# Usability Recommendations for Annotation Tools

**Manuel Burghardt**
Media Informatics Group
University of Regensburg
manuel.burghardt@ur.de

## Abstract

In this paper we present the results of a heuristic usability evaluation of three annotation tools (GATE, MMAX2 and UAM CorpusTool). We describe typical usability problems from two categories: (1) general problems, which arise from a disregard of established best practices and guidelines for user interface (UI) design, and (2) more specific problems, which are closely related to the domain of linguistic annotation. By discussing the domain-specific problems we hope to raise tool developers' awareness for potential problem areas. A set of 28 design recommendations, which describe generic solutions for the identified problems, points toward a structured and systematic collection of usability patterns for linguistic annotation tools.

## 1 Introduction

To find valuable clues about annotation tools and the role of usability, we have reviewed the LAW proceedings from 2007-2011[1] (altogether 140 articles) systematically with regard to their main topics. As expected, most articles are concerned with linguistic corpus annotation scenarios, which are oftentimes realized by deploying automatic tools. However, articles which use a manual or semi-automatic annotation approach are just as frequent. Most manual annotation projects rely on annotation tools, which are either selected from the wide range of freely available tools, or crafted for the very project. Although the usability of such tools, which is oftentimes paraphrased as *ease-of-use* or *user-friendliness*, is generally understood as an important factor to reduce time and effort for laborious annotation projects (Dandapat et al., 2009; Santos and Frankenberg-Garcia, 2007), a serious account on how to systematically test and engineer usability for annotation tools is largely missing. Dipper et al. (2004) are amongst the few who evaluate the usability of a selection of tools in order to choose an adequate candidate for their annotation project. In other respects, usability is only mentioned as a rather vague requirement that is (if at all) implemented according to the developer's personal assumption of what makes a usable tool (cf. e.g. Eryigit, 2007).

The rest of the paper is structured as follows: in chapter 2 we show that usability is not some vague postulation, but actually a criterion that can be measured and systematically engineered. Chapter 3 describes the testing method that has been applied to evaluate three annotation tools (GATE, MMAX2 and UAM CorpusTool) in order to reveal typical usability problems. We discuss the results of the evaluation in chapter 4 and present usability recommendations for annotation tools in chapter 5. These recommendations will help developers to design tools which are more usable than current implementations. They can also be used as a basic usability checklist for annotators who have to choose from the wide variety of available tools. Finally, the set of recommendations will serve as a starting point for further research concerning the usability of annotation tools, with the ultimate goal being to provide a wholesome collection of usability patterns for this

[1] http://www.cs.vassar.edu/sigann/previous_workshops.html



*Proceedings of the 6th Linguistic Annotation Workshop*, pages 104–112,
Jeju, Republic of Korea, 12-13 July 2012. 

very domain. Chapter 6 provides an outlook to the wider context of this particular study.

## 2 Usability fundamentals

### 2.1 Defining usability

According to Nielsen (1993), usability can not be described as a one-dimensional criterion, but must rather be seen as a concept that consists of multiple components such as *learnability*, *efficiency*, *memorability*, *error rate* and *satisfaction*. Each of these usability components can be measured individually, thus making the hitherto vague concept of usability more concrete. There are also more formal definitions, e.g. the ISO 9241-11 standard (1999), which characterizes usability as

> "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."

Barnum (2011) emphasizes the use of the term *specified* in this definition, which indicates that usability has to be engineered for a specific user with specific goals in a specific context.

### 2.2 Usability engineering

Usability engineering can be seen as a set of activities, which describe a systematic way to create usability for a system throughout its development life-cycle. Hence, there are several suggestions for usability engineering life-cycles, which show similarities and parallels to existing software engineering and development processes. The ISO standard for *human-centered design of software* (ISO 9241-210, 2010) describes four elementary stages: (1) understand and specify context of use, (2) specify user requirements, (3) produce design solutions, (4) evaluate designs and iterate the previous steps if necessary.

