# Domain Usability of User Interfaces

Michaela Bačíková and Jaroslav Porubän

Department of Computers and Informatics,
Faculty of Electrical Engineering and Informatics,
Technical University of Košice,
Letná 9, 042 00 Košice, Slovakia
{michaela.bacikova,jaroslav.poruban}@tuke.sk

**Abstract.** In this paper we're referring to the usability definition which is only ergonomic. Usability however has also a second aspect beside ergonomic – domain. Domain usability, as we called it, can decide about the application's success or failure equally as the ergonomic usability. If the user does not understand the domain dictionary used in the application, he or she's not able to work with it and that means the whole application is useless. In this paper we try to redefine usability by creating new definitions of understandability and domain usability. Finally the total usability is formed by two inseparable parts âĂŞ domain and ergonomic. We realize that defining both terms will never be exact, because of the ambiguity and diversity of the natural language and variety of each person's thinking. It can however be a guide or a rule for creating applications that are as close as possible to a domain user.

**Key words:** user interfaces, usability, ergonomic usability, understandability

## 1 Introduction

*"Current graphical user interfaces are based on metaphors of real world objects and their related operations, which are well known to anyone from everyday life. Metaphors are presented by the user interface in graphical form as windows, icons and menus."*

Z. Prokoláb

The usability is often perceived as "ergonomics" or "human friendliness", user satisfaction with using the application or device, usefulness, effectiveness. Also the Nielsen's definition [1] is up to this day still used as the cornerstone of defining usability and also for placing additional guidelines for creating user interfaces. But nor the Nielsen's definition nor the known guidelines (like [2], [3], [4] or [5]) do not deal with the side of the usability related to the domain content or consistency of user interface domain dictionary at all, or they refer to it only in specific boundaries of their context.

Therefore we think there is a need for defining domain usability as an important attribute of usability. That way at least a basic guideline would exist for

designing and creating applications, which would correspond to the real world. The importance of matching the application terminology to the real world shows the experiment described in [6] which proved, that applications with domain structure better matching the real world have a better performance and usability.

Without the right terms occurred in the application's interface, it is not usable. Although the user interface is really good-looking, if the user does not understand the labels of various buttons or menu items, he or she cannot work with it and therefore the whole application is useless. Consequently the domain usability is of a great importance and it can be the decisional point between the application success or failure.

Currently there is a huge amount of applications, which differ not only by their appearance, but also by the terminology used. Even different systems in one specific domain differ in their textual content. It happens in several cases, that the application does neither use the terms from the domain, or they contain errors, which prevents the users from working with the application.

During the design and implementation the programmers usually aim to create good-looking and perfectly error-free applications. They order the components effectively so the user would not be restrained in their work. The well-known rule thumb is as follows: *"The application should assist the user while performing their work, not getting in the way of it"*. The programmers however often have a different perspective of how to work with the application than the domain users. They are often more experienced and they have their established style of work. But from the domain point of view, they often-times have only a little knowledge about the specific domain, for which the application is developed. Thus they are not capable of correctly transferring the domain terms, relations and processes correctly into the application.

The problem with defining guidelines for domain usability is also the ambiguity and diversity of natural language. There is no rule for people to use their language in a specific way. Therefore if there would be some basic guidelines, then these guidelines would not be so specific, but only generic when related to the terms, which should be used. The domain usability is often perceived as implicit, which is good for the designer, but not good for a domain user.

To summarize our existing knowledge we identified the main problems as follows:

- There are no clear rules for designing the term structure of an application, so it corresponded with the domain.
- At present there are no official guidelines that talk about applications should matching the real world or describe domain terms and processes.
- Variety of human thinking
- Ambiguity and diversity of natural language

In this paper we therefore strive for defining the domain usability and we call the Nielsen's usability ergonomic. Both types of usability âĂŞ domain and ergonomic then form the overall usability of systems or devices. We also introduce some examples to illustrate the domain usability definition. We realize that

defining the domain usability is not and will never be exact, because of the ambiguity and diversity of the natural language and variety of each person's thinking. It can however serve as a guide or rule for creating applications in a manner, that they would be as close as possible to a domain user.

