Patterns for Evaluating Usability of Domain-Specific Languages

ANKICA BARIŠIĆ, Universidade Nova de Lisboa
PEDRO MONTEIRO, Universidade Nova de Lisboa
VASCO AMARAL, Universidade Nova de Lisboa
MIGUEL GOULÃO, Universidade Nova de Lisboa
MIGUEL MONTEIRO, Universidade Nova de Lisboa

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. The inclusion of the Domain Experts directly in the development cycle is a very important characteristic of Domain-Specific Languages, 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. These patterns can help show how to use an iterative usability validation development strategy to produce DSLs that can achieve a high degree of Usability.

Categories and Subject Descriptors: **H.5.2 [Information Interfaces and Presentation]**: User Interfaces—*Evaluation/methodology*; **H.1.2 [Models and Principles]**: User/Machine Systems—*Human Information Processing*; **D.3.3 [PROGRAMMING LANGUAGES]:** Language Constructs and Features—*Patterns, Specialized application languages;* **D.2.9 [SOFTWARE ENGINEERING]**: Management - Software quality assurance (SQA), Productivity, **D.2.13 [SOFTWARE ENGINEERING]**: Reusable Software - Domain engineering;

General Terms: Usability Evaluation of Domain-Specific Language

Additional Key Words and Phrases: Pattern Language, Domain-Specific Language, Usability Evaluation

ACM Reference Format:

Barišić, A., Monteiro, P., Amaral, V., Goulão, M. and Monteiro, M. P. 2012. Patterns for Evaluating Usability of Domain-Specific Languages. In Proceedings of the 19th ACM Conference on Pattern Languages of Programs (PLoP 2012) (SPLASH, Tucson, Arizona, USA, October 19-21 2012).

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)[Barišić et al. 2012]. In order to accomplish and validate the desired outcome, 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.

Author's address: Ankica Bariši, CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal; email: a.barisic@campus.fct.unl.pt; Pedro Monteiro, CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal; email: pmfcm@campus.fct.unl.pt; Vasco Amaral, CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal; email: vma@fct.unl.pt; Miguel Goulão, CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal; email: mgoul@fct.unl.pt; Miguel P. Monteiro, CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal; email: mtpm@fct.unl.pt

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 [Gabriel et al. 2010]. 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 [Barišić et al. 2011]. 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. Applying these methods to DSL end users enables us to promote DSL usability as a priority in the DSL development. It follows that usability must then be considered from the beginning of the development cycle.

One way of doing this is through user-centered methods [Rubin and Chisnell 2008], i.e. placing the intended end users of a language as the focal aspect 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 iteration stages between the DSL development process and the usability evaluation process [Barišić, Amaral, Goulão and Barroca 2011]. Following an agile development approach focused on usability will allow us to track usability requirements and the impact of recommendations with a well-prepared evaluation process, by that allowing management to control budget and scope of language evaluation.

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 [Buschmann et al. 1996]. Thus, using patterns, we aim to disseminate the knowledge of these best practices to both expert and non-expert developers, easing the adoption of good 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. PATTERNS

A pattern language is a set of inter-dependent patterns that provide a complete solution to a complex problem [Buschmann, Meunier, Rohnert, Sommerlad and Stal 1996]. The main purpose of these patterns is to identify which commonly and successfully-used techniques for usability evaluation can be effectively applied in DSL design and guide the reader on the process of applying said techniques. As such, apart from the provided examples, the main core of Known Uses and examples we provide exist outside the realm of DSLs. This ensures the reader has a wide range of documented examples and case studies to choose from.

Our inter-dependent set of patterns is divided into the following three design spaces (see Figure 1):

Agile Development Process. This design space considers patterns devoted to project management and engineering of a DSL. This is the most important design space because it is through it that the development team (i.e. Language Engineers, Usability Engineers and Domain Experts) accesses the remaining design spaces.

- USER AND CONTEXT MODEL EXTRACTION. Before building a new DSL we should identify all intended user profiles and target context of use.
- EVALUATION PROCESS AND DESIGN PLANNING. Usability evaluations and experimental designs should be carefully planned through an experimental process model.
- ITERATIVE USER-CENTERED DSL DESIGN. Introducing DSLs User-Centered methods allows us to achieve a productivity increase.
- ITERATION VALIDATION. By validating the iterations in time-box fixed intervals we can monitor progress and check if development is going in the desirable direction.

- CONTEXT SCOPE TRADING. Short iterations require short and well scoped contexts.
- FIXED BUDGET USABILITY EVALUATION. In order to reduce the cost of Usability validation and increase the validity of design decisions, the development team should plan development budgets according to the scope of iteration.

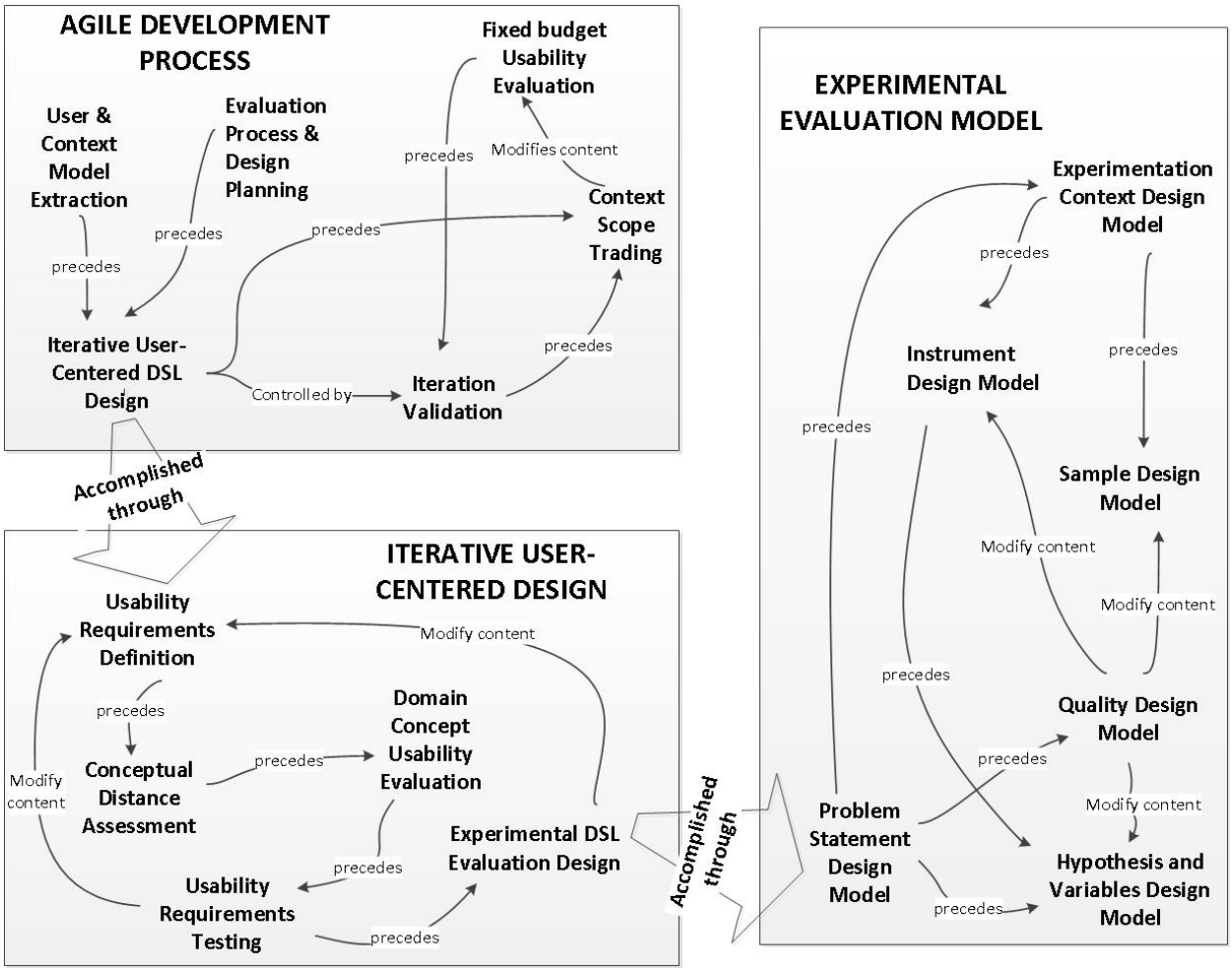

Figure 1. Patterns for Evaluating Usability of Domain-Specific Languages

Iterative User-Centered Design. The users are the central part of a DSL. This design space considers how to engage the users (both Domain Experts and End Users) in the development process. Also, it describes the roles and responsibilities of the Usability and Language Engineers in collecting valuable information about the DSL and its level of usability while it is being developed.

