Patterns for Evaluating Usability of Domain-Specific Languages

Ankica Barišić, Pedro Monteiro, Vasco Amaral, Miguel Goulão, Miguel Monteiro
CITI, Departamento de Informática
Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa
2829-516 Caparica, Portugal
+351 212 948 536
a.barisic@campus.fct.unl.pt, pmfcm@campus.fct.unl.pt,
vma@fct.unl.pt, mgoul@fct.unl.pt, mtpm@fct.unl.pt

ABSTRACT

For years the development of software artifacts was the sole domain of developers and project managers. However, experience has taught us that the Users play a very important role in software development and construction. On Domain Specific Languages the inclusion of the domain experts directly in the development cycle is a very important characteristic, as they have often an important role in making and constraining the domain of the language.

DSLs are credited with increased productivity and ease of use, but this fact is hardly ever proven. Moreover, usability tests are frequently only performed at the final stages of the project when changes have a significant impact on the budget. To help prevent this, in this paper we present a pattern language for evaluating the usability of DSLs. Our patterns can help show how to use an iterative usability validation development strategy to produce DSLs that can achieve a high degree of usability.

KEYWORDS: Pattern Language, Domain-Specific Language, Software Language Engineering, Usability Evaluation

1. INTRODUCTION

An increasing number of people rely on software systems to perform their daily routines, work-related and personal tasks. As such, the number of software systems has risen greatly in the last few years and new products need to be developed rapidly so as to satisfy the demand. Domain-Specific Languages (DSLs) arise in this context as a way to speed up the development of software by restricting the application domain and reusing domain abstractions. Thus, DSLs are claimed to contribute to a productivity increase in software systems development, while reducing the required maintenance and programming expertise. The main purpose of DSLs is to bridge the gap between the Problem Domain (essential concepts, domain knowledge, techniques, and paradigms) and the Solution Domain (technical space, middleware, platforms and programming languages).

However intuitive this idea might be, we need to have means to assess the Quality and success of the developed languages. Not embracing quality assessment is to accept the risk of building inappropriate languages that could even decrease productivity or increase maintenance costs.

Software Language Engineering (SLE) is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages. One of the crucial steps in the construction of DSLs is their validation. Nevertheless, this step is frequently neglected. The lack of systematic approaches to evaluation, and the lack of guidelines and a comprehensive set of tools may explain this shortcoming in the current state of practice. To assess the impact of new DSLs we could reuse experimental validation techniques based on User Interfaces (UIs) evaluation as DSLs can be regarded as communication interfaces between humans and computers. In that sense, using a DSL is a form of Human-Computer Interaction (HCI). As such, evaluating DSLs could benefit from techniques used for evaluating regular UIs.

We reviewed current methodologies and tools for the evaluation of UIs and General Purpose Languages (GPLs), in order to identify their current shortcomings as opportunities for improving the current state of practice [1]. That brought us closer to providing adequate techniques for supporting the evaluation process which, we argue, should be based on methods for assessing user experience and customer satisfaction, applied to DSL users. By promoting DSL usability to a priority in the DSL development, usability must be considered from the beginning of the development cycle.

One way of doing this is through user-centered methods [2], i.e. placing the intended users of a language as the focal point of its design and conception, thus making sure the language will satisfy the user requirements. In order to tailor such methods to DSL development, we need to establish formal correspondences for all stages between the DSL development process and the Usability evaluation process [1].

Patterns represent tangible solutions to problems in a well-defined context within a specific domain and provide support for wide reuse of well proven concepts and techniques, independent from methodology, language, paradigm and architecture [3]. Thus, using patterns, we aim to disseminate the knowledge of this best practice to both expert and non-expert language engineers, easing the adoption of these solutions in other systems.

The rest of this paper is organized as follows: In section 2 we present our pattern language and the patterns that compose it. In section 3 we describe related work, while in section 4 we conclude and discuss future work.

2. PATTERN LANGUAGE

A pattern language is a set of inter-dependent patterns that provide a complete solution to a complex problem [3].

Our pattern language is divided into two design spaces: Process and Organization and User-Centered Language Development (see Figure 1).

- **Process and Organization:** This design space considers patterns devoted to Project Management and Engineering of a Domain-Specific Language. This is the most important design space because it is through *Process and Organization* that the language engineers access the remaining design spaces.
- **User-Centered Language Development:** The users are the central part of a DSL. This design space considers how to engage the user in the development process and how to collect valuable information about the DSL and its level of usability while it is being developed.

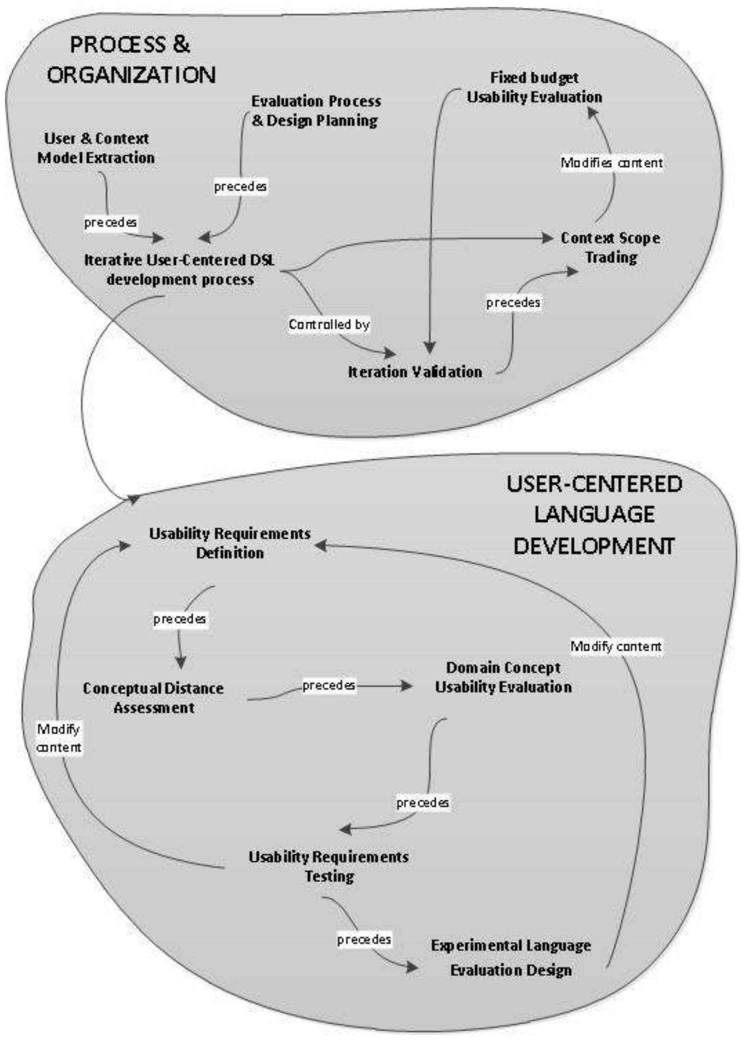
PROCESS & ORGANIZATION
User & Context Model Extraction
Evaluation Process & Design Planning
Fixed budget Usability Evaluation
precedes
precedes
Modifies content
Iterative User-Centered DSL development process
Context Scope Trading
precedes
Controlled by
Iteration Validation
USER-CENTERED LANGUAGE DEVELOPMENT
Usability Requirements Definition
precedes
Modify content
Conceptual Distance Assessment
precedes
Domain Concept Usability Evaluation
Modify content
precedes
Usability Requirements Testing
precedes
Experimental Language Evaluation Design

Figure 1 - Patterns for Evaluating Usability of Domain-Specific Languages

2.1. Pattern Language Terminology

Language Engineer is a professional who is skilled in the application of the engineering discipline to the creation of software languages. Language engineers design the software language and are responsible for making it functional at the system level. They are involved in the language specification, implementation, and evaluation, as well as providing templates and scripts [4].

End User is a person who uses software languages to create applications [4] (e.g. application developers). In domain-specific modeling the possible user base of the models can easily be broader, as it allows application users to be better involved in the application development process. In that case customers, other than typical application developers, can read, accept and in some cases change application specifications, being directly involved in the application development process. End user can work with models that apply concepts directly related to specific characteristics of configuration, like specifying deployment of software units to hardware or describing high-availability settings for uninterrupted services with redundancy and reparability for various fault-recovery scenarios. Yet another group of users is responsible for specifying services that are then executed in the target environment [5].

Domain Expert is a person that is involved in the language development process, sometimes known as a knowledge engineer. In the case of domain-specific modeling they do not need to have software development background, but they can specify application for code generation. They can specify models for concept prototyping or concept demonstration, and language engineers can proceed from these models. In contrast with end users, they should have domain knowledge that includes areas of all target model applications.

Usability is the quality characteristic that measures the ease of use of any software system that interacts directly with a user. It is a subjective non-functional requirement that can only be measured directly by the extent with which the functional architecture of the language satisfies users' needs based on their cognitive capacity. It focuses on features of the human-computer interaction.

Usability is result of the achieved level of Quality in Use of a software system i.e. a user's view of Quality. It is defined by ISO 9241 as *"the extent to which a product [service or environment] can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*[6]. It is dependent on achieving the necessary external and internal quality that is influenced by achievement of different quality attributes dependent on context of use. Tests of language usability are based on measurements of the users' experiences with it.

Productivity is the ratio between the functional value of software produced to the labor and expense of producing it. It is considered that good systems analysis enhances software productivity and software productivity is a success measure of systems analysis. Measure of productivity is based on the value of results of the software use. The high level of usability directly increases productivity of software.

Productivity metrics need to capture both the effort required to produce the software and the functionality provided to the software user. This measures should give software managers and professionals a set of useful, tangible data points for sizing, estimating, managing, and controlling software projects with rigor and precision [7].

User-centered methods are comprised in user-centered design of software product at different points of the product development lifecycle. They include user-centered techniques such as ethnographic research, participatory design, focus group research, surveys, walk through, preliminary prototyping, expert or heuristic evaluation, usability testing, as well as follow up studies [2].

2.2. Physicist's EAsy Analysis Tool for High Energy Physics

In order to exemplify the proposed pattern language, we will take an existing DSL for High Energy Physics (HEP) called Pheasant (Physicist's EAsy Analysis Tool), developed using some of the methods described in this paper. A detailed description of Pheasant can be found in [8].

In the context of High Energy Physics (HEP), physicists try to discover new short-lived particles and their properties or the properties of their interactions, in order to develop a model of the real world at a subatomic level. Large accelerators accelerate subatomic particles to induce collisions. These collision events are recorded by sub-detectors which measure and analyze the results. Afterwards, the large volume of data collected by detectors is mined and used to try to infer statistical physics results, validating them against currently proposed Physics Models. The physicists' analysis systems are composed of a visualization tool, a set of scientific calculation libraries, and a storage manager. Traditionally, in a first step of his analysis, the user selects a subset of data from the storage manager. Then, several reconstruction algorithms with scientific calculations filter out data and compute new values that are stored in private collections. Finally, the new data collection is visualized in the visualization tools (for instance by histograms).

The reconstruction and investigation of decays and decay chains of short-lived particles are the main computationally demanding tasks of the data analysis, which starts after the data acquisition. Roughly speaking, this phase, physicists have to select those kind of decays and particles they are interested in. For this selection, it is usually necessary to reconstruct parts of the particles' trajectories (also called segments), to match them with other segments in order to reproduce the full particle trajectories (called tracks), to extract further properties, and to deduce the complete decay chain.

The Pheasant project was developed to mitigate user's productivity problems in this domain. It aimed to develop reusable engineering methodologies through Model-Driven Techniques. A declarative Domain Specific Visual Query Language (DSVQL) was used to raise the abstraction level in the existing query systems and give room to new optimizations of different levels. The goal of Pheasant was to automate as much as possible this process as well as providing the users (with different profiles ranging from totally non computer experts to high-level programmers) appropriate abstractions to hide the complexity of programming error prone algorithms in languages such as C, C++ or Fortran, using a wide plethora of libraries and frameworks to achieve their goals.

