf.

[24] J. Sprinkle, M. Mernik, J.-P. Tolvanen *et al.*, “What Kinds of Nails Need a Domain-Specific Hammer?,” *IEEE Software*, vol. 26, pp. 15-18, 2009.

[25] J. Heering, and M. Mernik, “Domain-Specific Languages as Key Tools for ULSSIS Engineering,” in ULSSIS, Leipzig, Germany, 2008, pp. 1-2.

[26] T. Baar, "Correctly Defined Concrete Syntax for Visual Modeling Languages," Springer, 2006.

[27] K. Smolander, V. P. Tahvanainen, and P. Marttiin, “Metaedit - a exible graphical environment for methodology modelling,” in International Conference on Advanced Information Systems Engineering, CAISE'91, Trondheim, Norway, 1991.

[28] S. Kelly, K. Lyytinen, and M. Rossi, “Metaedit+ a fully configurable multi-user and multi-tool case and came environment.,” in 8th International Conference on Advanced Information Systems Engineering, CAiSE'96, Herak-lion, Crete, Greece, 1996, pp. 1-21.

[29] W. Moore, D. Dean, A. Gerber *et al.*, “Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework,” *IBM Redbooks*, 2004.

[30] Vanderbilt, "Gme: Generic modeling environment," 2007.

[31] S. Cook, G. Jones, S. Kent *et al.*, "Domain-Specific Development with Visual Studio DSL Tools," Addison-Wesley Professional, 2007.

[32] B. W. Boehm, *Software Engineering Economics*: Prentice-Hall, 1981.

[33] S. Easterbrook, J. Singer, M.-A. Storey *et al.*, *Selecting Empirical Methods for Software Engineering Research*: Springer, 2007.

[34] C. B. Seaman, “Qualitative Methods in Empirical Studies of Software Engineering,” *IEEE Transactions Software Engineering*, vol. 25, no. 4, pp. 557-572, July/August, 1999.

[35] M. V. Zelkowitz, and D. R. Wallace, “Experimental Models for Validating Technology,” *Computer*, vol. 31, no. 5, pp. 23-31, May, 1998.

[36] D. I. K. Sjoberg, J. E. Hannay, O. Hansen *et al.*, “A Survey of Controlled Experiments in Software Engineering,” *IEEE Transations Software Engineering*, vol. 31, no. 9, pp. 733-753, September, 2005.

[37] A. Kamandi, and J. Habibi, “Modeling Languages Study and Evaluation Techniques,” *Asia International Conference on Modelling & Simulation*, vol. 0, pp. 553-558, 2008.

[38] C. Wohlin, M. Host, P. Runeson *et al.*, *Experimentation in Software Engineering: An Introduction*, 2000.

[39] B. Kitchenham, *Guidelines for performing Systematic Literature Reviews in Software Engineering*, Keele University and University of Durham, UK, 2007.

[40] J. Nielsen, “Enhancing the Explanatory Power of Usability Heuristics,” in Conference on Human Factors in Computing Systems, Boston, Massachusetts, United States, 1994, pp. 152-158.

[41] "I. O. for Standardization. *ISO Norm 9241-11*," 2007; http://www.iso.org.

[42] S. Kelly, and J.-P. Tolvanen, “Visual domain-specific modelling: benefits and experiences of using metacase tools,” in International Workshop on Model Engineering, ECOOP'2000, 2000.

[43] J. W. Carlson, “A Visual Language for Data Mapping,” in OOPSLA Workshop on Domain-Specific Visual Languages, Tampa-Bay, Florida, USA, 2001.

[44] C. Schmidt, P. Pfahler, U. Kastens *et al.*, “SIMtelligence Designer/J: A Visual Language to Specify SIM Toolkit Applications,” in OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA, 2002.

[45] E. S. Grant, J. Whittle, and R. Chennamaneni, “Checking Program Synthesizer Input/Output,” in OOPSLA Workshop on Domain-Specific Modeling, Anaheim, California, USA, 2003.