– USABILITY REQUIREMENTS DEFINITION. While building domain concepts, through direct interaction with Domain Experts, it is valuable to collect background information of the target users of each language concepts, in order to specify what usability means to them.
– CONCEPTUAL DISTANCE ASSESSMENT. In order to understand how the design of the language's architecture impacts usability requirements, it is necessary to elect quality indicators and relate them to domain concepts
– DOMAIN CONCEPT USABILITY EVALUATION. Using metrics to analyze the metamodel's concepts representation allows the Language Engineer to reason on how different concept models impact the DSL's quality in use.

– USABILITY REQUIREMENTS TESTING. It is necessary to provide tests and evaluate if the current implemented features contribute to the defined goals.
– Experimental DSL Evaluation Design. 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.

EXPERIMENTAL EVALUATION MODEL. This design space indicates adoption of experimental software engineering practices to the Usability evaluation of DSLs. As these practices are generally known we are not describing them in scope of this paper. However, example of its application and systematic comparison of best DSLs evaluation experiments is given in .

2.1. Key concepts (Lexicon)

Language Engineer is a professional who is skilled in the application of the engineering discipline to the creation of software languages. They manage implementation priorities, design the software language and are responsible for making it functional at the system level. Language Engineers are involved in the language specification, implementation, and evaluation, as well as providing templates and scripts [Kleppe 2009].

End User is someone who uses software languages to create applications [Kleppe 2009] (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 [Kelly and Tolvanen 2008].

Domain Expert is a person involved in the language development process, also known as a knowledge engineer or consultant. 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. They are responsible for managing system goals and iterations. In contrast with End Users, they should have domain knowledge that includes areas of all target model applications.

Usability Engineer is a professional that is skilled in assessing and making recommendations that will improve Usability. Usability Engineers may be engaged into design of language in order to reason about concrete Usability metrics and design change impact. They are responsible for user research and evaluation management of product.

Domain-Specific Languages are programming languages that provide solutions to essential problems from a given domain (e.g. Physics Computing, Financial Domain, Healthcare, Control Systems). They are often used by Domain Experts, rather than programmers with a background in computer science. DSLs Usability has a deep impact on developers' Productivity. As such, DSLs should be evaluated as human-computer languages (i.e. User Interfaces) with respect to their Usability, so that they can be improved and are more efficient in bridging the gap between the Problem and the Solution domains. Some examples of well-known and successful DSLs include SQL, PostScript, LabView, Simulink, Lego Mindstorms, TexLanguage (like LaTex and BibTex) and Microsoft Excel.

Usability is the quality characteristic that measures the ease of use of any software system that interacts directly with an End 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"*[Iso 2001]. 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 the produced good to the effort and cost of producing it. It is considered that good software systems analysis enhances software Productivity and software Productivity is a success measure of systems analysis. The measure of Productivity is based on the achieved results of the software in use. A 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 End User. These 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 [Jones 1991].

User-centered method (design) is comprised of End User involvement in development of software product at different points of the lifecycle. They include 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 [Rubin and Chisnell 2008].

2.2. Ongoing example. 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 [Amaral 2005]. Due to complexity of this example, we will not delve much into the act of understanding domain-specific concepts of this particular DSL; its intended goal is to provide an understanding of the patterns' application to it.

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 that 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 appropriate 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, in this phase, physicists have to select those kinds 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 users Productivity problems in this domain. It aimed to develop reusable engineering methodologies through Model-Driven Development (MDD) 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 this process as much as possible, as well as to provide the End Users (with profiles ranging from the ones without programming expertise to high-level programmers) appropriate abstractions that hide the complexity of programming error prone algorithms in languages (e.g. C, C++ or Fortran), by 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 known-use of the 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 [Barišić et al. 2011].

2.3. Agile Development Process

It is necessary to establish iterative and incremental development process in which requirements and solutions evolve through collaboration of DSL development stakeholders. *Agile Development Process* breaks tasks into small increments and each iteration should fit in short time-boxes that typically not last more than a month [Martin 2003]. It promotes face-to face communication in workshops without impact of hierarchy roles of team members. All of them should take same level of responsibility that business and user needs are satisfied, by optimizing impact of evaluation feedback on language development. Appropriate iteration strategy that balance time invested into design of problem and its solution should be planned well with technical implementation. When goals are scoped and budget is fixed, we are ready to proceed to design and implementation activities that are guided by patterns given by *Iterative User-Centered Design*.

[Pattern] **USER AND CONTEXT MODEL EXTRACTION**

The main goal of designing a DSL, or any other language or software system, is to satisfy the user's requirements. We need to design the language in a way that the number of user profiles covered by Usability evaluation of a DSL should be significant in relation to the actual number of intended DSL End User profiles. This means that, in the majority of cases, the number of user profiles and contexts of use characteristics will also be relevant.

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

Forces

- *On-Budget Completeness.* The language developers need to balance the number of features that need to be incorporated in the language and evaluation design with the time and effort required to complete said design.
- *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.

By building a complete user and context model we are able to control for which extent of targeted user population, as well as environmental and technical range, Usability is reached. 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.

As the new domain concepts are identified for the DSL, potential users of those concepts, and contexts of use should be defined. 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 Usability Engineer need to be able to manage and extract feedback from a large number of users on a regular basis.

For that extent, building the context and user model should be done within the domain analysis phase of the DSL development.

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	5	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		

Example

The user model is obtained by identifying the list of main characteristics based on which categorization of user groups is accomplished. 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 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	air-condition system	3
	power	range:	550W-750W			heating machine	3
	office electrical power system:			secondary power:			
	keyboard:		optical				
	mouse:		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				
	mouse::		optical				

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 programming 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. 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.

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.

Related Patterns

- ITERATIVE USER-CENTERED DSL DESIGN. To begin the development process, it is required that the USER AND CONTEXT MODEL EXTRACTION is featured.
- EVALUATION PROCESS AND DESIGN PLANNING. While the resources for the user and context model are gathered, a plan for evaluation should also be considered.

Known uses

In usability testing one of the 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 [Rubin and Chisnell 2008]. Benefits of user and context modeling on management and final product are confirmed in areas of service and interface development.

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
			C++	ROOT	3
				ARTE	4
				BEE	5

[Pattern] **EVALUATION PROCESS AND DESIGN PLANNING**

During the development of a software artifact such as a DSL, the development team needs to carefully plan how the development stages should proceed and what are the required features that are to be developed in each step of development. In this case, the same attention must be given to Usability evaluations and experimental designs.

Problem

How to plan the processes of evaluation experiments and control the adequacy of the produced solutions to the intended users and respective context models?

Forces

- *Planning and Control.* Through good and careful planning the engineering team becomes more able to control and validate results, and to know the scope of their impact. Planning is a time consuming task and if not done carefully induces the risk of spending resources on evaluations with questionable validity and usefulness.
- *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' stand point all wished features and requirements are valid and essential. However, not all features fall within budget and not all users have the same amount of influence in the outcome and features of the DSL.
- *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 evaluation process and experimental designs, the Usability Engineer must document the main problem statements and their 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, Domain Experts and Language Engineers should list goals and system requirements that are basis for successful 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.

Table 4. Goal lists

SYSTEM GOALS	
Deal with petabytes of data.	5
Support hundreds of simultaneous queries.	5
Return partial results of queries in progress	4
Provide interactive query refinements.	4
Deal with data on secondary and tertiary storage access for simultaneous queries	3
Support statistical selection mechanisms (uniform random sampling)	4
Provide a flexible schema which supports versioning	3
Provide an environment for data analysis that is identical on desktop workstations and centralized data repositories.	3

EVALUATION GOALS	
Query steps in Pheasant vs. the object-oriented coding	5
Aggregation	3
Expressing a decay	4
Specification of filtering conditions	3
Vertex definition and the usage of user-defined functions	5
Path expressions (navigational queries)	4
Expressing the result set	3
The expressiveness of user-defined functions	4

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.