It served to confirm that the proposed query language tailored to the specific domain was beneficial to the end-user. The physicists, non-experts in programming, no longer were required to cope with different GPLs and adapt to the intricacies supporting database infrastructure.

The DSL developed through the Pheasant project is a good example of a pattern language to be illustrated in this paper, as it is a complete exercise for a DSL development and is designed with strong user feedback, focusing on understanding how the language is perceived, learned, and mastered. It also gives classification of users, categorizing them by identification of their specific requirements. The validation of the language through usability evaluation tests is included [9].

2.3. Process & Organization Patterns

2.3.1. User and Context Model Extraction

Problem Description: How to distinguish for which user profiles and contexts of use we have validated the DSL's usability level.

Context: The number of users involved in the usability evaluation of a DSL should be significant in relation to the actual number of intended DSL users. This means that, in the majority of cases, the number of user profiles and user contexts will also be relevant. This introduces the problem of knowing if all user groups are represented and how those user groups relate with the others and with the overall context and domain. Moreover, if usability is to be validated iteratively, the project manager and language engineer need to be able to manage and extract feedback from a large number of users on a regular basis.

Forces:

On-Budget Completeness: By building a complete user and context model we are able to control for which extent of targeted user population and environmental and technical range usability is achieved. However, this is hard to achieve on a strict budget and the development team should be aware that some requirements might only be identified at later stages.

User Coverage: It is sometimes easy to forget that, in general, a DSL is intended to be useful for only a relatively small set of users and not a wide range of them. When designing a language we must pay close attention not to place too much effort in satisfying requirements of non-target users.

Solution: Before building a new DSL we should identify all intended user profiles and target context of use. These user groups should be characterized by their background profiles and domain expertise, as well as different stakeholder positions in solving problem groups. These general user characteristics should be weighted according to its relevance, which will influence the relevance level of each chosen test user group.

Also, in the same way we should define a complete context model that will contain all technology variations that will be possible to use, equipment availability, additional software support and its compliance to new system, as well as intended working environments and its effect on using a system.

Building the context and user model should be emerged with Domain analysis phase of development the DSL. As the new domain concepts are identified for the DSL, potential users of this concept, and context of use should be defined.

Example: The user model is obtained by identifying the list of main characteristics that can help in categorizing the set of user groups. For the case of Pheasant (see

Table 1) these characteristics are prioritized with a Likert scale representing an evaluation importance weight ranging from 1 to 5, where 1 means ‘Unimportant’ and 5 means ‘very important’. After indicating these weights, it becomes trivial to extract important user models that need to be evaluated.

Table 1 – List of user characteristics

Technical characteristics				**Profile characteristics**				**Stakeholder**				**Personal characteristics**	
Knowledge about HEP experiments	5			Physicist	5	- experimenter	5	Experiment Role	5	- Experiment designer	5	Analytical thinking	5
Knowledge of particle physics	4					- theoretician	1			- Analyst performer	4	Logic reasoning	4
Knowledge of programming	3	- querying	4	Engineer	4			Academic Title	4	- Professor	5	Sight problems	1
		- c programming	4	Programmer	3					- PhD	4		
		- Fortran programming	2							- Master student	4		
		- c++ programming	2							- PostDoc	3		

This weight hierarchy will become increasingly detailed with each new iteration. For instance, if the main profile observed is that of a physicist, we need to find details which help to isolate specific characteristics, thus creating sub profiles. In the case of Pheasant, we are interested in physicists who 1) have knowledge of HEP experiments and particle physics, and 2) have knowledge of programing and querying.

The context model details the user's working equipment. As Pheasant is meant to be used from computers, it is essential to describe the scope of computer characteristics (see Table 2). This allows us to reason about whether any usability issues detected in the language can be traced to inappropriate equipment or working environment. User working environment can also cause user to obtain lower results during use of language, so it is important to describe and control main environment equipment.

Table 2 – Users working equipment and environment

Users Working Equipment						Users Working Environment	
office computer as working platform	processor power	capacity:	2GHz-3,6GHz	number of cores	2-4	working desk	5
	RAM	capacity:	2GB-8GB			chair	5
	internal storage	capacity:	250GB-2TB	number of discs:	1-4	windows	3
	monitor	size:	20''-24''	color:	yes	office lights	4
	network	capacity:	X	wireless,wired	offline	aircondition system	3
	power	range:	550W-750W			heating machine	3
	office electrical power system:			secondary power:			
	keyboard:		optical				
	mause:		optical				
office computer as connection device to clustered system	processor power	capacity:	2GHz-3,6Ghz	number of cores:	2		
	RAM	capacity:	1GB-4GB				
	internal storage	capacity:	120GB-1TB	number of discs:	1-2		
	monitor	size:	20''-24''	color:	yes		
	network	capacity:	112Mbs -1Gbs	wireless,wired	online		
	power	range:	300W-550W				
	electrical power system:			secondary power:			
	keyboard:		optical				
	mause:		optical				

Also, it is important to characterize the language operating environment to which we target the desired usability levels (see

Table 3). As it may be too expensive to perform testing with all language operating environments configurations, one should assign different priorities for different configurations, so that at least the most important configurations are tested.

Table 3 – Language operating equipment and environment

Language Operating Equipment		Operation System Environment			
Detector	1	**OS**	Linux	UNIX	5
Storage	5		Windows	Dos	3
Calculation libraries	5		Mackintosh	MAC OS	4
Robotic tape	2	**Visualization Tool**		JAS	5
Accelerator	1	**Framework**	Fortran	PAW	4
				ARTE	4
				ROOT	3
				ARTE	4
			C++	BEE	5

Related Patterns:

Iterative User-Centered DSL Development Process: To begin the development process, it is imperative that the language engineer proceeds with *User and Context Model Extraction*.

Evaluation Process and Design Planning: While the language engineer gathers resources for the user and context model, a plan for evaluation should also be considered.

Known uses: In usability testing one of main problems for achieving usable products is that development focuses mainly on the machine or system, not considering the human aspects of software. There are three major components that should be considered in any type of human performance situation: Activity, Context and Human. Designers should focus on all three elements during development [2]. Benefits of user and context modeling on management and final product are confirmed in areas of service and interface development.

2.3.2. Evaluation Process and Design Planning

Problem Description: How to plan the processes of usability evaluation experiments and control the adequacy of the produced solutions to the intended users and respective context models.

Context: Usability evaluations and experimental designs should be carefully planned through an experimental process model. Planning is a time consuming task and if not done carefully induces the risk of spending resources on usability evaluations with questionable validity and usefulness.

Forces:

- **Planning and Control:** Through good and careful planning the language engineer becomes more able to control and validate results, and to know the scope of their impact.
- **Reusability:** Results, if packaged correctly, can be reused or replicated on another solution or similar context as long as adequate measures for each context are controlled and validated. However, it becomes easier to reason about the impact of recommendations that resulted from each experiment and reuse these conclusions for another evaluation session.
- **Balance user need validity and budget:** from the users view point all wished features and requirements are valid and essential. However, not all features fall within budget and

not all users have the same level of importance to push new features.

- **Experimental evaluation cost:** There is a tension between the cost of a full-blown experimental evaluation and the need to make short delivery sprints.

Solution: When planning the process of Usability evaluations and experimental designs, the language engineer must document the main problem statements and the relations with intended experiments. The documentation should include initial sample modeling (considering all possible samples, groups, subgroups, disjoint characteristics, etc.), context modeling, instrumentation (*e.g.* type of usability tests and when to use them), the instrumentation perspectives (*e.g.* cognitive dimensions fundamental to assessing usability) and their relation with metrics acquired through data analysis and testing techniques.

To assess the validity of results that will lead us to reason about usability of the domain-specific solution, language engineers should carefully plan the process and extent of experiments. The main problem statements and intended usability experiments should be designed with care, to ensure replicability, and to control the result of alterations.

Example: In this pattern we need to identify and prioritize all goals of the language, as well as the goal of the evaluation. The goals for Pheasant are described in Table 4.

These goals will later be used to control which goals were addressed by the problem statements of experiments and the heuristic evaluations.

Table 4 – Goal lists

System goals	Evaluation goals
Deal with petabytes of data.	Query steps in Pheasant vs the object-oriented coding
Support hundreds of simultaneous queries.	Aggregation
Return partial results of queries in progress	Expressing a decay
provide interactive query refinements.	Specification of filtering conditions
Deal with data on secondary and tertiary storage access for simultaneous queries	Vertexing and the usage of user-defined functions
Support statistical selection mechanisms (uniform random sampling)	Path expressions (navigational queries)
Provide a flexible schema which supports versioning	Expressing the result set
Provide an environment for data analysis that is identical on desktop workstations and centralized data repositories.	The expressiveness of user-defined functions

Goals are fulfilled by executing tasks ans so we need to list and prioritize tasks, to further help us decide how to design instrumentation and metrics to capture this special tasks (see

Table 5)

As the goal of Pheasant is to obtain better querying than in the previos approaches, it is important to list comparation elements that should be addressed during evaluation (see Table 6).

Table 5– Task list

Query tasks		User tasks	Cognitive tasks
Run/tag selection:	- Trigger selection	Inform status	Query writing
	- Run period	Write query	Query reading
event Selection:	- Filled bunch	Save query	Query interpretation
	- No coasting beam	Load query	Question comprehension
	- No empty events	Generate code	Memorization
	- Refined confirmation of the trigger	Undo/Redo	Problem solving
Reconstruction:	- Track selection	Execute	
	- Particle ID filter condition	Get Query results	
	- Combination of tracks	Define Shema	
	- Vertexing	DefineUDF	
	- Kinematic or geometric filter conditions	Define constants	
Histogramming and/or comparison with Monte Carlo Simulation			

Table 6 – List of comparison elements

Comparision elements	
textual v.s. graphical syntax	general purpose v.s. domain specific
Expressive	Readability
Easy to learn	Assesability
Syntax error Free	design reuse
Semantics error Free	high-level abstraction
Small Conceptual distance	clarity of program specification
Memorizable	program checkin
Easy to use	language performance
Non-Ambigous	Maintainability
Formalizable	Portability
	Effectivness

Capture test
Final exams
Immediate comprehension
Reviews
Productivity
Retention
Re-learning

Related Patterns:

Iterative User-Centered DSL Development Process: Developing an evaluation plan of action is an important starting point for iterative development.

Evaluation Process and Design Planning: A plan for evaluation should be considered side by side with the user and context model it intends to evaluate.

Known uses: Identifying and controlling evaluation process and design trough set of tasks, evaluation goals, and different test approaches is a common approach for evaluating experience in using any product or service. Examples of its use can be found in assessments of customer satisfaction, evaluation of public opinion, evaluation of psychological capabilities in human resources, as well as in evaluation of user interfaces. Detailed example of practical application to query languages can be seen in [10].

2.3.3. Iterative User-Centered DSL Development Process

Problem Description: How to ensure that the domain-specific solution will result in high level of the DSL users' productivity when compared to the existing baseline.

Context: When developing a new DSL, the development cycle is intertwined with scheduled deliveries of incremental versions of the DSL. Since the focus of development is usually on the delivery time and functionality rather than the users it is usual to attain a solution which is not as usable as it should or could be.

Forces:

- **Cost of Usability Control vs. Cost of Future modifications:** if we do not control usability during the several development stages, essential usability failures may lead us to meta-level changes that are equivalent to development from scratch.
- **Development Cost:** Developing any language is a very expensive endeavor, more so because of the need to ensure that in the end we will get a highly usable language.

Solution: To prove the long claimed productivity increase provided by introducing DSLs, Language Engineers need to ensure that high usability level of produced DSL is achieved.

In order to do this, and in turn increase the chances of adoption by users within the domain, the language engineers should embed user-centered design activities within the DSL development process itself. It is important to involve domain experts and end-users in the development process, therefore empowering them to drive the project. However, executives and users of the language models should be involved but not overly committed to it, as users will quickly become afraid of being accountable for eventual project mishaps.

Each iteration of the development cycle should be combined with a user-centered design activity where usability requirements are defined and validated through constant interaction with target user groups. This means that the user becomes an invaluable part of the development process and receives some measure of responsibility over the outcome of language design and development.