## 2 State of the Art

Jacob Nielsen in 1994 already underlined the importance of textual content of User Interfaces (UIs). He talks only very generally about it, but he stresses the importance of "the system's addressing the user's knowledge of the domain" [1].

Prokoláb in his work [7] deals with the problem of ambiguity of user interface term apparatus design and term semantics in the interface. His solution to introduce the so called "semantic user interfaces", the core of which is a general ontology, which is a basis for creating all interfaces in the specific domain. The interfaces can be of different appearance and arrangement, but the domain dictionary must remain the same. Prokoláb also stresses the importance of application's copying the domain and he identifies the problem of absence standards for the known theoretical background, which would reduce the designers of intuitive interfaces in choosing voluntary attributes (menu labels, buttons, tool-tips, menu topology) for displaying metaphors of user interface. That's why UIs become diverse even inside a specific application domain.

The paper [6] deals with benefits of matching the domain content in the software with the target domain. They refer to the domain content of the application as to the "domain structure". They performed an experiment where they implemented a new version of a NASA application with an emphasis on copying the real world better. They found out that there is a big difference in the performance of the old application and the new, while the new application was better. They stress the need of application's corresponding to the real world and respecting it as early as during the application design.

Shneiderman [8] deals with the complexity of the interface and he stresses that the complexity of the textual content should not be too high for the application will be less usable. The study of web usability for older adults [9] besides other attributes also deals with the translation and reading complexity of web pages. The authors turn to indexes for evaluating the complexity of text in the interface like the ARI index [10] or the Kincaid index [11].

Badashian et al. [2] try to define the basic guidelines for creating usable interfaces. Among other they also stress the importance of the consistency of the terms used in the application and of matching between the system and the real world.

Understandability of web pages is defined in the area of web accessibility [4]. Comparing to our definition however, it deals with only with some of the attributes mentioned in our definition: interface language, language barriers and errors. It focuses only on web pages specifically, not on user interfaces in general. It also defines other attributes that do not fit into the context of the user interface

usability. Similar properties are defined also in [2], but the domain usability is not explicitly defined.

The state of the art paper [12] from 2001 contains a wide view of automatic usability methods and tools. From over 100 tools, only the Sherlock tool deals with domain content of applications. It is a tool for Windows interfaces and rather than addressing ergonomic factors, it focuses on task-independent consistency checking (e.g. same widget placement and labels) within the UI or across multiple UIs. User studies have shown a 10 to 25% speed-up for consistent interfaces [13]. Sherlock evaluates visual properties of dialog boxes, terminology (e.g. identify confusing terms and check spelling), as well as button sizes and labels.

Hilbert in [14] deals with automated usability information extraction from event sequences, which are the natural outcome of normal operation of each user interface. The event sequences should correspond with the real world as much as the domain dictionary.

The number of works referring to matching the application's content to the real world indicates the importance of the domain usability.

## 3 Original Usability Definition

Usability often refers to one single attribute of a software system or of a device. It is however a big number of different interconnected attributes. **Usability** was first defined by Nielsen [1] as a whole (but diverse) property of a system, which traditionally is related to these five attributes:

- *Learnability*: The system should be easy to learn so that the user can rapidly start getting some work done with the system.
- *Efficiency*: The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
- *Memorability*: The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
- *Errors*: The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.
- *Satisfaction*: The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

Usability can represent a decisive factor between system success or failure.

However the usability guidelines have changed and evolved through the time, the usability definition remained unchanged since Nielsen first defined it [1] in 1994 it still serves as a basic guide to create usable software systems.

## 4 Ergonomic and domain usability

The definition of usability that was mentioned in the chapter 3 perceives usability from an ergonomic point of view, i.e. in the terms of user experience, satisfaction with using the application, application quality, and effectiveness.