[46] J. Evermann, and Y. Wand, “Toward Formalizing Domain Modeling Semantics in Language Syntax,” *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 31, no. 1, pp. 21-37, January, 2005.

[47] B. Trask, and A. Roman, “Using Domain Specific Modeling in Developing Software Defined Radio Components and Applications,” in ECOOP Workshop on Domain-Specific Program Development (DSPD), Nantes, France, 2006.

[48] V. Svansson, and R. E. Lopez-Herrejon, “A Web Specific Language for Content Management Systems,” in OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada, 2007.

[49] H. Prähofer, D. Hurnaus, C. Wirth *et al.*, “The Domain-Specific Language Monaco and its Visual Interactive Programming Environment,” *IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 104-110, September, 2007.

[50] D. S. Kolovos, R. F. Paige, L. M. Rose *et al.*, “Implementing the Interactive Television Applications Case Study using Epsilon,” in Model-Driven Development Tool Implementers Forum, Zurich, Switzerland, 2007.

[51] D. A. Sadielek, “Prototyping Domain-Specific Languages for Wireless Sensor Networks,” in Workshop on Software Language Engineering, Nashville, Tennessee, USA, 2007.

[52] N. Bencomo, P. Grace, C. Flores *et al.*, "Genie: Supporting the Model Driven Developmnt of Reflective, Component-based Adaptative Systems," in ICSE, Leipzig, Germany, 2008, pp. 811-814.

[53] B. Mora, F. García, F. Ruiz *et al.*, "SMML: Software Measurement Modeling Language," in OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA, 2008.

[54] M. A. V. N. Marcos Rodrigo Sol Souza, "A Domain-Specific Language for Interoperability Between Object-Oriented and Mainframe Systems," in Workshop on Domain-Specific Program Development (DSPD), Nashville, Tennessee, USA, 2008.

[55] J. Hosking, N. Mehandjiev, and J. Grundy, "A Domain Specific Visual Language for Design and Coordination of Supply Networks," *IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 109-112, 2008.

[56] Y. Teiken, and S. Floring, "A Common Meta-Model for Data Analysis based on DSM," in OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA, 2008.

[57] J. Merilinna, "Domain-Specific Modelling Language for Navigation Applications on S60 Mobile Phones," in OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA, 2008.

[58] T. Patki, H. Al-Helal, J. Gulotta *et al.*, "Using Integrative Modeling for Advanced Heterogeneous System Simulation," in OOPSLA Workshop on Domain-Specific Modeling, Nashville, Tennessee, USA, 2008.

[59] T. Reichert, E. Klaus, W. Schoch *et al.*, "A Language for Advanced Protocol Analysis in Automotive Networks," in ICSE, Leipzig, Germany, 2008, pp. 593-602.

[60] Z. Hemel, R. Verhaaf, and E. Visser, "WebWorkFlow: An Object-Oriented Workflow Modeling Language for Web Applications," in MoDELS '08: International conference on Model Driven Engineering Languages and Systems, Toulouse, France, 2008, pp. 113-127.

[61] R. Tairas, S.-H. Liu, F. Jouault *et al.*, "CoCloRep: A DSL for Code Clones," in Workshop on Software Language Engineering, Nashville, Tennessee, USA, 2007.

[62] M. Barbero, J. Bézivin, and F. Jouault, "Building a DSL for Interactive TV Applications with AMMA," in Model-Driven Development Tool Implementers Forum, Zurich, Switzerland, 2007.

[63] R. Pohjonen, and S. Kelly, "Interactive Television Applications using MetaEdit+," in Model-Driven Development Tool Implementers Forum, Zurich, Switzerland, 2007.

[64] D. Correal, and R. Casallas, "Using Domain Specific Languages for Software Process Modeling," in OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada, 2007.

[65] J. Merilinna, and J. Pärssinen, "Comparison Between Different Abstraction Level Programming: Experiment Definition and Initial Results," in OOPSLA Workshop on Domain-Specific Modeling, Montréal, Canada, 2007.

[66] F. Jouault, J. Bézivin, C. Consel *et al.*, "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.