Example: Like the pattern explains, we should build a schedule of all iterations at the beginning, clearly identifying participants and what features are to be tested. At each passing iteration we can then re-prioritize the remaining iterations according to what was accomplished.

These schedules should also include careful aproximations of how much time and how many participants will be involved in active work on the usability evaluation. This includes the time that is required to make guidelines, list requirements, choose metrics, perform focused workshops to discuss the results, analysis of results and so on. An example of a one such schedules is shown in

Table 7.

In this case, the set of Pheasant iterations can be seen as a single development cycle step after which, if additional development was required, we would have similar usability iterations inside a new cycle with the new product in use. On this next cycle, the schedule would be easier to predict since they would be based on the numbers from the previous cycle. This gives the language engineers the means to control the cost of evaluation.

Table 7 – Evaluation iteration description

ITERATION	DESCRIPTION	OUTPUTS	PERSONS		TIME
1st	heuristic analysis of implemented features with domain expert	list of typical tasks user want to perform with the language	usability expert	1	200h
		list of usability problems from previous cycle	domain experts	1-2	
		list of beneficial usability aspects from previous approach	language engineer	1-2	
2nd	heuristic analysis of implemented features with usability expert	checklist of usability for interfaces (ref)	usability expert	2-3	40h
		specification list of element structure, position, etc.	domain expert	1	
			language engineer	1-2	
3rd	usability analysis of language metamodel quality	List of language semantic clones	usability expert	1	60h
		List of language syntactic clones	domain expert	1	
			language engineer	2	
4th	pilot test for the first experimental evaluation with users	list of features that need to be rechecked	usability expert	1	120h
		list of tasks to perform with language	domain experts	2-3	
			language engineer	1-2	
5th	experimental evaluation with users following experiment design	list of detailed task and usability elements	usability expert	1	120h
		metrics specification	subjects	14-24	
			language engineer	1-2	

As expected, the 200 hour requirement of the first iteration includes the time needed to prepare and estimate the first evaluations. The following iterations require considerably less time as they are base in the previous ones.

Related Patterns:

> **Iteration Validation:** Each iteration should be followed by a validation stage where the output of the iteration is validated against expectations.
>
> **Context Scope Trading:** Allows the analysis of what should be done in the next iteration.
>
> **User-centered language development:** At the level of this pattern all patterns within the *User-centered language development* design space should be considered.

Known uses: The usability engineering lifecycle is iterative by itself and should be merged with development of any product [11]. Involvement of user-centered techniques in iterative development of software product is becoming common, and examples vary from user interfaces to service oriented applications [12].

2.3.4. Iteration Validation

Problem Description: How to control which usability problems were solved, and analyze their possible relation with new ones that could arise.

Context: As the DSL development process progresses and the number of features increases, it is easy to lose track of intermediary goals. It then becomes increasingly important to validate what has been accomplished at each iteration and measure how far we are to our true goal of a usable DSL.

Forces:

New features vs. fixes: During development, it is frequent to discover new requirements that the user considers of importance. It is up to the Language Engineer and project manager to decide if these are considered new features or fixes to improve usability. The latter should have top priority while the former should be carefully analyzed and sized.
Featurism vs. usability: The language engineer should clearly define the line where the number of features begins to jeopardize usability rather than promoting it.

Iteration Validation schedule: The validation itself should be short and concise, so as to not overstep the boundaries of the current iteration's development schedule. However it should be dense enough to allow the least amount of work to be postponed for additional iterations.

Regression Testing: At each iteration evaluation is focused mainly on new features of the language but, as the language is growing incrementally, it ends up re-covering language details addressed in previous iterations. This is essential to ensure that new features don't deem previous features unusable, however there is also a cost associated with retesting every previously tested feature. In this case the requirement is that at key iterations, when a new stable major version of the language is developed, testing and validation is performed on the full set of language features and not only on those newly added.

Solution: Although DSLs are developed in constant interaction with domain experts, by validating the iterations in *Time Box* fixed intervals we can monitor progress and check if it is going in the desirable direction. If it is not, developers are able to react to possible problems on time. At any point during language development, new requirements may arise and it is the job of the language engineer to evaluate them from a language point-of-view, while the project manager is required to analyze and frame the new requirements into the time-box. The length of the project itself should not be allowed to extend over the intended deadline or to surpass the original budget except on very specific cases when the new requirements translate into make-or-break features that cannot fit into the original project scope. Nonetheless, every change in the project has to be carefully analyzed and a compromise must be reached with the decision-maker stakeholders.

If *Iteration Validation* is not completed at least every few iterations, when the number of features developed is enough to warrant user tests, then there is a higher risk of failure of iterative development.

Time boxing is concluded with a progress report and with documenting results of the validations in an *Iteration Assessment* that consists of:

- A list of features that obtained required level of usability
- A list of usability requirements that were not addressed
- A list of usability requirements that need to be reevaluated or that represent new requirement items

This should be done through explicit communication with all relevant stakeholders of the validated iteration.

Example: Picking up Pheasant's 5th iteration from

Table 7, validation of the iteration is accomplished by defining what features were addresses and successfully implemented and which still require some work (see Table 8). Understanding the status of usability evaluation for the current iteration allows us to redesing the schedule for the next few iterations.

Table 8 – Iteration validation

Validated	To be revalidated	Not addressed	Additional functionality
Expressing filter conditions	Path expressions	Enviromental equipment testing	Query reuse
Expressing and using vertexing	Expressing and using UDFs	Interface design heuristics from Microsoft	Query scripting
Expressing the result set	Different data schema feature		
Expressing a decay			
Structuring the query			

Related Patterns:

Validate Iterations: More than understanding if iterations are on track and re-working the following iterations accordingly, as the *Validate Iterations* pattern [13] suggests, *Iteration Validation* requires the project team to validate if usability remains a concern throughout every iteration.

Iterative User-Centered DSL Development Process: Validation is a part of the iterative development process.

Context Scope Trading: The output of *Iteration Validation* is fed into *Context Scope Trading* to allow the analysis of future iterations.

Fixed budget Usability Evaluation: Validation controls how the budget was spent to accommodate usability questions.

Known uses: Validating iterations of product development cycle is beneficial for controlling development of any end product. It makes clear what issues are addressed and reviles new requirements that are overseen in planning of first cycle, and keeps track of validated approaches. This methodology helps to justify new specifications for project management and involves their decisions trough project [13].

2.3.5. Context Scope Trading

Problem description: How to ensure that each development iteration remains focused on the user's needs while maintaining a short time frame.

Context: When working within a budget and time limit, it is hard to focus on all usability requirements at each iteration and continue to ensure a successful iteration outcome. Some requirements are bound to receive more attention than others and lengthy requirements tend to always get pushed to future iterations [14]. Although tempting, in medium/large projects it is impossible to take into account all intended user profiles, environmental dependences and domain concepts in a single iteration. It is up to the Language Engineer and Project Manager to decide the iteration scope and to recognize how to profit from short iterations bursts.

Forces:

> **In-loco user:** Working directly with representative user groups, will allow detecting early the majority of usability defects so that they can be fixed at a minimum cost.
>
> **Following Recommendations:** Following guidelines of recommendations for the most relevant quality characteristics can be a time consuming task. However this will result in early adoption of best practices that will eventually contribute to a usable solution.
>
> **User Needs vs. Project Management:** Sometimes defining requirement priorities according to user needs goes against Project Management best practices. It is up to the language engineer and project manager to ensure that both goals are achieved within the same package.

Solution: Short iterations require short and well scoped contexts. Each iteration needs to precisely characterize the context that specific iteration will capture from the set of global context, intended users and domain solution.

To keep the user as the focus of each development iteration, the results of usability tests should be used to ensure that development prioritizes the most significant features, with focus on prioritized quality attributes and on the most representative user groups for the relevant context.

In order to effectively achieve this, each iteration should be preceded by a *Scope Trading Workshop* where all relevant stakeholders should come to an agreement on the context scope of the iteration. They should also agree on how the captured outcome of usability tests and experimental evaluations is to be handled.

The workshop should be used to:

- Assign a strict sequence of priorities to items in usability requirements list, depending on relevance of the domain concept's use-case;
- Identify the most relevant items from the backlog that should be solved in the next iteration;
- Reanalyze priorities of usability problems according to intended scope of user & context model;

This workshop should take place in the domain analysis phase, after validating iterations. Prior to the first iteration of the development process, identification of scope is achieved according to the extracted *user and context model* from the initial project plan. The intended scope of user and context model is analyzed more in depth after its definition during the workshop.

Example: Following the scope model defined in the *User and Context Model Extraction* (section 2.3.1), we define the Language Use Scope as given in Figure 2.

This scope is a subset of the scope defined in

Table 1 and Table 2, accounting for the fact that changes occurred in the set of available user groups and environment throughout the iterations. Using this new reduced scope and with the definition of evaluation for the iterations of the first cycle, as defined in

Table 7, we define the current Evaluation Scope as is shown in Figure 3.

Having defined this scope, it is easier to calculate the budget of the evaluation, and to design experimental evaluation focusing just on the given goals.

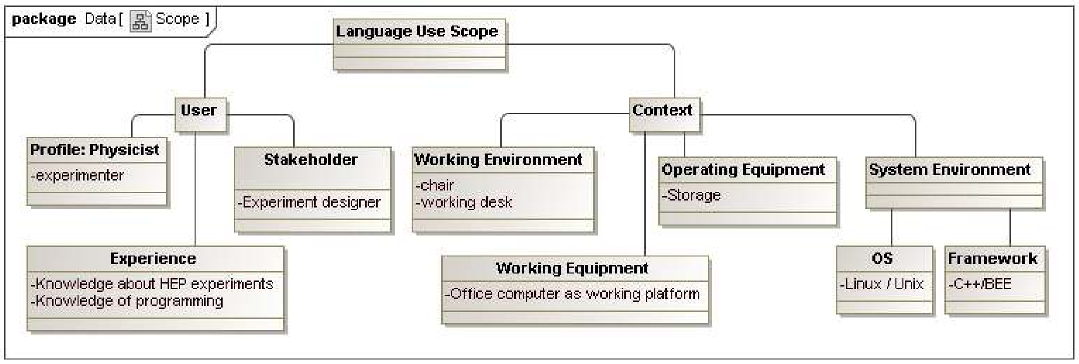

Figure 2 - Language Use Scope

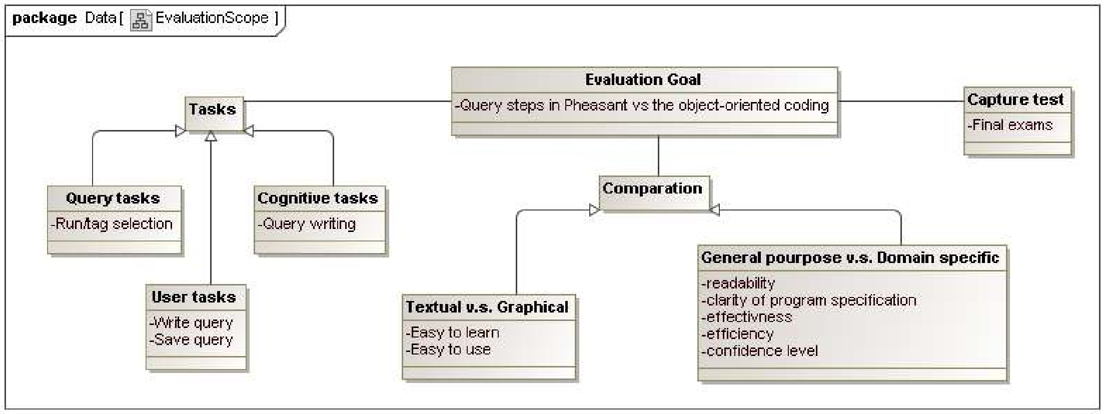

Figure 3 – Evaluation Scope

Related Patterns:

Scope Trading: These patterns are very similar in idea, however, while *Scope Trading* [13] relates more to strict requirements, *Context Scope Trading* can be seen as an extension of the original pattern to allow context trading considerations, which are valuable for DSLs.