Goals are fulfilled by executing tasks, therefore we need to list and prioritize them to further decide how to design instrumentation and metrics to capture these tasks (see Table 5)

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

Related Patterns

– ITERATIVE USER-CENTERED DSL DESIGN. Developing an evaluation plan of action with goal and requirement analysis is an important starting point for iterative development.

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 [Reisner 1981].

Table 5. Task list

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

Table 6. List of comparison elements

COMPARISON ELEMENTS	
TEXTUAL VS. GRAPHIC SYNTAX	GENERAL PURPOSE VS. DOMAIN SPECIFIC
Expressive	Readability
Easy to learn	Accessibility
Syntax error Free	design reuse
Semantics error Free	high-level abstraction
Small Conceptual distance	clarity of program specification
Memorizable	program checking
Easy to use	language performance
Non-Ambigous	Maintainability
Formalizable	Portability
	Effectiveness

CAPTURE TEST
Final exams
Immediate comprehension
Reviews
Productivity
Retention
Re-learning

[Pattern] **ITERATIVE USER-CENTERED DSL DESIGN**

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 user's needs, it is usual to attain a solution which did not reached desired level of quality in use and quality of experience.

Problem

How to ensure that the domain-specific solution will result in increased level of users' productivity when compared to the existing baseline?

Forces

- *Cost of Usability Control vs. Cost of Future Modifications.* If we do not control usability tests during the several development stages, essential evaluation failures may lead us to meta-level changes that are equivalent to language development from scratch.
- *Development Cost.* Developing any language is a very expensive endeavor, more so because of the need to ensure that we will produce highly usable language that provides qualitative experience.

Solution

As we discussed previously in EVALUATION PROCESS AND DESIGN PLANNING, Productivity is related to the level of achieved Usability. Therefore, to prove the long claimed productivity increase provided by introducing DSLs, Usability Engineer need to introduce User-Centered methods to DSL life-cycle.

On other hand, in order to increase the chances of adoption by End 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, empowering them to drive the project and specify their use case scenarios. 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 approximations 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, and implement focused workshops to discuss the results, analysis of results and so on. An example of a one such schedules is shown in Table 7.

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 Expert	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 Expert	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	End User	14-24	
			Language Engineer	1-2	

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 development team the means to control the cost of evaluation.

As expected, the 200 hours 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 based on the previous ones.

Related Patterns

- USER AND CONTEXT MODEL EXTRACTION. User and context model need to be extracted so that is possible to plan which of them will have impact on iteration.
- EVALUATION PROCESS AND DESIGN PLANNING. Goals need to be explicitly expressed in order to plan each iteration.
- 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.
- ITERATIVE USER-CENTERED DESIGN. At the level of this pattern all patterns within the ITERATIVE USER-CENTERED DESIGN design space should be considered.

Known uses

The Usability engineering lifecycle is iterative by itself and should be merged with development of any product [Mayhew 1999]. Involvement of user-centered techniques in iterative development of software product is becoming common, and examples vary from user interfaces to data oriented applications [Catarci 2000].

[Pattern] **ITERATION VALIDATION**

Developing any form of complex software artifact, the professionals in charge of development need to constantly reevaluate priorities of features and requirements according to the way the project is developing, its goals, schedules and budget.

Problem

How to control which usability problems were solved, and analyze their possible relation with new ones that may arise?

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 development team to decide if these are considered new features or fixes to improve quality of solution. The latter should have top priority while the former should be carefully analyzed and sized.
- *Featurism vs. usability.* The Usability Engineer should clearly define the line where the number of features begins to jeopardize usability rather than promoting it.
- *Loss of focus.* 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
- *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 re-testing 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 Usability Engineer 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 in 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 the 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 successfully implemented and which still require some work (see Table 8). Understanding the status of usability evaluation for the current iteration allows us to redesign the schedule for the next few iterations.

Table 8. Iteration validation

VALIDATED	TO BE REVALIDATED	NOT ADDRESSED	ADDITIONAL FUNCTIONALITY
Expressing filter conditions	Path expressions	Environmental 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 [Völter and Bettin 2004] suggests, ITERATION VALIDATION requires the project team to validate if usability remains a concern throughout every iteration.
- ITERATIVE USER-CENTERED DSL DESIGN. Validation is a part of the iterative design and development process of a DSL.
- 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.
- USABILITY REQUIREMENTS TESTING. Based on requirements test results we have means to perform iteration validation.

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 [Völter and Bettin 2004].

[Pattern] **CONTEXT SCOPE TRADING**

During the development of the DSL, the development team needs to maintain both the focus of the development and the timeline and budget set by the project owners.

Problem

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

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 and 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 development team to ensure that both goals are achieved within the same package.
- *Sustainable focus.* 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 [Jones 1996].
- *Spread thin.* 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 engineering team to decide the iteration scope and to recognize how to profit from short iterations bursts.

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 agile 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 and 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 pattern, we define the User and Context scope as given in Figure 2.

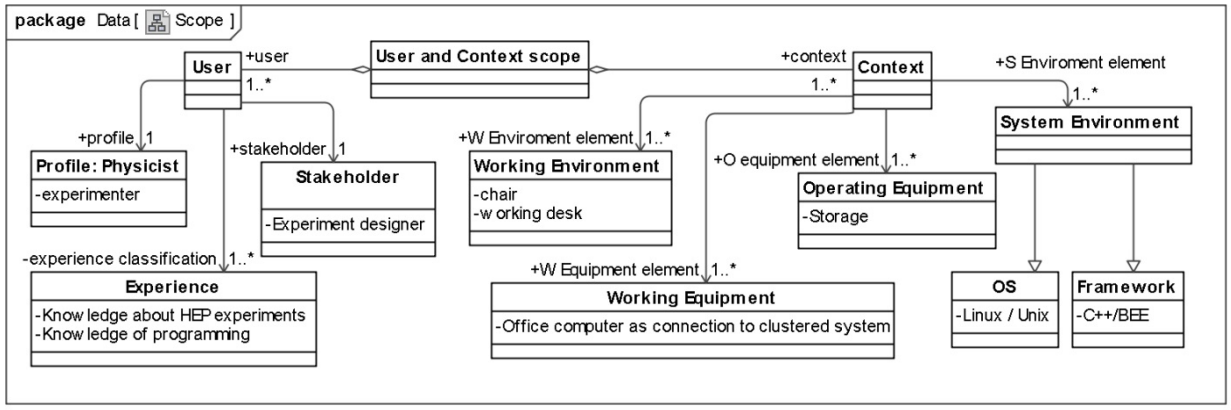

Figure 2. Language Use Scope

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.

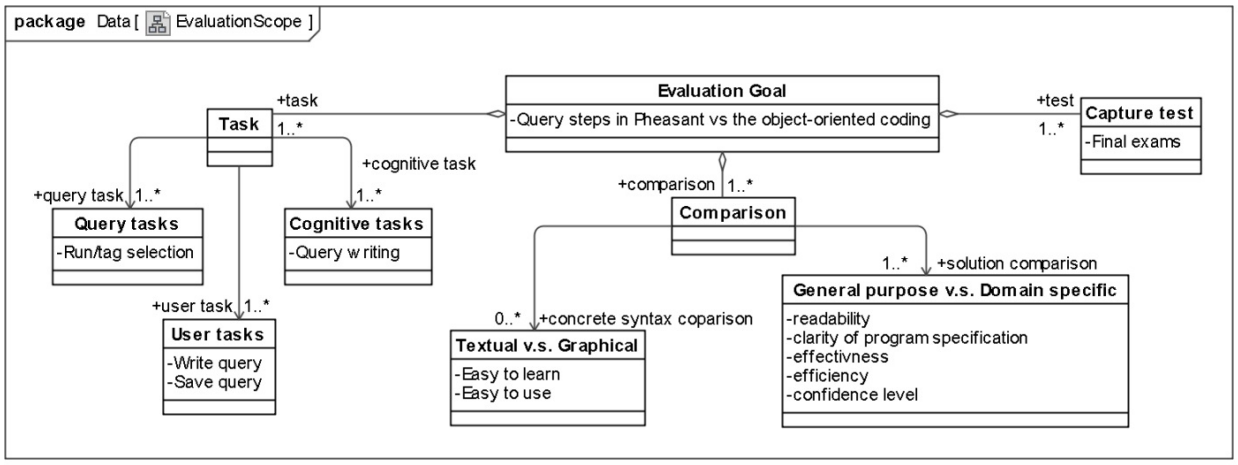

Figure 3. Evaluation Scope

Related Patterns

- SCOPE TRADING. These patterns are very similar in idea, however, while *Scope Trading* [Völter and Bettin 2004] 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 DESIGN. CONTEXT SCOPE TRADING is a mandatory development strategy of ITERATIVE USER-CENTERED DSL DESIGN.