[67] J. Zeng, C. Mitchell, and S. A. Edwards, "A Domain-Specific Language for Generating Dataflow Analyzers," in Workshop on Language Descriptions, Tools and Applications, Vienna, Austria, 2006.

[68] H. Prähofer, D. Hurnaus, and H. Mössenböck, "Building End-User Programming Systems Based on a Domain-Specific Language," in OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA, 2006.

[69] N. Bencomo, and G. Blair, "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.

[70] K. Bierhoff, E. S. Liongosari, and K. S. Swaminathan, "Incremental Development of a Domain-Specific Language That Supports Multiple Application Styles," in OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA, 2006.

[71] A. W. B. Furtado, and A. L. M. Santos, "Using Domain-Specific Modeling towards Computer Games Development Industrialization," in OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon, USA, 2006.

[72] J. Luoma, S. Kelly, and J.-P. Tolvanen, "Defining Domain-Specific Modeling Languages: Collected Experiences," in OOPSLA Workshop on Domain-Specific Modeling, Vancouver, British Columbia, Canada, 2004.

[73] J. Sprinkle, and G. Karsai, "A domain-specific visual language for domain model evolution," *Journal of Visual Languages & Computing*, vol. 15, no. 3-4, pp. 291-307, June-August, 2004.

[74] J. C. Grundy, J. G. Hosking, R. W. Amor *et al.*, "Domain-specific visual languages for specifying and generating data mapping systems," *Journal of Visual Languages & Computing*, vol. 15, no. 3-4, pp. 243-263, June-August, 2004.

[75] L. Howard, "CAPE: A Visual Language for Courseware Authoring," in OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA, 2002.

[76] J. Bettin, "Measuring the potencial of domain-specific modelling techniques," in OOPSLA Workshop on Domain-Specific Visual Languages, Seattle, Washington, USA, 2002.

[77] J. Gray, T. Bapty, and S. Neema, "An Example of Constraint Weaving in Domain- Specific Modeling," in OOPSLA Workshop on Domain-Specific Visual Languages, Tampa-Bay, Florida, USA, 2001.

[78] F. Shull, V. Basili, J. Carver *et al.*, "Replicating software engineering experiments: addressing the tacit knowledge problem," *International Symposium on Empirical Software Engineering*, vol. 0, pp. 7-16, 2002.

[79] F. Shull, M. G. Mendonça, V. Basili *et al.*, "Knowledge-sharing issues in experimental software engineering," *Empirical Software Engineering, International Symposium on*, vol. 9, pp. 111-137, 2004.

[80] J.-P. Tolvanen. "Domain-Specific Modeling: How to Start Defining Your Own Language," 2006; http://www.devx.com/enterprise/Article/30550.

[81] R. G. Sargent, "Validation and verification of simulation models," in Winter Simulation Conference, Los Alamitos, CA, USA, 2004.

[82] J. Aranda, N. Ernst, J. Horkoff *et al.*, "A Framework for Empirical Evaluation of Model Comprehensibility," in International Workshop on Modeling in Software Engineering, 2007.

[83] A. E. Bobkowska, "A Framework for Methodologies of visual Modeling Language Evaluation," in MIS '05: Proceedings of the 2005 symposia on Metainformatics, Esbjerg, Denmark, 2005, pp. 2.

[84] J. Nielsen, "Usability Inspection Methods," in Conference on Human Factors in Computing Systems, Boston, Massachusetts, United States, 1994, pp. 413 - 414.

[85] B. A. Kitchenham, "Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods," *SIGSOFT Software Engineering Notes*, vol. 21, pp. 11--14, 1996.

[86] J. D. Gould, and C. Lewis, "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM,* vol. 28, pp. 300-311, 1985.

[87] J. L. Mathe, J. B. Martin, P. Miller *et al.*, "A Model-Integrated, Guideline-Driven, Clinical Decision-Support System," *IEEE Software,* vol. 26, pp. 54-61, 2009.

[88] J.-P. Tolvanen. "Speak the Language," 2009; http://www.medicaldevice-network.com/features/feature59141/.