Iterative User-Centered DSL Development Process: *Context Scope Trading* is a part of the development methodology of *Iterative User-Centered DSL Development Process.*

Fixed budget Usability Evaluation: The iteration scope defined within *Context Scope Trading* constrains what can and can't be done within budget limits.

Iteration Validation: The output of each validation stage is used to define what went wrong and if its solution is within budget.

Known uses: Scope trading on any product development method gives input means to its budget definition [13]. Any evaluation requires precise definition of its scope, in order to be able to validate its results and indicates trade-offs in design decisions [2].

2.3.6. Fixed budget Usability Evaluation

Problem description: How to maintain the budget within planned limits and ensure development results in a language with satisfying level of usability.

Context: We need to develop a usable DSL for a fixed budget. The abstract nature of the language and complexity of the domain knowledge prevents contractual details from capturing every aspect that needs to be considered for a language design and implementation that leads to a system that optimally supports users in their work.

Forces:

> **Scope vs. Cost: Usability** evaluation, its scope and context, should be wisely planned in order to minimize its cost but provide valid usability assurance.

Solution*:* The Language engineer should regularly validate iterations to user-drive the language under construction. However, in order to reduce the cost of Usability validation in each iteration the development team should focus on:

- Using short time-board iterations that concentrate on implementing main features first and drafts of additional ones
- Producing shippable DSLs in short iterations sprints. Since only a few features will be addressed in each iteration, the end result might have features which are left obviously unfinished and ambiguous. These unfinished features should act as motivators for user feedback.
- Getting 'live' feedback about unfinished features through brainstorming of possible solutions
- Producing first level applications and evaluate them with users, focusing to capture usability validations related to the language design

After each usability evaluation, Usability requirements that have failed validation must be annotated with clarifications, and listed alongside any new usability requirement that may have emerged during the last iteration. Subsequently the development team re-calculates realistic costs for all open usability requirements to enable scope trading and iteration sizing.

After a few such iterations, the work can be packaged and made available in the form of intermediary release. At this stage usability evaluation can/should be performed in real context of use with representative user groups, and language artifacts can be fully validated.

Example: Having defined the evaluation iterations of the first evaluation cycle, presented in Table 7, we can calculate and fix the budget for our evaluation cycles. This budget is recalculated after each iteration validation. Cost estimation is made easier by having detailed cost diagrams. This enables the project manager to compare the cost of each independent evaluation against the achieved result. Keeping this budget accounting also allows a more precise prediction of future costs.

Table 9 shows, for the first iteration cycle of Pheasant, how the budget evolved to encompass changes in iteration duration and cost estimation. At each passing iteration, the actual cost of the iteration was checked against the expected cost and budget corrections were made to the following iterations so that the project can be globally balanced. Having a well-balanced budget means that it becomes easier to know if the project is going according to what is expected.

Table 9 - Budget evolution for Pheasant

Initial Data		1st iteration			2nd iteration			3rd iteration			4th iteration			5th iteration		
Expected Work days	**A priori Estimation**	Days	Cost Estimation	Cost Correction	Days	Cost Estimation	Cost Correction	Days	Cost Estimation	Cost Correction	Days	Cost Estimation	Cost Correction	Days	Cost Estimation	Cost Correction
15	1.000 €	**17**	**1.050 €**	**+5%**	17	1.100 €	+5%	17	1.100 €		17	1.100 €		17	1.100 €	
20	1.200 €	21	1.250 €	+4%	**21**	**1.370 €**	**+10%**	21	1.370 €	0%	21	1.370 €		21	1.370 €	
25	1.500 €	25	1.550 €	+3%	25	1.670 €	+8%	**26**	**1.620 €**	**-3%**	26	1.620 €	0%	26	1.620 €	
32	2.100 €	32	2.150 €	+2%	32	2.270 €	+6%	32	2.220 €	-2%	**32**	**2.070 €**	**-7%**	32	2.070 €	0%
39	3.100 €	39	3.150 €	+2%	39	3.270 €	+4%	39	3.220 €	-2%	39	2.970 €	-8%	**42**	**2.970 €**	**0%**

One thing that must be noted in the budget of the successive iterations is that the number of expected work days also changes. This is an important fact as this indirectly influences both the monetary cost of the iteration and the scope of the following iterations.

Related Patterns:

Fixed Budget Shopping Basket: It is never enough to stress that it is important to keep a fixed budget for whichever iteration style. Fixed Budget Shopping Basket [13] details how to split the overall project development budget over all iterations.

Context Scope Trading: The iteration scope that is defined in turn constrains what can and cannot be done within budget limits.

Iteration Validation: The output of *Fixed budget Usability Evaluation* is used by *Iteration Validation* to understand if iterations are going according to plan.

Known uses: This pattern represents a concrete application of a method from risk management and analysis. It is used for lowering the risks that result from big project investments and provides various advantages such as requiring the contractor to be responsible for project design and development, as well as for legacy of the projects. Applicability of these models in scheduling and cost estimation of a fixed budget that is built in construction projects is shown to be very beneficial [15].

2.4. User-centered language development

2.4.1. Usability Requirements Definition

Problem Description: How to define expectations and desired quality in use of the intended DSL.

Context: When building a DSL we always have to keep in mind that the target outcome is to develop a language that can be used by a set of target users. These users originate from potentially different cultural backgrounds and have different responsibilities and motivations within the domain. That means that the perspective with which each end user of the language can look at it varies. By looking to the same language artifact, different stakeholders will mainly focus on a partial view of it, but all those partial views should be kept consistent. Features will have different importance to different stakeholders, shifting his interest to different measures of quality. We essentially need to have a way to keep all target user groups needs in mind when developing the language.

Forces:

> **Independent perspectives on quality:** Language engineers are able to reason about quality during development process. However, their perspective on quality does not necessarily match the perspective of other stakeholders, namely the DSL's end users. Failing to identify this mismatch may lead to a solution that does not meet the expectations of the DSL users.
>
> **Conceptual model:** Analysis of usability requirements can bring us closer to building a correct conceptual model of solution from the end-users point of view.
>
> **Language Comparison:** When surveying common used software tools in the domain it is very easy to end up comparing apples with oranges. Systematic studies of the tools of the trade need to be performed, placing careful consideration with the intended use of the different tools. Tools with slightly different applicability, even if used in the same restrict context of use should not be compared, unless the comparison takes into account these application dissimilarities. For instance, Microsoft Excel and the statistical software R can both be used to perform statistical analysis. However these are two very different tools and each excels in its own specific niche.

Solution: While building domain concepts, through direct interaction with domain experts it is valuable to collect background information of the intended users of each language concepts, to find what usability means to them.

The language engineer should formulate a survey, questionnaire or interview with intended user groups about their knowledge background and experience with previous approaches. This will help the language engineer to define a precise user and context model that should be the focus of the iteration cycle. While electing domain concepts, critical features that the user is concerned with should be identified and their relation with appropriate quality dimensions and attributes should be modeled. This model will later be used during experiment design to construct correct instruments, like questionnaires, to measure the distance between wished and achieved quality in use of provided solution.

In addition it is necessary to collect all data relating to the work environment and software products that are already in use to solve the problems inherent to the domain. It is important to identify characteristics that the users find that are useful, frustrating or lacking while using those products. In this way language engineers can find what quality means in the specific context of use for the DSL user profile.

Example:

In the case of Pheasant, one of the main requirements that motivated the project was the need to provide a more efficient and easier to learn query language, thus overcoming the problems of the previous approach. However, the new Pheasant queries needed to remain consistent with the underlying system framework, so that would not be necessary to change previously existing queries or future queries developed in other systems. The Pheasant language needed to be developed aiming to raise the level of abstraction in such a way that the end users could ignore individual query implementations of the different frameworks and in fact share their queries (i.e. have a way to talk about the specification of their queries without having to go deeply into the details of the programming environment).

In the following Table 10 we present the list of Usability requirements and tasks for Pheasant. They can be assessed at levels of Internal/External Quality and Quality in Use.

Table 10 – Usability Requirements

Usability Requirement	Description	Internal quality	External Quality	Quality in Use
Understandability	The language elements should be easy to understand, represented in terms that user is familiar with	Check consistency with physics notation they use		
	Language syntax elements should be easy to remember by the user			
Expressiveness	Provide simple way to present complex queries			
	Improved readability of queries			
Operability	Language actions and elements should be consistent			
	Error messages should explain how to recover from the error Undo should be available for most actions			
	Actions which cannot be undone should ask for confirmation			
	Prevent users from producing syntax errors (e.g. misspelling) Prevent users from producing semantic errors (e.g. query not behaving as the user expects it to)			
Functionality	Most frequent Querying task should be easy to do	Build concept element from most frequent tasks which have common logic	Count number of steps required to perform task	Measure time and number of mouse clics/keystrokes to perform the task
	Concepts that are parts of same task should be presented sequentially, following same logic	Sequence of domain concept relations should be analyzed against the tasks they belong to	Make sequence diagrams with domain concepts	Focus on repetitive operations of tasks and make sure they have the same use process
Learnability	The user documentation and help should be complete			Check how fast is user able to perform querying using help
	The help should be context sensitive and explain how to achieve common tasks for different types of users			Check if the user is able to reuse same concepts in different context.

	Language syntax elements should be easy to remember by user			Follow how recent user is asking help to find same concepts (operators, relation symbols)

The following diagram (Figure 4) shows how Internal/External Quality influences Pheasant's Quality in Use:

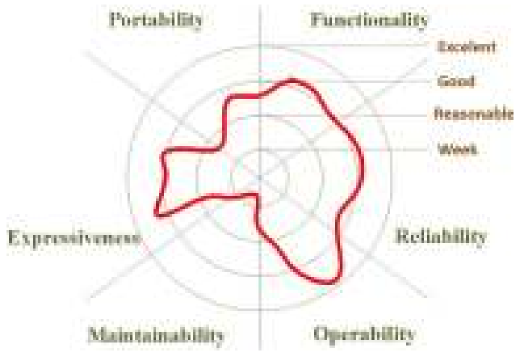

Figure 4 – Kiviat diagram of Internal/External Qualities for Pheasant

Related Patterns:

> **Conceptual Distance Assessment:** The requirements identified in *Usability Requirements Definition* are prioritized based on the quality attributes they impact.
>
> **Usability Requirements Testing:** Usability tests performed at each iteration are evaluated against the usability requirements so as to allow a definition that encompasses the current usability status of the language.

Known uses: Usability is seen as a special aspect in requirement engineering, of which the main phase is requirements definition [16]. Benefits of requirement engineering for model driven development approach can be seen in examples of software product lines, supporting traceability and contributing to flexibility and simplicity in development [17].

2.4.2. Conceptual Distance Assessment

Problem Description: How to measure conceptual distance between the user point of view to solve the problem and the provided solution.

Context: Extracting information from the users is a valuable source of data by which to measure the current status of ours solutions. However, to be able to analyze how each requirement impacts the DSL, we need to find a way to extract influential quality attributes.

Forces:

> **Quality Impact on Usability:** more than defining what quality attributes is important, it is essential to identify the quality attributes whose lack of actually impacts usability. That information should enable developers to produce pertinent usability metrics.

Solution: In order to understand how the design of the language's architecture impacts the usability requirements, the language engineer is required to elect quality attributes and connect them with domain concepts, creating a two-way relationship of <influences/is influenced by>.

Furthermore, for each domain concept and related usability requirement, we should identify both, its frequency and relevance within the domain. Weights should be assigned between the quality attributes and the domain concepts according to their influence on the final usability of the language.

This process will allow the language engineer to decide which usability tests are most pertinent in the current development stage and for a specific usage context. Controlling iteration priorities in turn enables a higher level of management over the usability process, by defining which usability aspects and features are to be tested iteration-wise.

Example: For Pheasant, considering only query writing tasks, the list of subtasks that the user is required to cope with and respective frequency is as follows (see Table 11):

Table 11 – Task frequency use table

Task	Frequency
Inform Status	3
Write Query	5
Generate Coder	4
Execute	4
Get Query Result	4
Define Shema	3

Writing query task consist of four subtasks: (i) Selecting Collections, (ii) Selecting Events, (iii) Selecting the Decay and (iv) Selecting the Result. These subtasks are capturing the domain concepts presented as the metamodel elements (see Table 12).