### 2.3 Usability testing

Usability testing is an important activity throughout the usability engineering life-cycle (cf. stage 4 of the ISO 9241-210 process), but it may also be used as a stand-alone-method, to achieve one of the following goals:

(I) To find out which system is better (comparison)

(II) To judge how well a system works (summative judgment)

(III) To find out why a system is bad (reveal usability problems)

The annotation tools evaluated in this paper are neither compared to each other, so as to find out which one is best, nor are they tested against some predefined set of criteria. The goal of our evaluation is to reveal usability problems for existing annotation tools (cf. goal III).

There is a huge collection of different usability testing methods, which can be used to conduct a usability evaluation. Basically, they can be divided into two main categories (Rosson and Carroll, 2002): *Empirical methods*, which collect information about the usability of a system by observing and interviewing actual users, and *analytic methods*, which rely on usability-experts who try to put themselves in the position of actual users. Usually, analytic methods are used early in the design process because they are less laborious than empirical methods. Empirical methods however are by nature more demanding, as they require real users, and the data has to be interpreted by usability experts afterwards. Among the analytic methods are the so-called *inspection methods*, which include e.g. the *cognitive walkthrough* (CW) and the *heuristic evaluation* (HE).

**Cognitive walkthrough** — During a CW the evaluator tries to put himself in the position of an actual user in order to explore and experience the system from the user's point of view. It is important to know the basic characteristics of the actual user (e.g. by observing real users) and to make use of four control questions (Wharton et al., 1994) (cf. Table 1).

The CW method can be described as being very structured and task-oriented: the evaluator explores and tests the system as he tries to solve some predefined tasks step by step. These tasks have to be designed in such a way as to ensure that the evaluator will experience the most important features of the system. The evaluator documents every step, either positive or negative, on his way to solving the task.

| | |
|---|---|
| **Q1** | Will users know what they need to do next to accomplish their task? |
| **Q2** | Will users notice that there is a control available that will allow them to accomplish the next part of their task? |
| **Q3** | Once users find the control, will they know how to use it? |
| **Q4** | If users perform the correct action, will they see that progress is being made toward completing the task? |

Table 1: Control questions to support empathy with the actual user.

**Heuristic evaluation** — Basically, the HE is a rather unstructured expert evaluation, where a collection of usability principles (the heuristics) serves as a basic guideline for the usability-experienced evaluator. The heuristics are formulated in a generic way and are meant to provide some basic structure for the evaluation process. Among the most widely-known sets of usability heuristics are Nielsen's (1994) ten heuristics[2] (cf. Table 2).

| | |
|---|---|
| **H1** | Visibility of system status |
| **H2** | Match between system and the real world |
| **H3** | User control and freedom |
| **H4** | Consistency and standards |
| **H5** | Error prevention |
| **H6** | Recognition rather than recall |
| **H7** | Flexibility and efficiency of use |
| **H8** | Aesthetic and minimalist design |
| **H9** | Help users recognize, diagnose, and recover from errors |
| **H10** | Help and documentation |

Table 2: Nielsen's heuristics for user interface design.

These heuristics are intended to facilitate the discovery of actual usability problems, as the evaluator relates identified usability problems to one or more heuristics and ranks the severity of the problem. Once the evaluation is finished, the heuristics make it easy to cluster usability problems and to identify those problematic areas where the system needs to be improved.

A HE can be conducted by multiple evaluators. For the ideal cost-benefit ratio, Nielsen (1994) recommends 3-5 evaluators, as this number of evaluators on average discovers about 60-75% of all potential usability problems of a system. The ideal evaluator is a double-expert, i.e. he is both a domain expert and a usability expert (Nielsen, 1992).

**Heuristic walkthrough** — Sears (1997) describes the heuristic walkthrough (HW) as a method which sorts out some of the problems of existing inspection methods. Among the problems of the HE is its lack of structure and its strong focus on abstract heuristics. As a result, the heuristic evaluator is prone to find only problems that are captured by the heuristics, or if still unexperienced, he might even find false usability problems by misinterpreting the heuristics. While conducting a HE it is important to know that not every violation of a heuristic results in a usability problem. Sometimes the violation of one heuristic can be interpreted as an intentional compromise for not violating three other heuristics. The CW on the other hand has too much structure by relying on a list of user tasks and a guided set of questions. The CW approach discourages the discovery of usability problems that are not covered by the tasks or the questions.