– 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 [Völter and Bettin 2004]. Any evaluation requires precise definition of its scope, in order to be able to validate its results and indicates trade-offs in design decisions [Rubin and Chisnell 2008].

[Pattern] **FIXED BUDGET USABILITY EVALUATION**

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.

Problem

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

Forces

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

Solution

The engineering team 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 development team 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.

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 [Völter and Bettin 2004] 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 [Öztaş and Ökmen 2004].

Table 9. Budget evolution for Pheasant

INITIAL DATA	EXPECTED WORK DAYS	15 days (0-15)	5 days (16-20)	5 days (21-25)	7 days (26-32)	7 days (33-39)
	A priori Estimation	1.000 €	1.200 €	1.500 €	2.100 €	3.100 €
1ST ITERATION	Days	17	21	25	32	39
	Cost Estimation	1.050 €	1.250 €	1.550 €	2.150 €	3.150 €
	Cost Correction	+5%	+4%	+3%	+2%	+2%
2ND ITERATION	Days	17	21	25	32	39
	Cost Estimation	1.100 €	1.370 €	1.670 €	2.270 €	3.270 €
	Cost Correction	+5%	+10%	+8%	+6%	+4%
3RD ITERATION	Days	17	21	26	32	39
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.220 €	3.220 €
	Cost Correction		0%	-3%	-2%	-2%
4TH ITERATION	Days	17	21	26	32	39
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.070 €	2.970 €
	Cost Correction			0%	-7%	-8%
5TH ITERATION	Days	17	21	26	32	42
	Cost Estimation	1.100 €	1.370 €	1.620 €	2.070 €	2.970 €
	Cost Correction				0%	0%

2.4. Iterative User-Centered Design

It is necessary to engage the End Users in the language design in order to collect valuable information about their working scenarios and requirements [Righi and James 2007]. In order to assess appropriateness of given concept design decisions it is necessary to identify meaningful quality attributes for each domain concept and its use. Metrics should be defined and calculated based on their dependency to designed concepts and should be in conformance with evaluation goals. Finally, they are expected to result with concrete hypothesis, tests, metrics, samples and statements that should be addressed and validated trough *Experimental Evaluation Model*

[Pattern] **USABILITY REQUIREMENTS DEFINITION**

Understanding what is within the agreed budget for some project development is a skill that requires both a focus on the project and on the users' expectations by which the project's success is measured. When the main goal of the project is to achieve a usable solution, managing the users' expectations becomes much more important and can define the entire development strategy.

Problem

How to define expectations and desired usability of the intended DSL?

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 DSLs End 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. 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 and complete requirements model from the End Users point of view.
- *Language Choice.* When surveying commonly 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 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. We essentially need to have a way to keep all target user groups' needs in mind when developing the language.

The Usability 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 engineering team to define precise user scenarios 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 engineering team can find what quality means in the specific context of use for each user profile.

The solution provided intends to provide the basis by which the engineering team will define requirements and domain-specific goals that need to be considered. For a more in-depth explanation of this solution, we advise the reader to scan through the following example.

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 Table 10 we present the partial list of Usability requirements and tasks for Pheasant. They can be assessed at levels of Internal/External Quality, Quality in Use and Quality of Experience [Iso 2011].

Table 10. Usability Requirements

USABILITY REQUIREMENT	DESCRIPTION	INTERNAL QUALITY	EXTERNAL QUALITY	QUALITY IN USE	QUALITY OF EXPERIENCE
Understandability	The language features should be easy to understand, represented with familiar notation to user	Check consistency with physics notation	Validate ambiguous feature design decisions with Domain Expert	Give simple tasks to users and capture time and eye movement in order to find required features	Capture user opinion about features that take a longer time to be assessed by user
Expressiveness	Provide simple way to present complex queries	Repetitive construct flows of solving complex queries should be represented near each other	Comparison tests on effort needed to solve the same queries with different designs	Measuring time needed by expert users to solve complex queries	Feedback on logical flows of provided solution
	Improved readability of queries	Check query representations of baseline approach and its problems	Comparison tests on effort needed to solve the same queries with different designs	Correctness of query interpretation by end users	Capture user suggestions of improvements for contracts that are not interpreted correctly, likability and confusions of solution representation
Learnability	The user documentation and help should be complete	All syntactic elements of language should be well documented and consistent with metamodel change	All given language functionalities should be explained in documentation and followed by example	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 that provided description of use for each syntactic element covers all use cases that include that element	For given use cases, check coverage of the examples provided for given language functionalities	Check if the user is able to reuse same concepts in different context.	(Usually contextual help will present simple example. These should be checked with more complex examples)
	Language syntax elements should be easy to remember by the user	For each syntax element, ask the user to give it a meaning, and if it is confused ask for other suggestion	Provide examples on how to solve problems, and ask users to solve similar problem for which solution requires the same constructs (without consulting teaching materials).	Follow how frequently users ask for help to find same concepts (operators, relation symbols)	Capture repetitive misinterpretations of language elements by novice users and provide quick test to experienced users for that elements and collect feedback with additional suggestions
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 clicks/keystrokes to perform the task	Collecting feedback about likeability and pleasure that provided solution given to users
	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	Collecting feedback about likeability and pleasure that provided solution given to users

Operability	Language actions and elements should be consistent	Feature and behavior diagram validation with Domain Experts	Testing if all diagram relations and rules are implemented correctly	Correctness of solving tasks that are constructed based on scenarios from which diagrams were extracted	User opinion on improving consistency for tasks that have low level of correct solutions
	Error messages should explain how to recover from the error	Specifying language constructs where error recovery should be implemented	Testing error recovery by specification	Giving tasks that lead users to error messages and asking them for feedback about them	Collecting feedback about missing, misleading and incorrect error messages
	Undo should be available for most actions	specifying undo construct	Testing of undo construct	Capturing use of undo construct while solving tasks	Collecting feedback about missing, misleading and incorrect undo options
	Actions which cannot be undone should ask for confirmation	Specifying (dangerous) actions that cannot be undone	Testing all specified actions	Providing tasks that need solutions that cannot be undone by the user, and asking their opinion about them	Collecting feedback about missing undo confirmations and the actions that should not be specified like so
	Prevent users from producing syntax errors (e.g. misspelling)	Specifying model checkers inside the language	Implementing and testing model checkers	Capturing user's repetitive intent to produce same syntactic errors, and asking their opinion on how they can be more intuitive	Collecting syntax errors that may be produced by use of language in log files
	Prevent users from producing semantic errors (e.g. query not behaving as the user expects it to)	Specifying model checkers inside the language	Implementing and testing model checkers	Capturing incorrect query implementations and interviewing expert users about the given meaning (to identify cognitive problem solution or implemented meaning problems)	Capture user's frustrations of repetitive semantic errors

The diagram given by Figure 4 shows how different internal and external quality characteristics from ISO standards influence Pheasant's Usability.

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.
- EXPERIMENTAL DSL EVALUATION DESIGN. The usability requirements defined at the level of this are specified in QUALITY DESIGN MODEL that is part of *EXPERIMENTAL EVALUATION MODEL*

Known uses

Usability is seen as a special aspect in requirement engineering, of which the main phase is requirements definition [Carlshamre 2001]. Benefits of requirement engineering for MDD approach can be seen in examples of software product lines, supporting traceability and contributing to flexibility and simplicity in development [Alférez et al. 2008].