Table 12 – Query subtask connection with metamodel elements

METAMODEL: QPheasant			Querying subtask
Connectable	<--Selection		Selecting the Decay
	<--TransitionResult		
Transition			Selecting the Decay
Aggregation			Selecting the Decay
CollectionNode	<--CCOP	<--Union	Selecting Collections
		<--Intersection	
	<--CollectionSet		
	<--Exclusdion		
Event			Selecting Events
ResultNode	<--OneD		Selecting the Result
	<--TwoD		
	<--ThreeD		
	<--Histogram		

Comparisson			Selecting the Decay
Distance	<--AbsDistance		Selecting the Decay
	<--RelDistance		

After having this analysis, it makes it easier to connect the metamodel elements with usability requirements and produce concrete metrics in the terms of combination of subtasks that user need to perform.

Related Patterns:

> **Usability Requirements Definition:** In order to consider Domain Concept impact on the development, a clearly defined list of usability requirements is essential.
>
> **Domain Concept Usability Evaluation:** The impact of the domain concept on the quality of the end product influences evaluation priority and importance.

Known uses: Conceptual distance has its roots in cognitive psychology. The concept of modularity that is involved in model driven development allows us to measure this distance using cognitive maps [18]. Application of this approach is visible in terms of analysis of cognitive effectiveness [19], [20].

2.4.3. Domain Concept Usability Evaluation

Problem Description: How to capture domain concept related usability problems using metrics.

Context: There are many advantages of determining the required quality characteristics of a DSL before it is developed and used.

During the metamodel implementation phase, which is usually complex as the language engineer needs to model all the domain concepts into the metamodel, it is also the time when all domain concepts are fresher and can thus be analyzed from a top-down perspective.

However, the domain concepts defined in the language metamodel should not be considered final and can/should be analyzed at fixed stages during development in order to evaluate the ability of the metamodel to apprehend all needed domain concepts and to allow for inclusion of usability requirements.

Performing some measure of qualitative analysis of initial language metamodel, which contains the domain concepts mapping at their initially stages, is an important step in language engineering, since problems identified at earlier phases would not be propagated onto the following phases of development.

Forces:

> **Metamodel evaluation:** The level by which a metamodel is analyzed for usability issues has a direct relation to future failures in implementation.

Solution**:** Using metrics to analyze metamodel concept's representation allows the language engineer to reason on how different concept modeling will impact the quality in use of the DSL. Applying internal and external quality metrics we can reason about syntax dependences (*i.e.* metamodel's features) and their relation (*i.e.* meaning that they give).

Ideally the language engineer should be able to understand how changes and variations in the metamodel's design influence functionality, operability and overall usability of the language. With this knowledge he can measure and decide the importance of quality attributes to achieve the end goal and therefore which ones should be targeted and subsequently validated.

Not all metrics and measurements contribute to this end as they might not provide important feedback regarding quality improvement. The most significant metrics analyze direct DSL usage by DSL users and extract information from the gathered DSL corpus.

Clone Analysis: Like in GPLs, duplicated code is a very well-known code smell that indicates modularization problems[21]. In DSLs corpus, more than a need to modularize, the existence of several clones, consistently showing up with a given pattern, should trigger our attention.

Cluster Analysis: Identifying clusters of domain concepts in the language corpus allows the language engineer to evaluate if related concepts or concepts that are often used together represent a sub-language within the DSL, i.e. how the changes in the corpus are reflecting in the usability of the DSL. This is again a modularity issue, as clusters should be, as much as possible, modularly independent from other clusters, thus usability issues in one cluster should not influence other clusters.

Semantics-based Analysis: Performing language analysis on the metamodel might help identify variations of the same meaning.

Usage Analysis: Metamodel elements with a high level of use by the users require more thought and consideration according to usability than less used concepts.

Metamodel design Patterns: Specification of a metamodel is dependent on the designer's domain knowledge and language expertise. Thus, it is advisable to follow existing designs patterns for metamodels.

Careful consideration of these and other available heuristics of actual usage of the DSL will allow the language engineer to direct project resources to the most critical language features.

Example: Evaluating Pheasant is not a trivial task. Nonetheless, the physicist, who takes the role of the query modeler, is immediately aware of the changes in the instances of the meta-meta-model just by using the visual operators when modeling his query (see Fig. 5). This picture represents the direct mapping that exists from the user actions in the model to the metamodel of language.

For the first cycles, the influence of quality characteristics of the language corpora on the user should be determined from user tasks. From these, and after the first quality assessment of the metamodel, the language engineer identifies potential need for clones and clusters. For instance, consider that the user identifies the need for two ways to accomplish the same thing, i.e. two distinct processes leading to the same outcome. The language engineer needs to design this in the metamodel. In this case the metamodel element representing the action needs sub-elements representing the different variations of the same task. This need should then be validated by discussing the true impact of these clusters and clones on the language's usability. In later validations of quality in use these agreements should be tracked, so as to understand if the existing metamodel analysis premises are needed in the new version or if the scope changed.

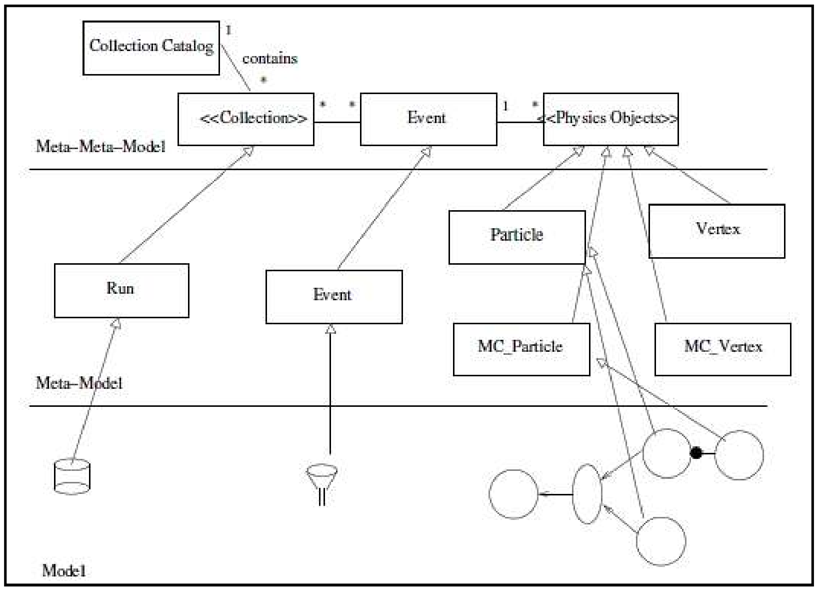

Figure 5 – Corpora relation to the metamodel tasks (Taken from [8])

Related Patterns:

> **Conceptual Distance Assessment:** The true impact of domains concepts in the quality in use of the DSL is measured by *Domain Concept Usability Evaluation*.
>
> **Usability Requirements Testing:** *Domain Concept Usability Evaluation* will also help reduce the budget for usability testing by directing tests to the most essential language features.

Known uses: Evaluation of concepts is performed using a conceptual dimensions framework [22]. This approach is also used in user interfaces evaluation by building a conceptual models [23].

2.4.4. Usability Requirements Testing

Problem Description: How to analyze if the goal usability requirements are being met by the DSL.

Context: Satisfying the user's needs should be the primary goal of a DSL. Therefore all DSLs have a strong consideration for quality in use, i.e. usability. It is important not only to define what are the principles by which the language is to be measured, i.e. which usability requirements and quality attributes define if a specific language is usable or not, but also what tests can be performed to ensure that the desired level of quality is achieved.

Forces:

> **Cost of Heuristic Validation:** Heuristic validation can be a very time consuming task. However, performing non-expensive heuristic validation, we can reveal lots of relevant information about achieved level of usability.
> **Cost of User Evaluation with small number of participants:** Validation of usability with a small number of users between release cycles can identify lots of usability failures.

Iterative Feedback: All feedback collected can be used to create mean values for the indicators of the next iteration cycles.

Solution: At the end of each iteration, a *usability requirements testing* stage is required to evaluate if the current implemented features go towards the usability goals previously defined [11, 24, 25].

When considering which tests to perform it is useful to consider the current state of the end product. There are usually three different levels of usability testing, depending on the current iteration:

- Initial developments or non-stable product versions should be tested by a very reduced set of users, and test should be strictly focused on the features under development. Feedback can be direct, e.g. through workshops and meetings, or through small questionnaires.
- Intermediate stable versions should be tested with a group of users that is representative of the user group most directly related to the provided stable features. It is important to test changes and variations between stable versions and also to test if previously validated features continue to achieve the intended goal. Feedback can be achieved through workshops and small questionnaires as in the previous stages but the questionnaires should be extended to include all features being tested. At this stage it is useful to observe and analyze user's usage processes to detect small scale usability problems related to automatic tasks and cognitive processes that usually are not reported.
- Release candidates are the most important focus of usability tests. The language engineer should ensure that the users are allowed to perform the tests with a minimum of interference and constrains. If a user cannot test due to a bug in the beginning of an activity, the entire test process is undermined.

Additionally the language engineer should define, with the assistance of key users, a set of heuristic based validation methodology that will allow validation of the DSL without direct user intervention. These can be for instance a measure of user clicks to achieve a certain use case, product performance and responsiveness, ability to rollback on user errors, content placement, etc.

There are a few guidelines that should be followed to successfully perform usability tests:

- Test usability with real DSL users
- Ideally use real usage test cases rather than dummy examples. For the final stages of development, a beta testing of a stable version of the DSL in real life usage environment should be considered.
- Tasks and features being tested should be directly related to the goals and concerns of the current iteration.
- All user feedback should be accounted for, even if no measure of importance can be given to the feedback, it might serve to provide feedback on the user's state of mind and motivations.
- If possible allow for discussion. Users usually have different views of a same subject and it is useful to allow them to debate these views in order to reach a common understanding.

One important fact about usability testing is that tests should be targeted at the domain under study. Some domains are more prone to accept some types of tests rather than others. It's up to the language engineer to detect these patterns and proceed accordingly.

Also, most users are not aware that test versions might have minor issues and bugs that where not detected (ergo the need for tests). When encountering a fatal bug, most users will immediately consider the implications of that bug if it were on a real case situation and the setbacks it might cause. This is a potentially fatal outcome for the tests as users will be cautious of accepting new versions for testing.

Example: Falling back to the Goal of the 5th iteration (Table 7), i.e. knowing how easy the language is to learn and use, Usability tests are constructed following the next table.

Table 13 – Usability testing

Usability measures		Test types	Treatmant
Effectiveness	- error rates while user completes querying sentences	Immediate comprehension	Learning
Efficiency	- time spent to complete a query	Reviews	Learning, Testing
Satisfaction	- confidence feedback about query	Final exams	Testing

The testing instruments were developed as Evaluation Queries and Feedback questionnaires.

Evaluation Queries are given in four levels of complexity. Queries are given in natural language English to be rewritten in the previously learned language (i.e. Pheasant). For each of the queries, time taken to reply them is taken. In the Pheasant project, queries were evaluated according to an error rate scale (0-5) and correctness was measured according to a self-assessment by the subject of his reply, essentially rating his feeling of the correctness of the answer. The rates were: totally correct (TC), almost correct (AC), totally incorrect (TI), not attempted (NA).

After each session, the participants were asked to judge the intuitiveness, suitability and effectiveness of the query language. After the tests are completed, the participants were asked to compare specific aspects of query languages. They rated which query language they preferred and to what extent. After the evaluation session the participants were asked to write down informal comments and suggestions for improving the language.