The HW method borrows ideas from both, HE and CW: from HE it takes the free-form evaluation and the list of usability heuristics, from CW it takes the idea of user tasks and the check-questions, which emphasize the most important steps during a dialog. The HW also incorporates ideas from the usability walkthrough method (Karat et al., 1992), which is a two-way process consisting of a heuristics-based, free-form evaluation, and a more structured, task-based phase.

## 3 Usability evaluation of annotation tools

This study applies the HW method to demonstrate that the usability of annotation tools can be tested even with scarce resources. Another goal is to provide some exemplary proof that existing tools suffer from considerable usability problems, which di-

[2]Nielsen's ten heuristics (accompanied by short, explanatory decriptions) are also freely available online: http://www.useit.com/papers/heuristic/heuristic_list.html

rectly influence the benefit-to-cost ratio of annotation projects (Dandapat et al., 2009). A third goal is to collect typical usability problems from the annotation domain, which can serve as a starting point to generate a collection of best practices and usability recommendations for the design of annotation tools.

### 3.1 Evaluation design

This subsection describes how the HW has been adopted to evaluate annotation tools.

**Evaluators and prearrangements** — For the evaluation of three exemplary annotation tools we chose three evaluators, with each of them testing each tool. One of the three evaluators was a double-expert[3], i.e. the evaluator is not only experienced in usability-testing, but also has experience in linguistic annotation and the use of annotation tools. The other two evaluators are usability experts, with a basic background in linguistic annotation. The double-expert thus had the additional function of making the usability experts aware of domain- and user-specific problems and requirements (cf. Reidsma et al., 2004). A brief introductory text, which contained the essential contextual information, was provided for the other evaluators before they conducted the actual tests. Additionally, the double-expert could be addressed during the first phase (CW) if any domain-specific problems kept the evaluators from solving their tasks. The tasks were designed by the double-expert and pretested by two additional test persons before the actual HW session. Although the tasks were slightly modified for each of the three tested tools, they included the following basic constituents:

(I) Import a text document into the tool

(II) Create an annotation scheme with two annotation layers, one for parts of speech, and one for phrases

(III) Create some basic tags in each of the created annotation layers

(IV) Annotate the first sentence of the imported text

(V) Delete an annotation

[3]Note: the double-expert is also the author of this paper.

**Limitations of this study** — Further requirements for annotation tools, like e.g. the search and querying for annotations within the tool, or the export of annotated data for further processing, have not been studied in this evaluation, as the tasks would have become to complex for a HW session. For means of feasibility we did not consider the special needs of multi-user annotation scenarios in this evaluation study. We also simplified our test scenario by assuming that the schema designer and the actual annotator are the same person. Large annotation projects, which involve many different annotators and schema designers at different skill levels, however imply additional requirements for annotation tools. Such multi-user requirements are hard to test with expert-based evaluation approaches, but should be rather addressed by using empirical test methods (e.g. user observation or interviews).

**System exploration (CW)** — During the first phase of the evaluation the main steps and user comments were recorded as a screen capture with the corresponding audio track. The main steps and important remarks were also written down by the double-expert, who acted as a passive observer. After the evaluators had finished the first phase of the HW, the documented steps were quickly recapitulated by the observer.

**Documentation of problems (HE)** — In the second phase, the evaluators wrote down usability problems which they had discovered while solving the tasks from the first phase. During this phase, they were still allowed to use and explore the annotation tool. The evaluators used a template for problem documentation, which provides fields for the name of the problem, the severity of the problem, and the violated heuristic(s). The scale for the severity rating ranges from 1 (cosmetic problem) to 4 (usability catastrophe).

**Data analysis and clustering** — At the end of the test sessions, all usability problems were analyzed by the double-expert. The problems were aggregated if several evaluators described the same problem for one tool. The problems were also clustered into thematic categories, which emerged during the analysis of the problems, and which are described in

more detail in the results section.