But also other factor has its own impact on the usability of an application: *domain content* of the application user interface, to which we focus on in our research. Each software system is developed for a concrete domain therefore its interface should contain terms, relations and describe processes from this specific domain for the user to be able to work with it. If the user does not understand the terms, relations or processes in the interface, then the whole application is useless. This application feature can be called understandability and we will try to define it as follows:

**Understandability** is the property of a system that affects usability and which relates to following factors:

- content: interface of an application system should map terms, relations and processes from the domain, which it is designed for.
- *Consistency*: words used throughout the whole system should not differ, if they describe the same functionality, the dictionary should be consistent.
- *World language used in the interface*: the language of the interface should be the language of the user.
- *Domain specificity*: the interface should contain terms specific for a particular domain, not general.
- *Language barriers and errors*: the interface should not create language barriers for the user and it should not contain language errors.

Each of these properties will be discussed further in this paper.

The type of usability that's affected by the factor of UI understandability we can call **domain usability**.

So the usability can be divided into two basic types: **ergonomic usability** which was redefined in the chapter 3; and **domain usability** which was defined here as a new term. These two types can be combined together when evaluating usability: for example during testing of the number of steps needed to execute a particular task. A user gets a task to execute on two different user interfaces made for the same domain. Both ergonomic and domain factor effect the completion of the task.

According to [14] usability can be also divided into:

- *Formative* – enables providing feedback to designers to inform and evaluate design decisions.
- *Summative* âĂŞ- primarily involves making judgements about "completed" products, to measure improvement over previous releases or to compare competing products.

The formative and summative specification can be applied both to the domain or ergonomic usability.

The individual aspects of the understandability will be further discussed in the following subsections along with illustrating examples.

### 4.1 Domain Content

The interface of an application system should map terms, relations and processes from the domain, which it is designed for.

Imagine the system is manufactured for the domain of medicine. Without any explanation or referring to sources for better understanding of the issue, the user interface of such application should not contain terms and relations for example from domains of building construction or traffic. It should however contain terms from medicine. The logical reason is that a medic is not familiar with the dictionary of building constructor or traffic manager. Logically, the interface should also define processes from the specific domain of medicine by implementing sequences of events that can be executed on the interface.

### 4.2 Consistency

Words used throughout the whole system should not differ, if they describe the same functionality, the dictionary should be consistent.

Clear words and commands should serve as a standard through the whole system, especially if the system consists of several subsystems. For example it is better to use the same wording labels for buttons or menus that have the same functionality throughout the whole system.

The consistency is supported today by localization property. Although consistency was not the primary goal of localization, rather it was to ensure translation: all words used in the application interface are stored in one place and all of them will be translated into another language.

The main idea behind localization is a set of files containing all words that are displayed in the application interface. These words are translated into several languages that are supported by the application. Each translation is segregated into a separate file. A mechanism for obtaining these words is implemented in a library. When the application language is changed, the application just takes another localization file and reads the terms from it. For example BlackBerry JDK uses its own special type of localization files. Android uses xml files to store the text values.

### 4.3 World language

The language of the interface should be the language of the user (or user expert). If the application is developed for Slovak market, then it should use Slovak language as the primary language. If the application is developed for users from several different countries, it should provide translation to different languages and a possibility to switch between them. However the terms in one single translation should be only of the same language, i.e. the translation should be complete there should be no foreign words.

### 4.4 Domain specificity

The interface should contain terms specific for a particular domain. Terms that are too generic usually reduce usability of the user interface. If the system is made for a wider sphere of customers, then it should not use generic terms. As an example we can imagine a system for Slovak schools âĂŞ elementary schools, high schools and universities altogether. Each of these types of schools uses its specific terms.

The elementary and high schools in Slovakia have lessons, they get exams and tests, they are evaluated with grades ranging from 1 to 5 and at the end of the school year the schoolchild gets a certificate. The university students have exercises and lectures, they get tests and final exams, they are evaluated with grades ranging from A to E and FX and at the end of the bachelor or engineering degree study the student gets a diploma.