Example of result analysis of confidence with using the language constructs is given in

Table 14 – Language constructs analysis

Pheasant / BEE	Non-P	P	Mean
Structuring the query	5/1	4/4	4.5/2.5
Different data schema feature	3.5/1	3.5/3	3.5/2
Expressing filter conditions	5/1	4.5/2	4.75/1.5
Expressing and using vertexing	5/1	5/4	5/2.5
Expressing the result set	5/1	5/3.5	5/2.25
Expressing a decay	5/1	4.5/2	4.75/1.5
Path expressions	5/3.5	3/5	4/4.25
Expressing and using UDFs	4.5/1	3.5/5	4/3
	4.8/1.3	4.2/3.9	

Related Patterns:

> **Iteration Validation:** the tests performed in *Usability Requirements Testing* are used to supply feedback to each *Iteration Validation*.
>
> **Usability Requirements Definition:** Feedback data collected can help define next iteration usability requirements.
>
> **Domain Concept Usability Evaluation:** The users' feedback provides a good starting point to define which domain concepts are correctly mapped and which pose problems.
>
> **Experimental Language Evaluation Design:** *Usability Requirements Testing* is complementary to *Experimental Language Evaluation Design* as the goals and test methodology differs.

Known uses: This approach originates from usability engineering [2]. Its application can be seen in existing usability evaluation examples [9], [26], [27].

2.4.5. Experimental Language Evaluation Design

Problem Description: How to design and control the process of empirical experimentation to achieve language evaluation.

Context: Using Iterative User-Centered DSL Development Process, not all iterations require full-fledged evaluation in order for the requirements to be considered successfully achieved. However, presenting to the DSL user a final version of the language without it being thoroughly and extensively tested by DSL users in a real-life use case is not an ideal solution. Nonetheless option is used many times due to the complexities of performing experimental evaluation with DSL users.

Forces:

> **Experimentation definition:** The definition of the experimentation expresses something about why a particular language evaluation was performed and may help justify the budget assigned to this type of validation [28].
>
> **User Expectations:** The expectations of users need to be managed and evened out prior to the experiment, otherwise there is a high chance of impact in the end result: an extremely good result, if expectations are low or a poor result in case of high expectations.
>
> **User Distribution:** Ensuring that experimental evaluation is performed with an equatitative distribution of users representative of the most influential groups will reduce selection bias and ensure the end results will be representative of the goal real life usage.
>
> **Hypothesis Guessing:** The language engineer and development team through experience usually have a pre-conceived idea of the hypothesis result. This can influence the behavior of the experiment's participant.

***Solution*:** When a release candidate version of the DSL for a specific target user group seems to be ready for deployment, an experimental usability validation should be performed with real users and real test case scenarios.

Experiment planning expresses something about how it will be performed. Before starting the experiment, some considerations and decisions have to be made concerning the context of the experiment. The language engineer needs to define:

- Problem statement
- The hypotheses under study, i.e. what composes the claim that the DSL is in accordance with the users' definition of quality in use; The hypothesis usually can be supported or refuted, but not proven;
- The set of independent and dependent variables that will be used to evaluate the hypotheses. These have to be correctly chosen in order to provide results with any measure of statistical validity;
- What are the user groups represented in the experiment and how and which users are to be select;
- The experiment's design
- Instrumentation design, i.e. the artifacts used in the experience (e.g. questionnaires).
- The means to evaluate the experiment's validity.

Only after all these details are sorted out should the experiment be performed. The outcome of planning is the *experimental language evaluation design*, which should encompass enough details in order to be replicable by and independent source, case the need arises.

Experimental evaluation is based on quantitative evaluation of measurable properties collected from real scenarios. In this case, the aim of the experiment is to support or refute the hypothesis that the end result DSL has a direct and positive impact on usability and user performance.

Example: Following with the example of Pheasant and experimental evaluation models [29], we define the problem statement as a confluence of the academic context in which Pheasant is to be used. Therefore usability objectives and the experiments to measure these objectives have to take into account this context, i.e. academic level of the users, purpose, objectives and goals. This will help model a problem statement that encompasses all contextual aspects (Figure 2).
The context of an experiment determines our ability to generalize from the experimental results to a wider context. However, regardless of the specific context of the experiment, there are a number of context parameters that remain stable and their value is the same for all the subjects in the experiment.

Thus, having an instrument design model definition makes the task of analyzing the feedback received for target features across different iterations and users a much easier task. Modeling instruments is also useful to measure the independent tasks that directly impact usability. Experimenters in human factors have developed a list of tasks to capture particular usability aspects (*Sentence writing; Sentence reading, Sentence interpretation, Comprehension, Memorization and Problem solving*).

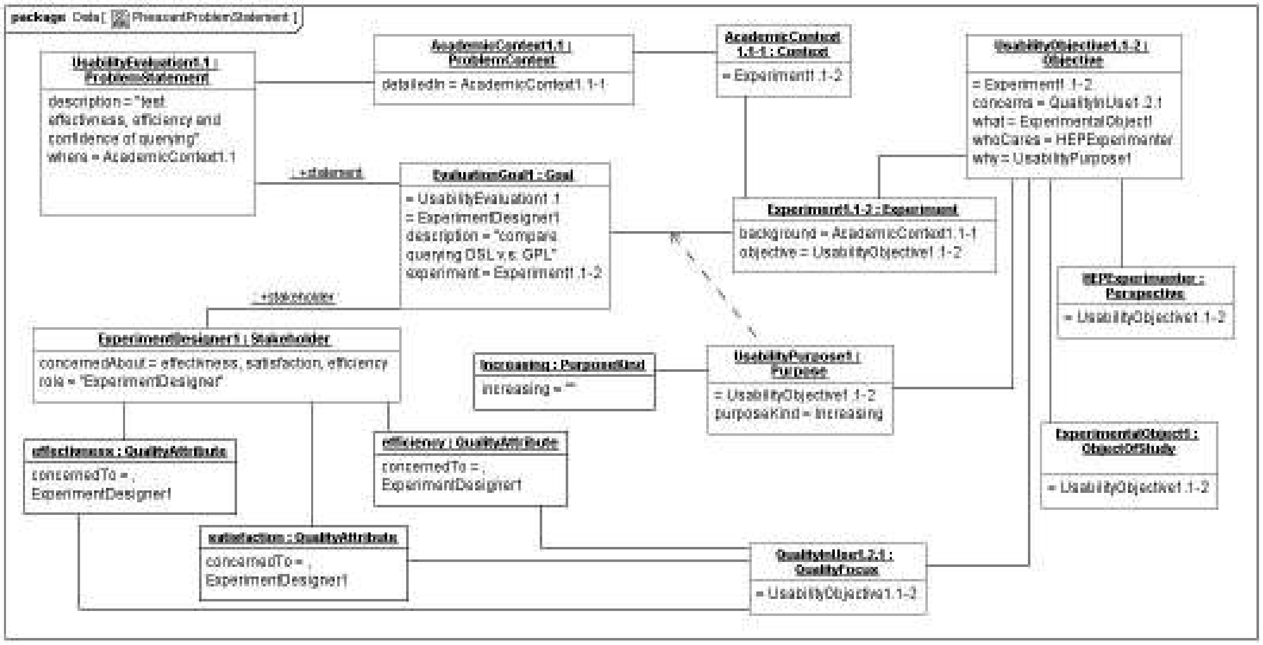
package Data[PheasantProblemStatement]
UsabilityEvaluation1.1 : ProblemStatement
description = "test effectiveness, efficiency and confidence of querying"
where = AcademicContext1.1
AcademicContext1.1 : ProblemContext
detailedIn = AcademicContext1.1-1
AcademicContext 1.1-1 : Context
= Experiment1.1-2
UsabilityObjective1.1-2 : Objective
= Experiment1.1-2
concerns = QualityInUse1.2.1
what = ExperimentalObject1
whoCares = HEPExperimenter
why = UsabilityPurpose1
: +statement
EvaluationGoal1 : Goal
= UsabilityEvaluation1.1
= ExperimentDesigner1
description = "compare querying DSL v.s. GPL"
experiment = Experiment1.1-2
Experiment1.1-2 : Experiment
background = AcademicContext1.1-1
objective = UsabilityObjective1.1-2
: +stakeholder
HEPExperimenter : Perspective
= UsabilityObjective1.1-2
ExperimentDesigner1 : Stakeholder
concernedAbout = effectiveness, satisfaction, efficiency
role = "ExperimentDesigner"
Increasing : PurposeKind
increasing = ""
UsabilityPurpose1 : Purpose
= UsabilityObjective1.1-2
purposeKind = Increasing
ExperimentalObject1 : ObjectOfStudy
= UsabilityObjective1.1-2
effectiveness : QualityAttribute
concernedTo = , ExperimentDesigner1
efficiency : QualityAttribute
concernedTo = , ExperimentDesigner1
satisfaction : QualityAttribute
concernedTo = , ExperimentDesigner1
QualityInUse1.2.1 : QualityFocus
= UsabilityObjective1.1-2

Figure 6 –Pheasant experimental Problem Statement model

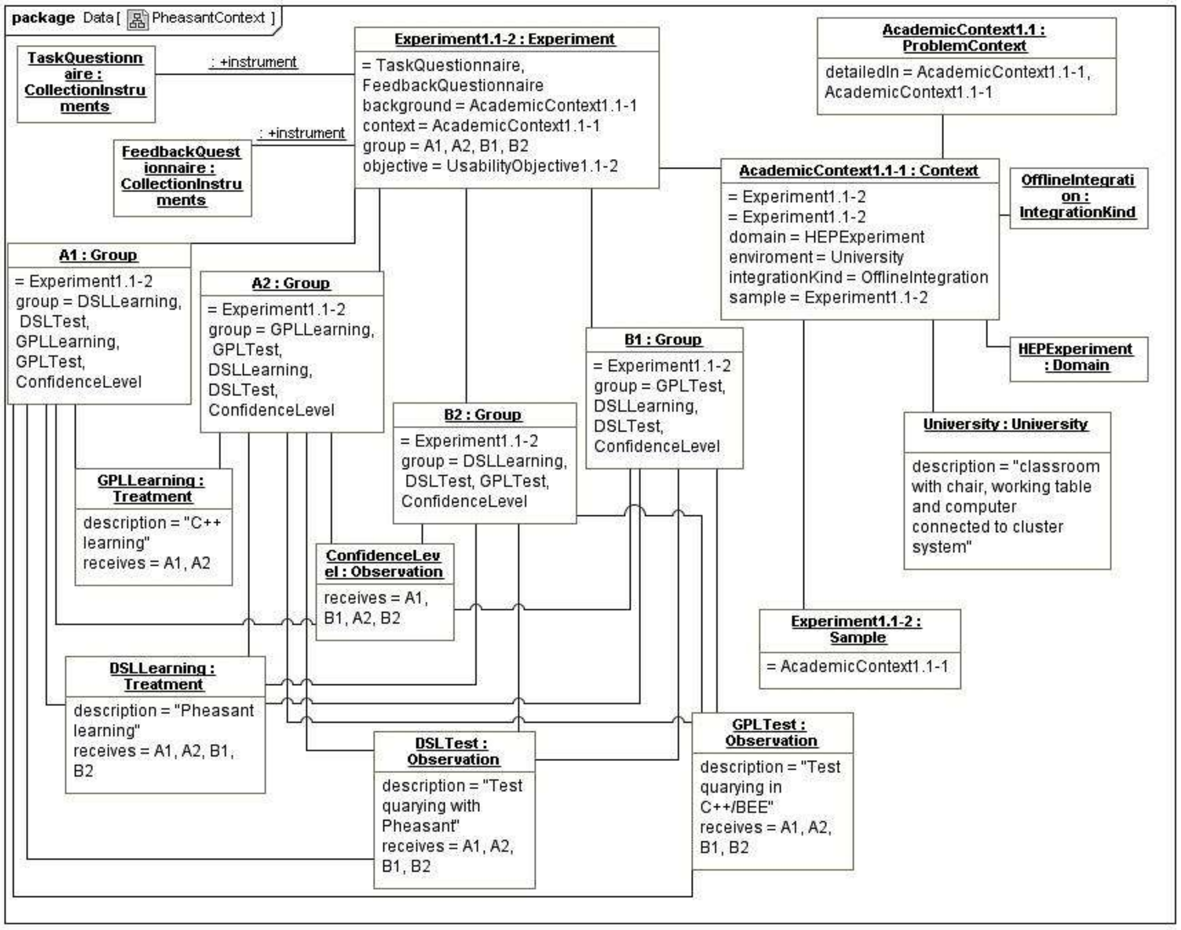
package Data[PheasantContext]
TaskQuestionnaire : CollectionInstruments
: +instrument
FeedbackQuestionnaire : CollectionInstruments
: +instrument
Experiment1.1-2 : Experiment
= TaskQuestionnaire, FeedbackQuestionnaire
background = AcademicContext1.1-1
context = AcademicContext1.1-1
group = A1, A2, B1, B2
objective = UsabilityObjective1.1-2
AcademicContext1.1 : ProblemContext
detailedIn = AcademicContext1.1-1, AcademicContext1.1-1
AcademicContext1.1-1 : Context
= Experiment1.1-2
= Experiment1.1-2
domain = HEPExperiment
enviroment = University
integrationKind = OfflineIntegration
sample = Experiment1.1-2
OfflineIntegration : IntegrationKind
HEPExperiment : Domain
University : University
description = "classroom with chair, working table and computer connected to cluster system"
Experiment1.1-2 : Sample
= AcademicContext1.1-1
A1 : Group
= Experiment1.1-2
group = DSLLearning, DSLTest, GPLLearning, GPLTest, ConfidenceLevel
A2 : Group
= Experiment1.1-2
group = GPLLearning, GPLTest, DSLLearning, DSLTest, ConfidenceLevel
B1 : Group
= Experiment1.1-2
group = GPLTest, DSLLearning, DSLTest, ConfidenceLevel
B2 : Group
= Experiment1.1-2
group = DSLLearning, DSLTest, GPLTest, ConfidenceLevel
GPLLearning : Treatment
description = "C++ learning"
receives = A1, A2
ConfidenceLevel : Observation
receives = A1, B1, A2, B2
DSLLearning : Treatment
description = "Pheasant learning"
receives = A1, A2, B1, B2
DSLTest : Observation
description = "Test quarying with Pheasant"
receives = A1, A2, B1, B2
GPLTest : Observation
description = "Test quarying in C++/BEE"
receives = A1, A2, B1, B2