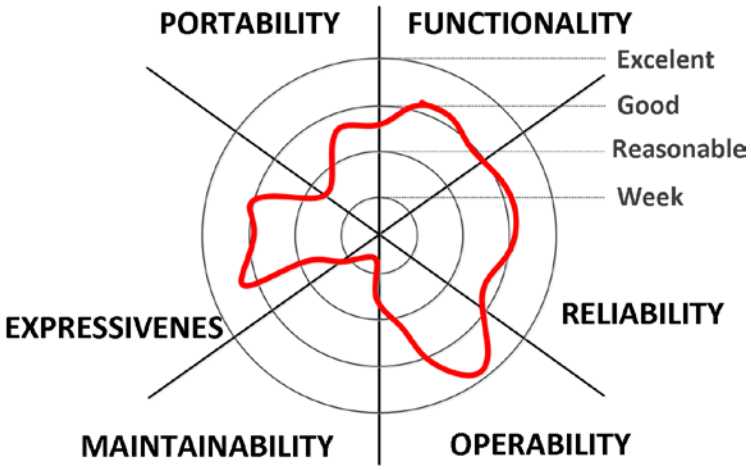

Figure 4. Kiviat diagram of Internal/External Qualities for Pheasant

[Pattern] **CONCEPTUAL DISTANCE ASSESSMENT**

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

Problem

How to measure conceptual distance between the user point of view to solve the problem and the provided solution?

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 engineering team 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.

Next, it is necessary to identify the frequency of different tasks that are covered by the iteration scenario. Tasks should be divided into subtasks that can be directly related with the domain concepts that will be tested.

This process will allow the Usability 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 described in Table 11.

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 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

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 the impact of Domain Concepts 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 MDD allows us to measure this distance using cognitive maps [Monteiro]. Application of this approach is visible in terms of analysis of cognitive effectiveness [Moody and Van Hillegersberg 2009], [N. Genon 2010].

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

[Pattern] **DOMAIN CONCEPT USABILITY EVALUATION**

There are many advantages of determining the required quality characteristics of a DSL before it is developed and used. Metrics are a common way to determine whether a software development project is within the parameters that were defined for its execution, i.e. budget and timeline. They are also useful to analyze whether some functional goals are being accomplished. For DSL development, the focus of metric-based analysis is the language metamodel.

Problem
How to capture domain concept related with usability problems using metrics?

Forces

- *Metamodel evaluation.* The level by which a metamodel is analyzed for usability issues has a direct relation to future failures in implementation. 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.
- *Agile development.* 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 the agile inclusion of usability requirements.

Solution
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.

Using metrics to analyze metamodel concept's representation allows the engineering team to reason on how different concept modeling will impact the Usability 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 engineering team 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. Examples of these metrics include:

- *Clone Analysis*. Like in GPLs, duplicated code is a very well-known code smell that indicates modularization problems[Beck et al. 1999]. 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 Pattern.* 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[Cho and Gray 2011].

Careful consideration of these and other available heuristics of actual usage of the DSL will allow the development team 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-Metamodel 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 engineering team 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.

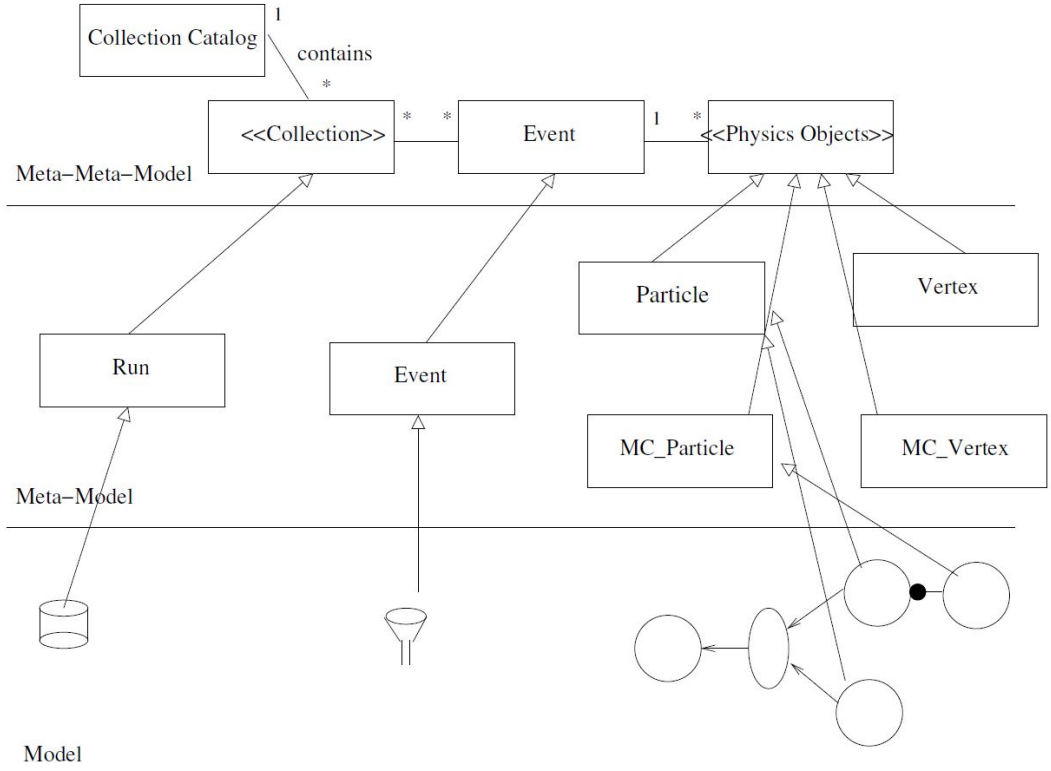

Figure 5. Corpora relation to the metamodel tasks (Taken from [Amaral 2005])

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 [Kosar et al. 2010]. This approach is also used in user interfaces evaluation by building a conceptual models [Johnson and Henderson 2002].

[Pattern] **USABILITY REQUIREMENTS TESTING**

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.

Problem
How to analyze if the goal usability requirements are being met by the DSL?

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 [Dumas and Redish 1999; Mayhew 1999; Nielsen 1994].

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 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 are expected to interact with 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 collected through workshops and small questionnaires, and reused for next iterations by extensions related to additional features. 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 Usability 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 Usability Engineer should define, with the assistance of key stakeholders, 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 roll back 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 engineering team 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	TREATMENT
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.

Related Patterns

- ITERATION VALIDATION. 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 DSL EVALUATION DESIGN. USABILITY REQUIREMENTS TESTING is a complementary activity to EXPERIMENTAL LANGUAGE EVALUATION DESIGN as the goals and test methodology differs.

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	

Known uses

This approach originates from usability engineering [Rubin and Chisnell 2008]. Its application can be seen in existing usability evaluation examples [Barišić, Amaral, Goulão and Barroca 2011], [Murray et al. 2000], [Conte et al. 2007].

[Pattern] **EXPERIMENTAL LANGUAGE EVALUATION DESIGN**

Using ITERATIVE USER-CENTERED DSL DESIGN, the Usability Engineer needs to define how to evaluate by which measure the language, or a prototype of the language, is in accordance with the elicited requirements.

Problem

How to design and control the process of empirical experimentation to get sound results?

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 [Basili 1996].
- *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 equitable 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 development team through experience usually has a pre-conceived idea of the hypothesis result. This can influence the behavior of the experiment's participant.
- *Evaluation Scarcity.* 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.

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 Usability 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 Usability; 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
- Context in which the experiment will be preformed
- Quality metrics that will be used
- 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 EVALUATION MODEL*, which should encompass enough details in order to be replicable by and independent source.

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.