[89] J.-P. Tolvanen, and C. Giese. "Modeling for Full Code Generation? Modeling for Software Development in the Automotive Industry," 2007; http://www.metacase.com/papers/Eclipse_Magazine_Volume12.pdf.

[90] A. Hulshout, and J.-P. Tolvanen. "Modeling for full code generation," 2007; http://www.embedded-computing.com/pdfs/MetaCase.Aug07.pdf.

[91] M. Y. Ivory, and M. A. Hearst, "The state of the art in automating usability evaluation of user interfaces," *ACM Computing Surveys,* vol. 33, pp. 470-516, 2001.

[92] V. Sugumaran, S. Park, and K. C. Kang, "Software Product Line Engineering," *Communications of the ACM,* vol. 49, December, 2006.

[93] J. White, J. H. Hill, J. Gray *et al.*, "Improving Domain-specific Language Reuse with Software Product-line Techniques," *IEEE Software,* vol. 26, pp. 47-53, 2009.

[94] G. Karsai, H. Krahn, C. Pinkernell *et al.*, "Design Guidelines for Domain Specific Languages," in OOPSLA Workshop on Domain-Specific Modeling, Orlando, Florida, 2009.

[95] J. Kärnä, J.-P. Tolvanen, and S. Kelly, "Evaluating the Use of Domain-Specific Modeling in Practice," in OOPSLA Workshop on Domain-Specific Modeling, Orlando, Florida, 2009.

[96] A. Seffah, M. Donyaee, R. B. Kline *et al.*, "Usability measurement and metrics: A consolidated model," *Software Quality Journal,* vol. 14, pp. 159-178, June, 2006.

[97] F. Hermans, M. Pinzger, and A. v. Deursen, "Domain-Specific Languages in Practice: A user Study on the Success Factors," in International Conference on Model Driven Engineering Languages and Systems (MODELS), Denver, CO, USA, 2009, pp. 423-437.

[98] M. Jimenez, F. Rosique, P. Sanchez *et al.*, "Habitation: A Domain-Specific Language for Home Automation," *IEEE Software,* vol. 26, pp. 30-38, 2009.

[99] R. B. Kieburtz, L. McKinney, J. M. Bell *et al.*, "A software engineering experiment in software component generation," *International Conference on Software Engineering*, vol. 0, pp. 542, 1996.

[100] C. Schmidt, B. Cramer, and U. Kastens, "Usability Evaluation of a System for Implementation of Visual Languages," in VLHCC '07: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2007, pp. 231-238.

[101] N. G. Trillo, "The Cultural Component of Designing and Evaluating International User Interfaces," in Hawaii International Conference on System Sciences, Hawaii, 1999.

[102] J. Caldwell, "Safety Icons and Usability: a Peircean Reanalysis," *International Professional Communication Conference*, vol. 0, pp. 1-8, 2009.

[103] J. Rumbaugh, "Notation notes: Principles for choosing notation," *Journal of Object-Oriented Programming*, vol. 9, pp. 11-14, 1996.

[104] A. F. Blackwell, "Metaphor in Diagrams," PhD thesis, Darwin College, Univ. of Cambridge, 1998.

[105] M. Ciolkowski, O. Laitenberger, and S. Biffl, "Software Reviews: The State of the Practice," *IEEE Software*, vol. 20, pp. 46-51, 2003.

[106] A. Blackwell, and T. Green. "A Cognitive Dimensions Questionnaire," 2007; http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf.

[107] J. Nielsen. "Why You Only Need to Test with 5 Users," http://www.useit.com/alertbox/20000319.html.

[108] F. P. Brooks, *The Mythical Man-Month*: Addison-Wesley Longman Publishing Co., Inc., 1995.

[109] S. L. Pfleeger, and B. A. Kitchenham, "Principles of survey research," *ACM SIGSOFT Software Engineering Notes*, vol. 26, pp. 16-18, 2001.

[110] M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering*, vol. 5, pp. 201-214, 2000.