Figure 7 – Pheasant experimental Context model

For Pheasant, the language engineer defined two types of instruments for the experimentation: Task Questionnaires, designed to capture Sentence Writing, Memorization and Problem Solving, and Feedback Questionnaires, which are used to get better insight in users satisfaction, and additional recommendations.

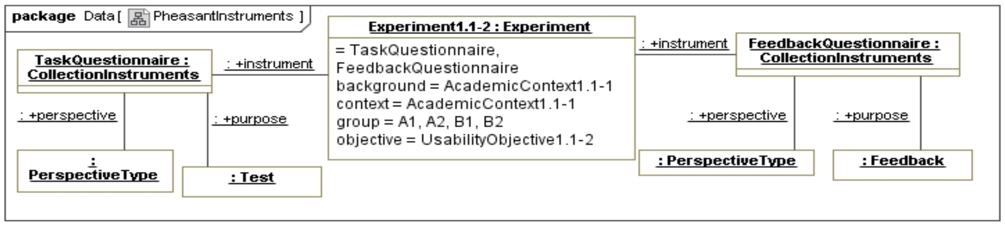
package Data [PheasantInstruments]
Experiment1.1-2 : Experiment
= TaskQuestionnaire, FeedbackQuestionnaire
background = AcademicContext1.1-1
context = AcademicContext1.1-1
group = A1, A2, B1, B2
objective = UsabilityObjective1.1-2
TaskQuestionnaire : CollectionInstruments
: +instrument
: +perspective
: +purpose
: PerspectiveType
: Test
FeedbackQuestionnaire : CollectionInstruments
: +instrument
: +perspective
: +purpose
: PerspectiveType
: Feedback

Figure 8 – Pheasant experimental Instrumental design model

The language engineer should clearly define the profile of the participants and the artifacts that are involved in the experiment.

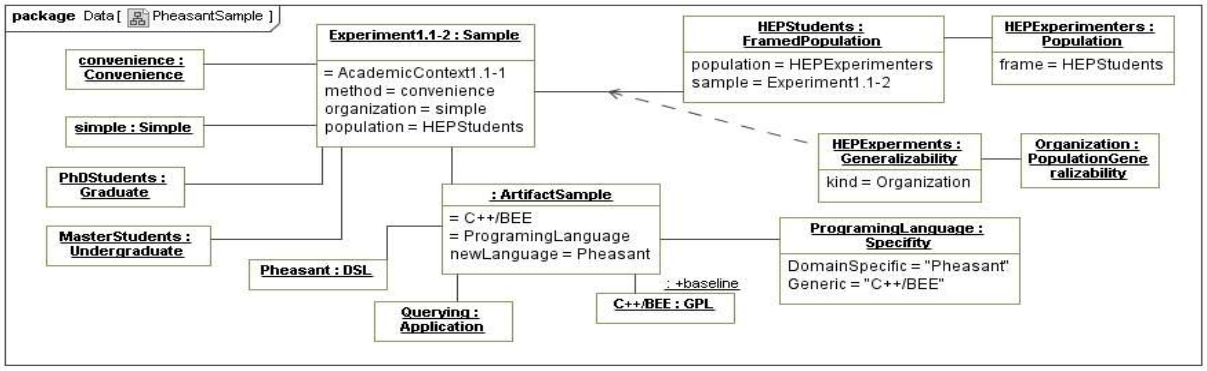

Figure 9 – Pheasant experimental Sample design model

Quality focus needs to be defined through criteria, which can be recursively decomposed into sub criteria. For each criterion we should specify different recommendations, i.e. positive assessments that characterize criteria. We should specify a weight for each recommendation to define which of them are more important than others for the subjects involved in the experimental evaluation.

Evaluations of each quality criteria should be performed through methods that are specified by metrics and/or practices. Metrics gives us numerical results that can be comprised between some limits when defined, while practice can be either a pattern or an anti-pattern, applied at the process level, or on a language. Both are directly evaluated on the experiment subjects' trough recommendations [30].

When a result of the evaluation does not satisfy the expected level of quality in use, the designer will need to increase the quality by setting a transformations or set of transformations. These transformations are related to language artifacts on which the evaluation was performed. Iterations can be done in same experimental settings until the desired quality is reached.

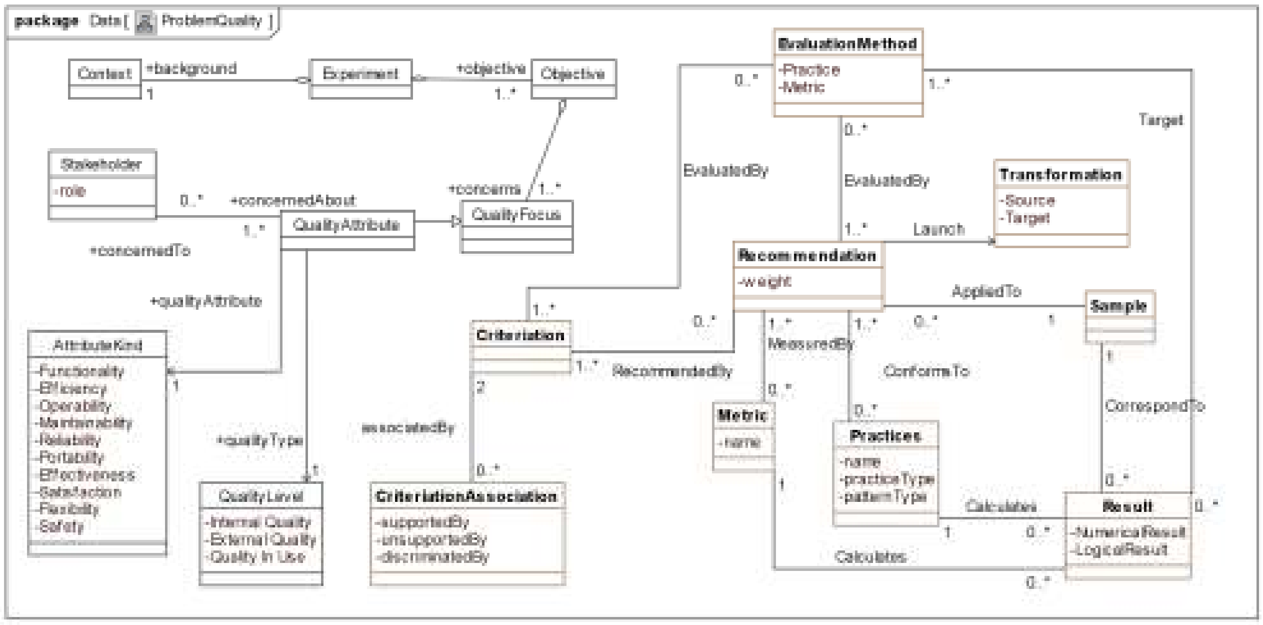

Figure 10 – Pheasant experimental Quality Design model

The analysis techniques chosen for the language evaluation experiment depend on the adopted language evaluation design, the variables defined earlier, and the research hypotheses being tested. More than one technique may be assigned to each of the research hypotheses, if necessary, so that the analysis results can be cross-checked later. Furthermore, each of the hypotheses may be analyzed with a different technique. This may be required if the set of variables involved in that hypothesis differs from the set being used in the other hypotheses under tested.

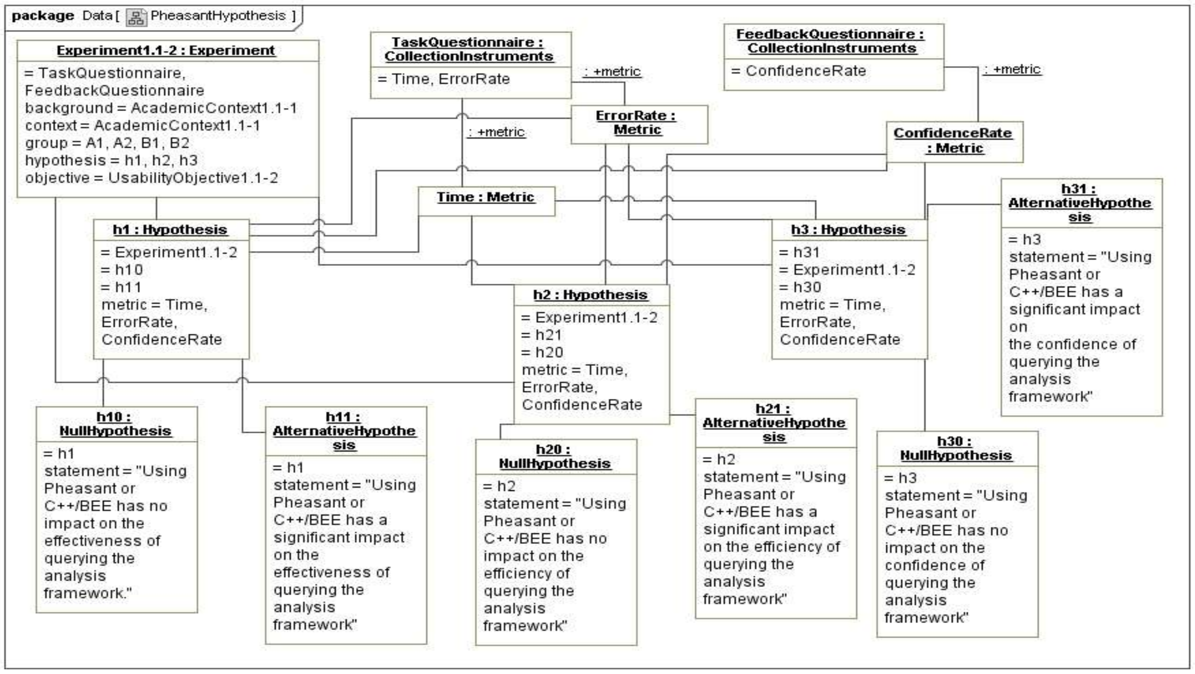
package Data[PheasantHypothesis]
Experiment1.1-2 : Experiment
= TaskQuestionnaire, FeedbackQuestionnaire
background = AcademicContext1.1-1
context = AcademicContext1.1-1
group = A1, A2, B1, B2
hypothesis = h1, h2, h3
objective = UsabilityObjective1.1-2
TaskQuestionnaire : CollectionInstruments
= Time, ErrorRate
: +metric
ErrorRate : Metric
: +metric
FeedbackQuestionnaire : CollectionInstruments
= ConfidenceRate
: +metric
ConfidenceRate : Metric
Time : Metric
h1 : Hypothesis
= Experiment1.1-2
= h10
= h11
metric = Time, ErrorRate, ConfidenceRate
h2 : Hypothesis
= Experiment1.1-2
= h21
= h20
metric = Time, ErrorRate, ConfidenceRate
h3 : Hypothesis
= h31
= Experiment1.1-2
= h30
metric = Time, ErrorRate, ConfidenceRate
h31 : AlternativeHypothesis
= h3
statement = "Using Pheasant or C++/BEE has a significant impact on the confidence of querying the analysis framework"
h10 : NullHypothesis
= h1
statement = "Using Pheasant or C++/BEE has no impact on the effectiveness of querying the analysis framework."
h11 : AlternativeHypothesis
= h1
statement = "Using Pheasant or C++/BEE has a significant impact on the effectiveness of querying the analysis framework"
h20 : NullHypothesis
= h2
statement = "Using Pheasant or C++/BEE has no impact on the efficiency of querying the analysis framework"
h21 : AlternativeHypothesis
= h2
statement = "Using Pheasant or C++/BEE has a significant impact on the efficiency of querying the analysis framework"
h30 : NullHypothesis
= h3
statement = "Using Pheasant or C++/BEE has no impact on the confidence of querying the analysis framework"