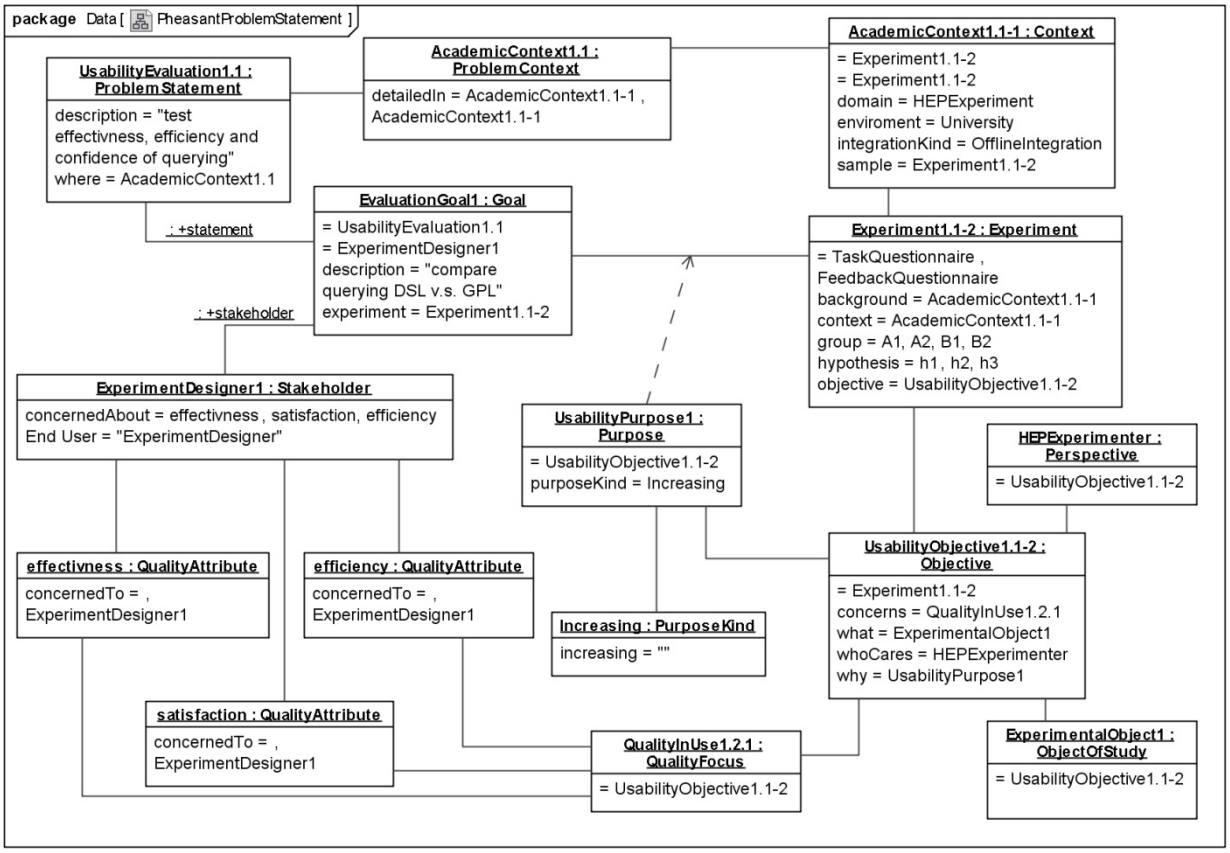

Figure 6. Pheasant experimental Problem Statement instantiation model

Example

Following with the example of Pheasant and experimental evaluation models [Barišić, Amaral, Goulão and Barroca 2012], 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 6).

The context of an experiment determines our ability to generalize from the experimental results to a wider context (Figure 7). 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 (Figure 8) 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*).

For Pheasant, the Usability 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.

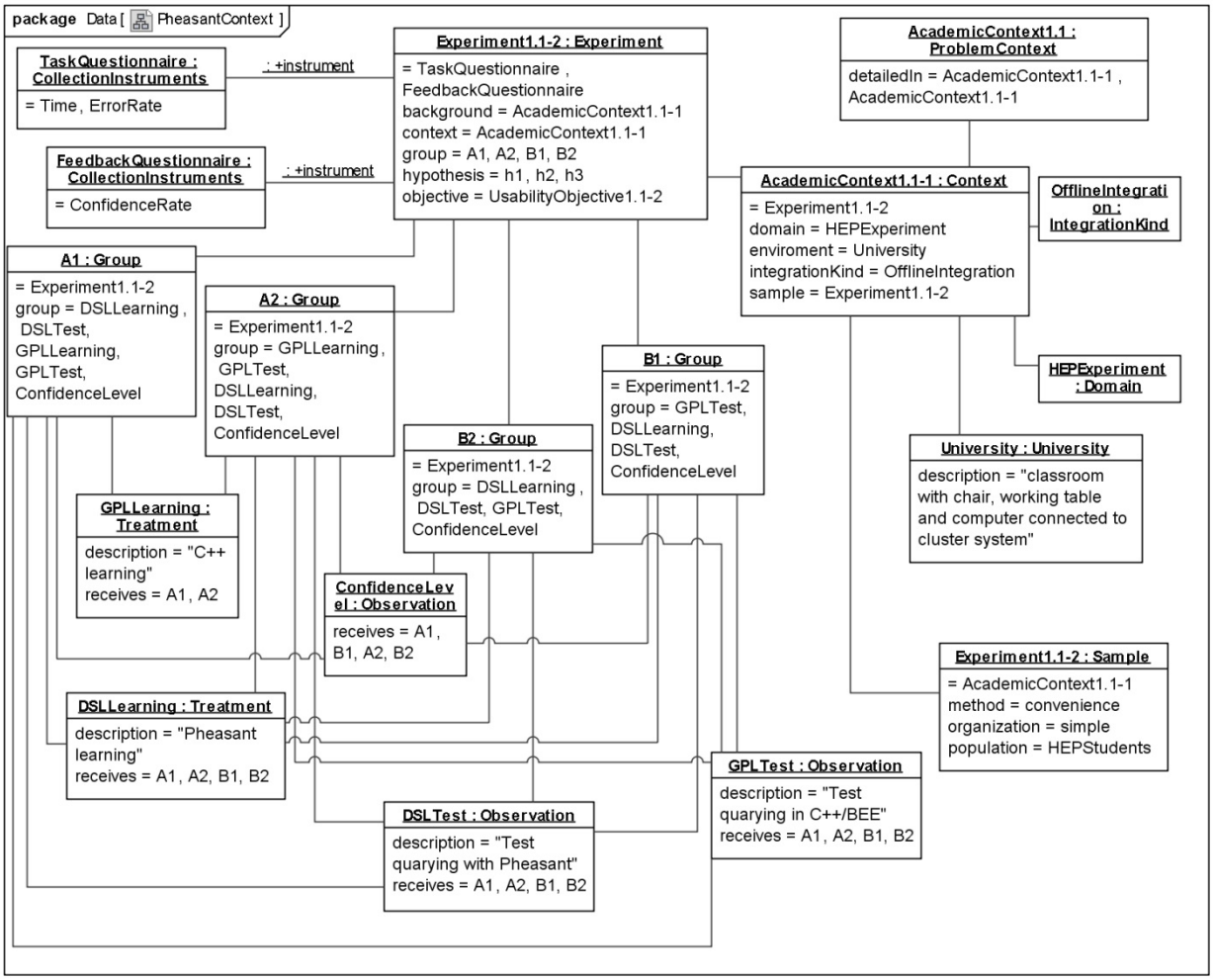

Figure 7. Pheasant experimental Context instantiation model

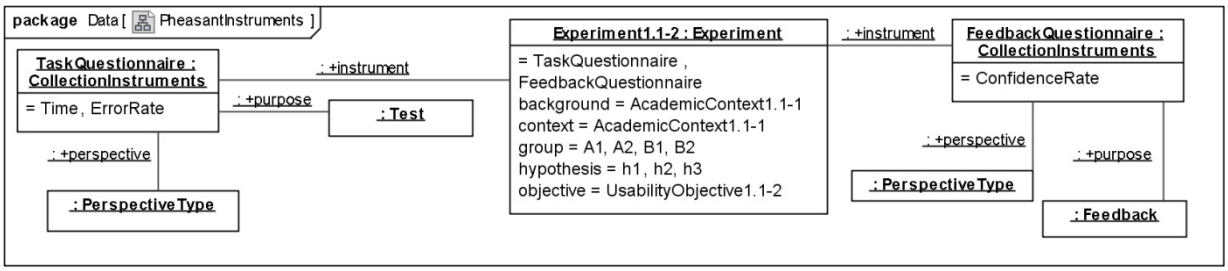

Figure 8. Pheasant experimental Instrumental design instantiation model

The Usability Engineer should clearly define the profile of the participants and the artifacts that are involved in the experiment (Figure 9).