### 3.2 Selection of tools

Elementary differences between the vast number of existing annotation tools can be found with respect to the type of software as well as to the modality of annotation. Software types reach from simple, proprietary stand-alone programs to complex, standardized annotation and text processing frameworks. Tools also differ in the modality of annotation (images, spoken or written text, audio or video files). We chose to evaluate three freely available tools for the annotation of written texts. The selected tools represent different software types and showed quite different implementation approaches in earlier pretests (Burghardt and Wolff, 2009).

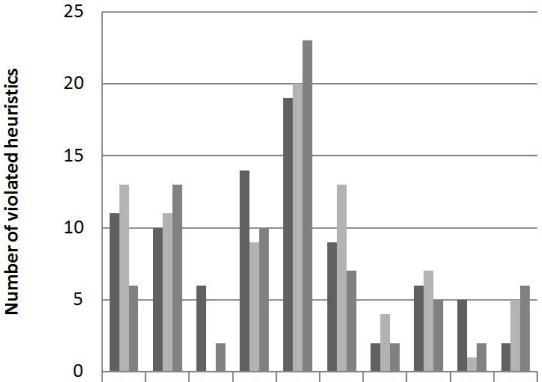
The first subject of evaluation was GATE[4] (*General Architecture for Text Engineering*), a widely used text annotation framework, which has been developed since 1997. GATE was last updated in 02/2012[5] and claims to have around 35.000 downloads p.a. (GATE, 2009). GATE is actually more than just an annotation tool, as it allows to integrate many automatic processing modules. However, for this evaluation, only the manual annotation features were tested and judged. Furthermore, we decided to evaluate MMAX2[6] (*Multi-Modal Annotation in XML*) and UAM[7] (*Universidad Autonoma de Madrid*) CorpusTool. Both tools are stand-alone annotation tools and therefore cannot be extended as easily as the GATE framework, but both implement interesting annotation features in very distinct and unique ways. Although the last update for MMAX2 dates back to 07/2010, and the number of downloads is at a moderate 4.700, we chose the tool, as it occurs frequently in literature and annotation projects. UAM CorpusTool was updated in 12/2011, and so far has 10.200 downloads[8].

## 4 Evaluation results

This section describes the results of the HW. The first part views the results with focus on the violated heuristics, and the second part focuses on more generic problem categories, which will be discussed in more detail in the next chapter.

### 4.1 Heuristic violations

There seems to be a trend toward the violation of H5 in each of the tools (cf. Figure 1), indicating that *error prevention* is a usability problem category that should be considered by annotation tool developers with particular attention. There are also numerous problems which violate H1 (*visibility of system status*), H2 (*match between system and the real world*), H4 (*consistency and standards*) and H6 (*recognition rather than recall*), and fewer records for the violation of H8 (*aesthetic and minimalistic design*) and H10 (*help and documentation*). In general, none of the tools does exceptionally well or bad with regard to these heuristics when compared to each other. At the same time, H3 (*user control and freedom*), H7 (*flexibility and efficiency of use*) and H9 (*help users recognize, diagnose, and recover from errors*) on average are not violated very often. This implies that the three evaluated tools contain many positive examples for implementing features which fall into the described heuristic categories.




| | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|
| GATE | 11 | 10 | 6 | 14 | 19 | 9 | 2 | 6 | 5 | 2 |
| MMAX2 | 13 | 11 | 0 | 9 | 20 | 13 | 4 | 7 | 1 | 5 |
| UAM CT | 6 | 13 | 2 | 10 | 23 | 7 | 2 | 5 | 2 | 6 |

Figure 1: Number of violated heuristics per tool.

Strikingly positive or negative counts of violated heuristics for individual tools will not be discussed in detail here, but are rather captured in the recommendations chapter. Nevertheless, the specific numbers display that there are tool-specific strengths and

[4] http://gate.ac.uk/

[5] Note: the evaluation was conducted with GATE 6.1.

[6] http://mmax2.sourceforge.net/

[7] http://www.wagsoft.com/CorpusTool/

[8] Both, MMAX2's and UAM CorpusTool's download numbers describe the state of February 2012.

weaknesses, which consequently means that recommendations and best practices will have to be gathered from different tools, and that none of the tested tools can without further ado be used as the *gold standard* for a perfectly usable annotation tool.