Figure 11 - Pheasant experimental Hypothesis and Variable design model

Related Patterns:

> **Usability Requirements Definition:** The requirements defined will be validated at this stage. Also, if the development cycle is not yet complete, the feedback from *Experimental Language Evaluation Design* is fed back into *Usability Requirements Definition* to redefine the goals of the next iteration evaluation.
>
> **Usability Requirements Testing:** *Experimental Language Evaluation Design* is complementary to *Usability Requirements Testing* as the goals and test methodology differs.

Known uses: Detailed evaluation design is used in both usability engineering and experimental software engineering. This approach is modeled from the language comparison from [31] and discussed in [29].

3. RELATED WORK

There is a related line of work on Human Computer Interaction patterns, branching areas like ubiquitous systems [32], web design[33], safety-critical interactive systems [34], as well as more general interaction design languages [35-38]. Although HCI has a large focus on usability, the patterns available mainly avoid process patterns and prefer patterns that represent actual usable human interaction artifacts[39], like News Box, Shopping Cart or Breadcrumbs.

Spinellis [40] presents a pattern language for the design and implementation of Domain Specific Languages. Contrary to ours, these patterns refer to concrete implementation strategies and not to the process of building the DSL or usability concerns. Günther [41] presents a pattern language for Internal DSLs. These patterns mainly focus on how to map domain concepts to language artifacts and follow by implementing said artifacts with a general purpose language capable of supporting internal languages. In [42] Jones, Dunnavant and Jay present a pattern language that handles the language part of DSL design. It presents patterns such as Key-value Pairs, Semantic Evaluator or Language Composition that describe language implementation features.

Much of our patterns are based upon Völter and Bettin's pattern language for model driven software development(MDD) [13]. These patterns represent a well-rounded view of MDD but they do not explicitly account for the importance of Usability in DSLs and therefore do not give explicit instructions on how to test and validate usability of the end product. It is our opinion that our pattern language can be composed with Völter and Bettin's to produce a more complete version of a pattern language for MDD with usability concerns. To the best of our knowledge, ours is the only pattern language focusing on Domain Specific Language development process with user centered design.

As for usability, there are not many patterns or pattern languages available to cover usability concerns. Folmer and Bosch[43] developed a usability framework based on usability patterns to investigate the relationship between usability and software architecture. This work however has little relation to usability tests and to the development of usable software through usability validation. Thy instead map well know HCI patterns, such as Wizard, Multi-tasking and Model-View-Controller to quality attributes and usability properties. However, this is somewhat related to our *Conceptual Distance Assessment* pattern and the framework could in theory be used to identify the mappings between domain concepts and quality attributes. Ferre et al's software architectural view of usability patterns [44] follows a similar approach. Graham's pattern language for web usability [45] deals with usability evaluation and usability testing process. However, we feel that his patterns are hard to follow due to the number of patterns and lack of formal structure. Furthermore, Graham's patterns are targeted at web-based software. The pattern language most similar to ours is Gellner and Forbig's Usability Evaluation Pattern Language[46]. This pattern language is composed of thirty five patterns for usability testing. Of those, the Eight Phase pattern represents a set of eight stages of the process of usability evaluation. This is a similar approach to ours and has the merit of summarizing the process into a single pattern. However, the goal of the pattern is to disseminate usability evaluation for small scale projects while our pattern language considers small to large projects.

4. CONCLUSION

The software development industry is only now starting to invest effort in providing efficient development strategies that consider usability. For the world of Domain Specific Language Engineering, is a very important feature.

This paper describes a pattern language for evaluating the usability of domain specific languages. The 17 patterns described here represent a collection of usability-oriented best practices, collected from a wide set of domains, from GPL design to human computer interaction. Very little work has been done in ensuring these best practices become standard practices in the DSL world. This work intends to provide a platform to disseminate this knowledge and help bridge the gap.

In the future we intend to refine this pattern language and continue to expand it. DSL development is a new and exciting field and there is no doubt that many more patterns wait to be found.

5. ACKNOWLEDGEMENTS

We would like to acknowledge CITI - PEst -OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) - for the financial support for this work.

Besides, we would like to thank Uira Kulesza, whose comments have been incredibly helpful and insightful. His shepherding helped us significantly improve the quality of this paper. We would like to thank also to anonymous reviewers and to our program committee member Uwe Zdun that was following the shepherding process.

REFERENCES

[1] A. Barišić, *et al.*, "How to reach a usable DSL? Moving toward a Systematic Evaluation," *Electronic Communications of the EASST (MPM),* 2011.

[2] J. Rubin and D. Chisnell, *Handbook of usability testing: how to plan, design and conduct effective tests*: Wiley-India, 2008.

[3] F. Buschmann, *et al.*, "A system of patterns: Pattern-oriented software architecture," ed: Wiley New York, 1996.

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

[5] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*: Wiley-IEEE Computer Society Press, 2008.

[6] ISO, "ISO/IEC 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability," ed, 2001.

[7] C. Jones, *Applied software measurement: assuring productivity and quality*: McGraw-Hill, Inc., 1991.

[8] V. Amaral, "Increasing productivity in High Energy Physics data mining with a Domain Specific Visual Query Language," in *Phd. Thesis, University of Mannheim*, ed, 2005.

[9] A. Barišić*, et al.*, "Quality in Use of Domain Specific Languages: a Case Study," presented at the Evaluation and Usability of Programming Languages and Tools (PLATEAU) Portland, USA, 2011.

[10] P. Reisner, "Human factors studies of database query languages: A survey and assessment," *ACM Computing Surveys (CSUR),* vol. 13, pp. 13-31, 1981.

[11] D. J. Mayhew, "The usability engineering lifecycle: a practitioner's handbook for user interface design," *The Usability Engineering Lifecycle A Practitioners Handbook for User Interface Design,* 1999.

[12] T. Catarci, "What happened when database researchers met usability* 1," *Information Systems,* vol. 25, pp. 177-212, 2000.

[13] M. Völter and J. Bettin, "Patterns for Model-Driven Software-Development," presented at the EuroPLoP'04, Irsee, Germany, 2004.

[14] C. Jones, "Software change management," *Computer,* vol. 29, pp. 80-82, 1996.

[15] A. Öztaş and Ö. Ökmen, "Risk analysis in fixed-price design–build construction projects," *Building and Environment,* vol. 39, pp. 229-237, 2004.

[16] P. Carlshamre, "A usability perspective on requirements engineering: from methodology to product development," Linköping, 2001.

[17] M. Alférez*, et al.*, "A Model-Driven Approach for Software Product Lines Requirements Engineering," presented at the 20th International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2008.

[18] M. P. Monteiro, "On the Cognitive Foundations of Modularity."

[19] D. Moody and J. van Hillegersberg, "Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams," *Software Language Engineering,* pp. 16-34, 2009.

[20] P. H. D. A. N. Genon, "Analysing the Cognitive Effectiveness of the BPMN Visual Notation," *Software Language Engineering,* 2010.

[21] K. Beck*, et al.*, "Bad smells in code," *Refactoring: Improving the design of existing code,* pp. 75-88, 1999.

[22] T. Kosar*, et al.*, "Comparing General-Purpose and Domain-Specific Languages: An Empirical Study," *Computer Science and Information Systems,* vol. 7, pp. 247-264, 2010.

[23] J. Johnson and A. Henderson, "Conceptual models: begin by designing what to design," *interactions,* vol. 9, pp. 25-32, 2002.

[24] J. S. Dumas and J. Redish, *A practical guide to usability testing*: Intellect Ltd, 1999.

[25] J. Nielsen, *Usability engineering*: Morgan Kaufmann, 1994.

[26] N. S. Murray*, et al.*, "Kaleidoquery--a flow-based visual language and its evaluation," *Journal of Visual Languages &amp; Computing,* vol. 11, pp. 151-189, 2000.

[27] T. Conte*, et al.*, "Usability evaluation based on Web design perspectives," 2007, pp. 146-155.

[28] V. R. Basili, "The role of experimentation in software engineering: past, current, and future," presented at the 18th International Conference on Software Engineering (ICSE 1996), 1996.

[29] A. Barišić*, et al.*, "Evaluating the Usability of Domain-Specific Languages," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, M. Mernik, Ed., ed: IGI Global, 2012.

[30] A. García Frey*, et al.*, "QUIMERA: a quality metamodel to improve design rationale," 2011, pp. 265-270.

[31] M. Goulão and F. B. Abreu, "Modeling the Experimental Software Engineering Process," in *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, 2007.

[32] J. Roth, "Patterns of mobile interaction," *Personal and Ubiquitous Computing,* vol. 6, pp. 282-289, 2002.

[33] D. K. Van Duyne*, et al.*, *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience*: Addison-Wesley Professional, 2003.

[34] A. Hussey, "Patterns for safer human-computer interfaces," *Computer Safety, Reliability and Security,* pp. 686-686, 1999.

[35] J. Tidwell, *Designing interfaces*: O'Reilly Media, Inc., 2005.

[36] V. A. Schmidt, "User Interface Design Patterns," ed: AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH HUMAN EFFECTIVENESS DIRECTORATE, 2010.

[37] M. Van Welie and G. C. Van der Veer, "Pattern languages in interaction design: Structure and organization," in *Interact'03*, Amsterdam, Netherlands, 2003, pp. 527-534.

[38] T. OReilly. (2005, What is Web 2.0: Design patterns and business models for the next generation of software. Available: oreilly.com

[39] M. Mahemoff and L. J. Johnston, "Usability Pattern Languages: the" Language" Aspect," 2001, p. 350.

[40] D. Spinellis, "Notable design patterns for domain-specific languages," *Journal of Systems and Software,* vol. 56, pp. 91-99, 2001.

[41] S. Günther, "Development of Internal Domain-Specific Languages: Design Principles and Design Patterns," presented at the PLoP 2011, Portland, OR, USA, 2011.

[42] C. Schäfer*, et al.*, "A Pattern-based Approach to DSL Development," in *compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, Portland, Oregon, USA, 2011, pp. 39-46.

[43] E. Folmer and J. Bosch, "Usability patterns in software architecture," 2003.

[44] X. Ferre*, et al.*, "A software architectural view of usability patterns," 2003.

[45] I. Graham, *A pattern language for web usability*: Addison-Wesley Longman Publishing Co., Inc., 2002.

[46] M. G. P. Forbrig, "A Usability Evaluation Pattern Language."