In the real life it's not possible to use lectures for elementary students, or grades ranging from 1 to 5 for the university students. The elementary students also cannot make the final exams or get a diploma. Logically these terms in the application's user interface should differ depending on the application's intended area of use.

### 4.5 Language barriers and errors

The interface should not create language barriers for the user and it should not contain language errors. For example using abbreviations without any explanation of their meaning, foreign terms, and words not translated from other language or grammatically incorrect words can be seen as a language barrier.

## 5 Conclusion

In this paper identified the main problems related to the domain content of user interfaces. We identified a second usability aspect beside ergonomic and we called it domain usability. We redefined usability and we defined a new term - understandability as the property of a system or a device that has a primary impact on usability equally as the ergonomic usability. We provided examples to illustrate different attributes of domain usability. They can also serve as guidelines for creating applications, that will match the real world and this way they will be as close as possible to a domain user.

## Acknowledgements

This work was supported by VEGA Grant No. 1/0305/11 Co-evolution of the Artifacts Written in Domain-specific Languages Driven by Language Evolution.

## References


1. J. Nielsen, *Usability Engineering,*. Academic Press, Boston, 1994.
2. A. S. Badashian, M. Mahdavi, A. Pourshirmohammadi, and M. M. nejad, "Fundamental usability guidelines for user interface design," in *Proceedings of the 2008 International Conference on Computational Sciences and Its Applications*, ser. ICCSA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 106–113. [Online]. Available: http://dx.doi.org/10.1109/ICCSA.2008.45
3. "Java look & feel design guidelines, version 2.0," http://java.sun.com/products/jlf/ed2/book, [Online May 2012].
4. W3C, "Web content accessibility guidelines (wcag) 2.0, part 3 about understandability," http://www.w3.org/TR/WCAG20/, 2008.
5. Google, "Android user interface guidelines," http://developer.android.com/guide/practices/ui_guidelines/index.html, [Online May 2012].
6. D. Billman, L. Arsintescucu, M. Feary, J. Lee, A. Smith, and R. Tiwary, "Benefits of matching domain structure for planning software: the right stuff," in *Proceedings of the 2011 annual conference on Human factors in computing systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2521–2530. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979311
7. K. Tilly and Z. Porkoláb, "Automatic classification of semantic user interface services," in *Ontology-Driven Software Engineering*, ser. ODiSE'10. New York, NY, USA: ACM, 2010, pp. 6:1–6:6. [Online]. Available: http://doi.acm.org/10.1145/1937128.1937134
8. B. Shneiderman, "Response time and display rate in human performance with computers," *ACM Comput. Surv.*, vol. 16, no. 3, pp. 265–285, Sep. 1984. [Online]. Available: http://doi.acm.org/10.1145/2514.2517
9. S. A. Becker, "A study of web usability for older adults seeking online health resources," *ACM Trans. Comput.-Hum. Interact.*, vol. 11, no. 4, pp. 387–406, Dec. 2004. [Online]. Available: http://doi.acm.org/10.1145/1035575.1035578
10. J. P. Kincaid, R. P. Fishburne, R. L. Rogers, and B. S. Chissom, "Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel." Tech. Rep., Feb. 1975. [Online]. Available: http://www.eric.ed.gov/ERICWebPortal/detail?accno=ED108134
11. J. P. Kincaid and M. W.C., "An inexpensive automated way of calculating flesch reading ease scores. patient disclosure document 031350," Us Patient Office, Washington, DC, 1974.
12. M. Y. Ivory and M. A. Hearst, "The state of the art in automating usability evaluation of user interfaces," *ACM Comput. Surv.*, vol. 33, no. 4, pp. 470–516, Dec. 2001. [Online]. Available: http://doi.acm.org/10.1145/503112.503114
13. R. Mahajan and B. Shneiderman, "Visual and textual consistency checking tools for graphical user interfaces," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 722–735, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1109/32.637386
14. D. M. Hilbert and D. F. Redmiles, "Extracting usability information from user interface events," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 384–421, Dec. 2000. [Online]. Available: http://doi.acm.org/10.1145/371578.371593