### 4.2 Problem counts and categories

The test results of the three evaluators for the annotation tools GATE, MMAX2 and UAM CorpusTool reveal a total of 143 usability problems, of which 81 can be identified as unique usability problems. The number of unique problems per tool is quite balanced, with 23 problems for MMAX2, and 29 problems for both GATE and UAM CorpusTool (cf. Table 3). The counts for unique problems together

| Tool | All problems | Unique problems | Average severity |
|---|---|---|---|
| GATE | 51 | 29 | 2.8 |
| MMAX2 | 41 | 23 | 2.9 |
| UAM CT | 51 | 29 | 2.8 |

Table 3: Number of identified usability problems per tool.

with the average problem severity of each tool show that neither of the tools outperforms the others with regard to usability. Although the average severity (scale: 1.0 - 4.0) of the problems found for each tool is not very meaningful by itself, the values (2.8 - 2.9) indicate that the majority of problems are more than just cosmetic problems or nice to have features, but rather serious issues that need to be addressed by tool developers.

By looking at the identified problems in more detail, it becomes obvious that most of them are very tool specific, which proves the previous claim that different tools have individual positive and negative features. During the process of sorting and aggregating the identified usability problems to meaningful clusters, two main categories with a total of seven subcategories emerged. The first main category can be subsumed as "general usability problems", i.e. problems in this category are not specifically related to the field of annotation tools, but could be traced in any other kind of software. The second category contains problems which are closely connected to the field of linguistic annotation.

### 4.3 General usability problems

The evaluation revealed a total of 30 general usability problems, which can be further distinguished as belonging to one of the following two subcategories (cf. Table 4):

| Cat. | Description | G | M | U | Total |
|---|---|---|---|---|---|
| **A** | Feedback and user guidance, error messages | 2 | 6 | 7 | 15 |
| **B** | UI elements and design | 4 | 3 | 8 | 15 |

Table 4: Number of general usability problems per tool (G=GATE, M=MMAX2, U=UAM CorpusTool).

Typical examples for such problems reach from cryptic error messages or unclear system prompts (category A) to badly designed buttons and menus (category B). As the treatment of such general problems is extensively described in numerous guidelines and best practice collections (cf. e.g. Johnson, 2007; Apple, 1992), these problems and their solutions will not be further discussed in this paper.

### 4.4 Domain-specific annotation usability problems

The second main category contains a total of 51 domain-specific annotation usability problems, which are aggregated to form another five subcategories (cf. Table 5).

| Cat. | Description | G | M | U | Total |
|---|---|---|---|---|---|
| **C** | Wording and metaphors | 4 | 1 | 2 | 7 |
| **D** | Import / edit primary data | 4 | 2 | 3 | 9 |
| **E** | Import / create / edit annotation scheme | 7 | 5 | 5 | 17 |
| **F** | Apply / edit / delete annotations | 6 | 3 | 2 | 11 |
| **G** | Visualize annotations | 2 | 3 | 2 | 7 |

Table 5: Number of domain-specific annotation usability problems per tool (G=GATE, M=MMAX2, U=UAM CorpusTool).

The problems in these subcategories are very interesting for tool designers, as they are closely connected to the specific domain of annotation tools. They are summed up as design recommendations in the next chapter.

## 5 Design recommendations for usable annotation tools

This section subsumes the insights gathered from positively evaluated tool features and the lessons learned from problematic features in the form of general design recommendations for annotation tools. These recommendations provide solutions for the most severe usability problems found in our evaluation study.

### 5.1 Wording and metaphors

The wording and the use of metaphors (category C) within an annotation tool are crucial for the basic understanding, the learnability and the memorability of the tool. Most problems occurred when the wording or the metaphors for basic functions deviated from conventions established by similar kinds of software, like e.g. text processing software. The wording for more domain-specific functions often seems to be very technical or theory-driven, i.e. it is not easily understood by the "plain annotator" (Reidsma et al., 2004).