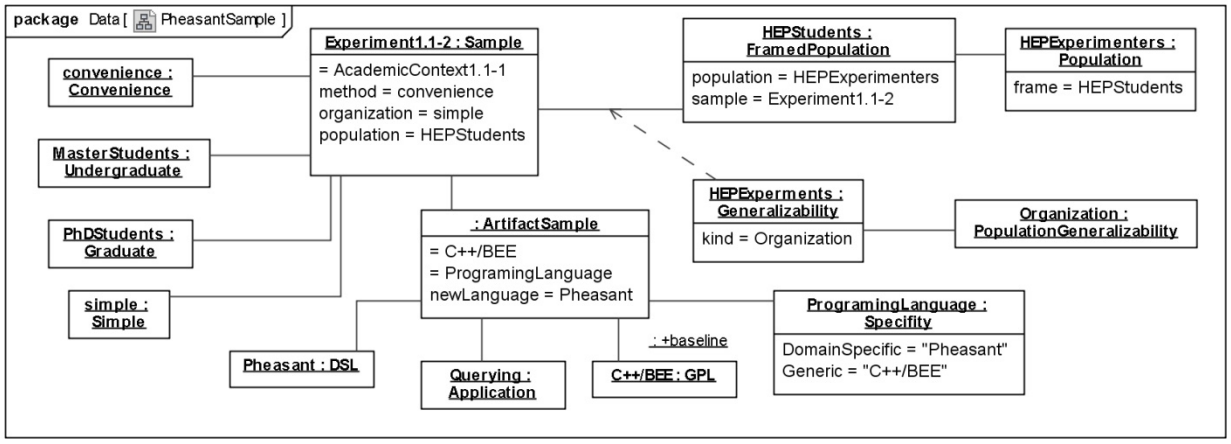

Figure 9. Pheasant experimental Sample design instantiation model

Quality focus needs to be defined through criteria, which can be recursively decomposed into sub criteria (Figure 10). 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 [García Frey et al. 2011].

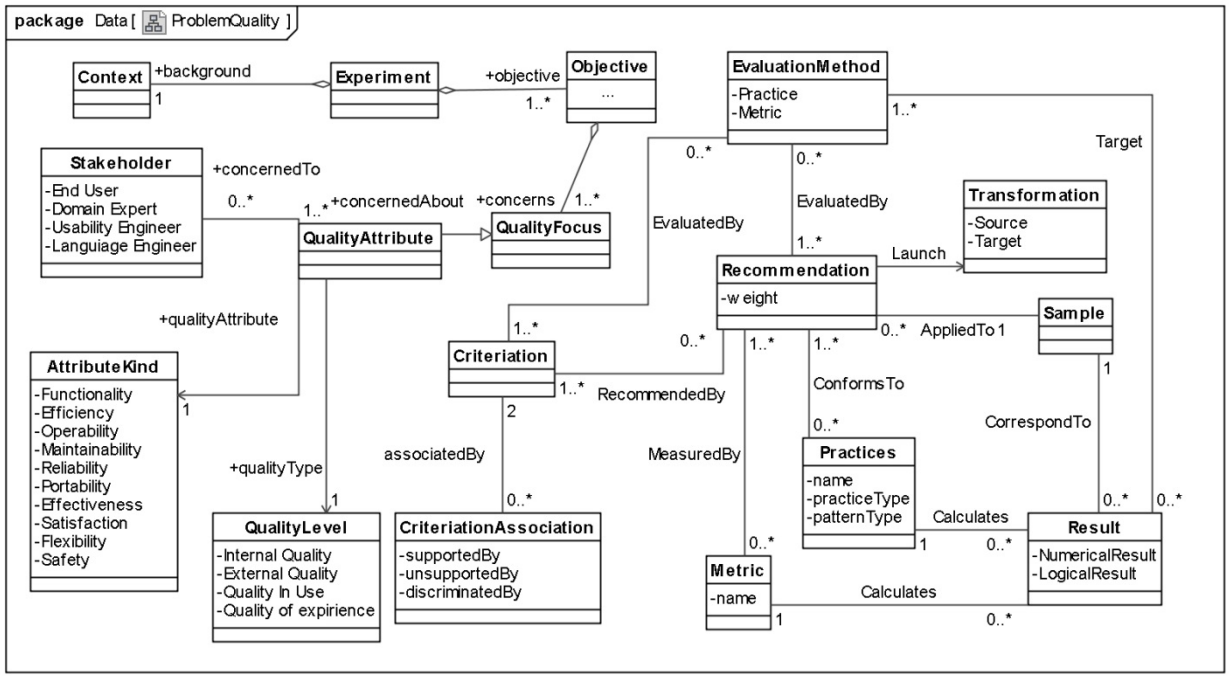

Figure 10 – Pheasant experimental Quality Design class diagram

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.

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 (Figure 11). 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.

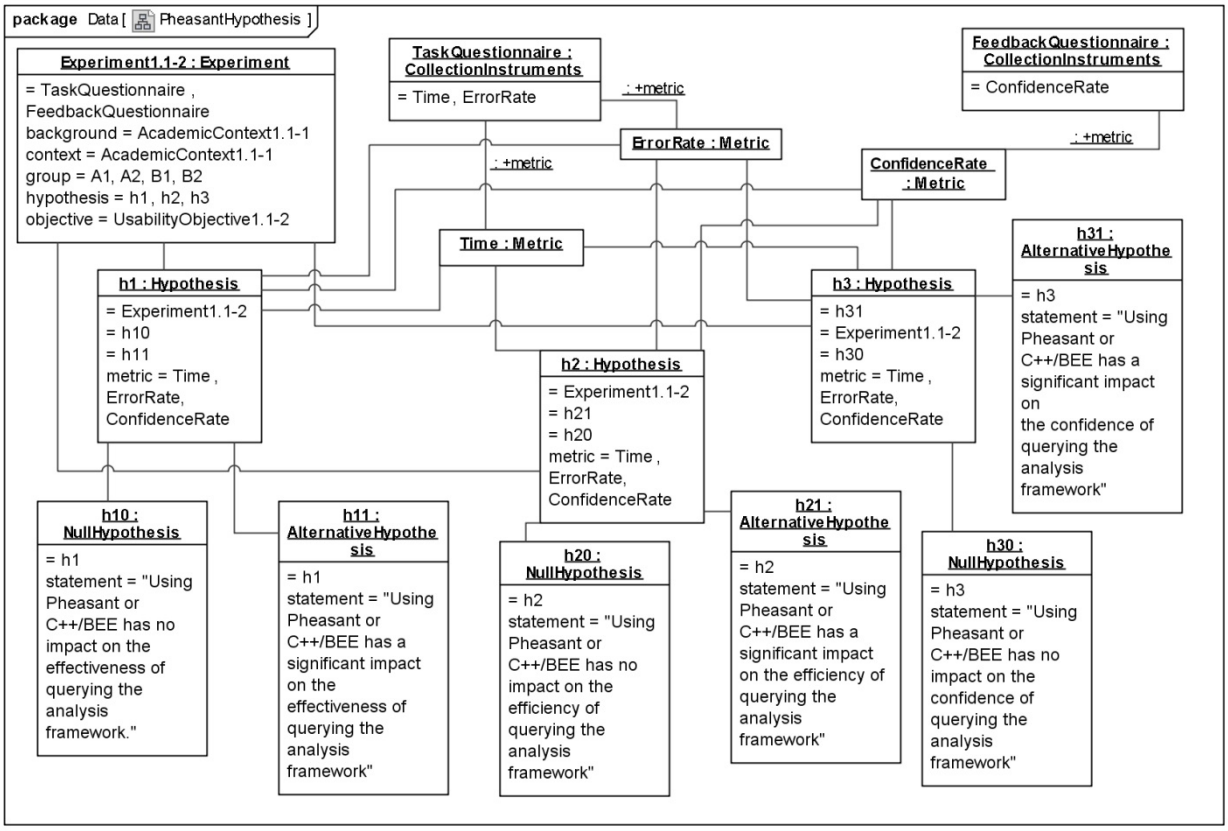

Figure 11 - Pheasant experimental Hypothesis and Variable design instantiation model

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 DSL EVALUATION DESIGN is fed back into USABILITY REQUIREMENTS DEFINITION to redefine the goals of the next iteration evaluation.
- USABILITY REQUIREMENTS TESTING. EXPERIMENTAL DSL EVALUATION DESIGN is a complementary activity to USABILITY REQUIREMENTS TESTING as the goals and test methodology differs.
- *EXPERIMENTAL EVALUATION MODEL.* Through this pattern we are setting the processes and scope of the *EXPERIMENTAL EVALUATION MODEL,* OF which example pattern applications are given.

Known uses

Detailed evaluation design is used in both usability engineering and experimental software engineering. This approach is modeled from the language comparison from [Goulão and Abreu 2007] and discussed in [Barišić, Amaral, Goulão and Barroca 2012].

3. RELATED WORK

There is a related line of work on HCI patterns, branching areas like ubiquitous systems [Roth 2002], web design [Van Duyne et al. 2003], safety-critical interactive systems [Hussey 1999], as well as more general interaction design languages [Oreilly 2007; Schmidt 2010; Tidwell 2005; Van Welie and Van Der Veer 2003]. 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 [Mahemoff and Johnston 2001], like News Box, Shopping Cart or Breadcrumbs.

Spinellis [Spinellis 2001] presents a pattern language for the design and implementation of DSLs. Contrary to ours, these patterns refer to concrete implementation strategies and not to the process of building the DSL or usability concerns. Günther [Günther 2011] 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 GPL capable of supporting internal languages.

Much of our patterns are based upon Völter and Bettin's pattern language for MDD [Völter and Bettin 2004]. 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[Folmer and Bosch 2003] 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 [Ferre et al. 2003] follows a similar approach. Graham's pattern language for web usability [Graham 2002] 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 [Gellner and Forbrig 2003]. 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 includes usability. For the world of DSL Engineering, it is a very important feature, because by raising level of abstraction this languages are meant to be used also by the people without programming education.

This paper gives a catalog of the patterns 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 framework to disseminate this knowledge and help bridge the gap.

In the future we intend to refine these patterns and continue to expand them based on their application on real-life development cases. 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 our helpful reviewers, especially Ademar Aguiar and André Santos, and to our program committee member Uwe Zdun that was following the shepherding process.

REFERENCES

Alférez, M., Kulesza, U., Sousa, A., Santos, J., Moreira, A., Araújo, J. AND Amaral, V. 2008. A Model-Driven Approach for Software Product Lines Requirements Engineering. In Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, July 1-3 2008 Knowledge Systems Institute Graduate School, 779-784.

Amaral, V. 2005. Increasing productivity in High Energy Physics data mining with a Domain Specific Visual Query Language. In Phd. Thesis, University of Mannheim.

Barišić, A., Amaral, V., Goulão, M. AND Barroca, B. 2011. How to reach a usable DSL? Moving toward a Systematic Evaluation. presented at the 5th International Workshop on Multi-Paradigm Modeling (MPM'2011) at Models 2011 Volume 50.

Barišić, A., Amaral, V., Goulão, M. AND Barroca, B. 2011. Quality in Use of Domain Specific Languages: a Case Study. In Proceedings of the presented at Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2011 Portland, USA, October 2011 ACM.

Barišić, A., Amaral, V., Goulão, M. AND Barroca, B. 2012. Evaluating the Usability of Domain-Specific Languages. In Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, M. Mernik Ed. IGI Global, 386-407.

Basili, V.R. 1996. The role of experimentation in software engineering: past, current, and future. In Proceedings of the 18th International Conference on Software Engineering (ICSE 1996)1996 IEEE Computer Society, 442-449.

Beck, K., Fowler, M. AND Beck, G. 1999. Bad smells in code. Refactoring: Improving the design of existing code, 75-88.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. AND Stal, M. 1996. A system of patterns: Pattern-oriented software architecture Wiley New York.

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

Catarci, T. 2000. What happened when database researchers met usability* 1. Information Systems 25, 177-212.

Cho, H. AND Gray, J. 2011. Design patterns for metamodels. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11 ACM, 25-32.

Conte, T., Massollar, J., Mendes, E. AND Travassos, G.H. 2007. Usability evaluation based on Web design perspectives IEEE, 146-155.

Dumas, J.S. AND Redish, J. 1999. A practical guide to usability testing. Intellect Ltd.

Ferre, X., Jusisto, N., Moreno, A.M. AND Sánchez, M.I. 2003. A software architectural view of usability patterns. In 2nd Workshop on Software and Usability Cross-Pollination, INTERACT, Zürich, Switzerland.

Folmer, E. AND Bosch, J. 2003. Usability patterns in software architecture. In 10th Int. Conf. on Human-Computer Interaction (HCII2003), pp. 93-97.

Gabriel, P., Goulão, M. AND Amaral, V. 2010. Do Software Languages Engineers Evaluate their Languages? In Proceedings of the XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), ISBN: 978-9978-325-10-0, Cuenca, Ecuador, April 2010, X. Franch, I.M.D.S. Gimenes AND J.-P. Carvallo Eds. Universidad del Azuay, 149-162.

García Frey, A., Céret, E., Dupuy-Chessa, S. AND Calvary, G. 2011. QUIMERA: a quality metamodel to improve design rationale ACM, 265-270.

Gellner, M. AND Forbrig, P. 2003. A Usability Evaluation Pattern Language. In Proceedings of the 2nd Workshop on Software and Usability Cross-Pollination, INTERACT, Zürich, Switzerland2003.

Goulão, M. AND Abreu, F.B. 2007. Modeling the Experimental Software Engineering Process. In 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007) IEEE Computer Society.

Graham, I. 2002. A pattern language for web usability. Addison-Wesley Longman Publishing Co., Inc.

Günther, S. 2011. Development of Internal Domain-Specific Languages: Design Principles and Design Patterns. In Proceedings of the PLoP 2011, Portland, OR, USA2011.

Hussey, A. 1999. Patterns for safer human-computer interfaces. Computer Safety, Reliability and Security, 686-686.

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

ISO 2011. ISO/IEC 25010-11 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

Johnson, J. AND Henderson, A. 2002. Conceptual models: begin by designing what to design. interactions 9, 25-32.

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

Jones, C. 1996. Software change management. Computer 29, 80-82.

Kelly, S. AND Tolvanen, J.-P. 2008. Domain-specific modeling: enabling full code generation. Wiley-IEEE Computer Society Press.

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

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

Mahemoff, M. AND Johnston, L.J. 2001. Usability Pattern Languages: the" Language" Aspect IOS Press, 350.

Martin, R.C. 2003. Agile software development: principles, patterns, and practices. Prentice Hall PTR.

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

Monteiro, M.P. September 2011. On the Cognitive Foundations of Modularity. In Proceedings of the Psychology of Programming Interest Group Annual Conference (PPIG), York, UKSeptember 2011.

Moody, D. AND van Hillegersberg, J. 2009. Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams. Software Language Engineering, 16-34.

Murray, N.S., Paton, N.W., Goble, C.A. AND Bryce, J. 2000. Kaleidoquery--a flow-based visual language and its evaluation. Journal of Visual Languages & Computing 11, 151-189.
N. Genon, P.H.D.A. 2010. Analysing the Cognitive Effectiveness of the BPMN Visual Notation. Software Language Engineering.
Nielsen, J. 1994. Usability engineering. Morgan Kaufmann.
OReilly, T. 2007. What is Web 2.0: Design patterns and business models for the next generation of software. Communications & strategies.
Öztaş, A. AND Ökmen, Ö. 2004. Risk analysis in fixed-price design–build construction projects. Building and Environment 39, 229-237.
Reisner, P. 1981. Human factors studies of database query languages: A survey and assessment. ACM Computing Surveys (CSUR) 13, 13-31.
Righi, C. AND James, J. 2007. User-centered design stories: real-world UCD case files. Morgan Kaufmann.
Roth, J. 2002. Patterns of mobile interaction. Personal and Ubiquitous Computing 6, 282-289.
Rubin, J. AND Chisnell, D. 2008. Handbook of usability testing: how to plan, design and conduct effective tests. Wiley-India.
Schmidt, V.A. 2010. User Interface Design Patterns Air Force Research Lab Wright-Patterson AFB OH Human Effectiveness Directorate.
Spinellis, D. 2001. Notable design patterns for domain-specific languages. Journal of Systems and Software 56, 91-99.
Tidwell, J. 2005. Designing interfaces. O'Reilly Media, Inc.
Van Duyne, D.K., Landay, J.A. AND Hong, J.I. 2003. The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience. Addison-Wesley Professional.
Van Welie, M. AND Van der Veer, G.C. 2003. Pattern languages in interaction design: Structure and organization. In Interact'03, I. Press Ed., Amsterdam, Netherlands, 527-534.
Völter, M. AND Bettin, J. 2004. Patterns for Model-Driven Software-Development. In Proceedings of the EuroPLoP'04, Irsee, Germany2004.