**R1** Do not invent new metaphors for fundamental interaction paradigms that are known from numerous other tools, but rather stick to conventionalized wording for basic actions like e.g. importing or saving a file

**R2** Refrain from using technical wording, although it might seem obvious from a developer's point of view, but rather try to rephrase technical concepts in the domain-specific language

**R3** Make sure that metaphors are understood by users from the domain of linguistic annotation; if using a set of metaphors, make sure they are consistent and easy to differentiate

**R4** The help function should use wording that describes a problem from the user's perspective

### 5.2 Primary data

In order to import a document (category D) into an annotation tool, the user usually has to set all kinds of importing and preprocessing parameters. In many cases, the plain annotator is unable to cope with all these settings and options, besides he does not realize which effects the settings will have on the later annotation process. Another potential problem with imported text occurs with the possibility of editing the primary data.

**R5** Guide the user through the import process and make clear which parameters have to be set by providing default values and a list of options rather than free text fields

**R6** Automatize preprocessing parameters as far as possible and provide standard users with meaningful default values; offer optional advanced settings for more experienced users

**R7** Provide a preview of the imported text, but make sure the user realizes it is only a preview and not the actual document

**R8** Allow users to optionally customize and style the appearance of the primary text (color, size, fonts, etc.)

**R9** Provide an adequate visual separation of primary data and annotation base markers

**R10** Provide a mechanism to import and organize multiple documents within an annotation project (basic corpus management features)

**R11** Make sure that the primary text cannot be edited accidentally; also make sure to inform the user about possible consequences of changes in the primary text

### 5.3 Annotation scheme

Before a user can start to annotate, he needs to be able to import or define a new annotation scheme (category E). The definition and editing of annotation schemes is realized very differently in the three tools, each with specific problems.

**R12** Allow the import of existing schemes and make clear which formal requirements will have to be met

**R13** Allow the creation and editing of an annotation scheme from within the tool; hide technical details by providing a graphical scheme-editor and offer an optional XML-mode for advanced users

**R14** Make clear which annotation scheme is associated with the imported text

**R15** For most users, the creation of an annotation layer, which has the function of a container, and the creation of tags for this layer, are closely connected:

provide a mechanism that does not separate the creation of a layer and the creation of the actual tags; at the same time, allow to edit the layer as a whole (delete, rename, change order, etc.) but also allow to edit individual tags on a layer

**R16** Provide an easy mechanism to move tags from one layer to another

**R17** As in many annotation projects the scheme gradually evolves with the actual annotation process allow the ad hoc modification of the scheme; make sure the user is aware of potential inconsistencies and provide a basic validation mechanism

### 5.4 Annotation process

In order to apply an annotation (category F), some user-defined unit of the original text has to be selected via mouse or keyboard, functioning as the "base of the annotation" (Fogli et al., 2004). Depending on whether the annotation base is a single word, or some specific phrase in a complex syntactic construction, the selection process itself can be fairly challenging for the human annotator already. Applying or deleting an annotation to or from a selected text-unit bears the most problem potential for interaction design. The interaction becomes even more demanding when multi-level annotations have to be applied, i.e. an annotation base is annotated with multiple, parallel annotations.

**R18** Provide conventionalized interaction mechanisms which are familiar from existing text editors such as single click, double click and click-drag-release

**R19** Provide an option for automatic segmenting tools such as tokenizers or sentence splitters; also allow for easy overwriting of those automatically generated segments if necessary

**R20** Allow easy modification (expand or shrink the range) and deletion of existing annotation bases

**R21** Display the annotation scheme of a specific layer of annotation at any time in order to simplify the application of the appropriate annotation

**R22** Provide a quick and easy annotation mechanism, with a minimum number of steps (=mouse-clicks / key-strokes): select an annotation base (step 1), select an appropriate annotation from the scheme (step 2), apply the annotation (step 3)

**R23** Provide an easy mechanism to select tags from different annotation layers

### 5.5 Annotation visualization

The recommendations for the last problem category are concerned with the adequate visualization of the annotated data (category G). The main challenge here is to integrate the annotations into the primary text in a way the user can distinct not only different annotations from the primary text, but also parallel annotations from each other.

**R24** Display an annotation when clicking on or hovering over an annotated text-unit

**R25** Provide filtering of visible annotations by single tags (e.g. show all nouns) and by the whole annotation layer (e.g. hide all part of speech annotations)

**R26** Allow the user to customize and style his annotation and the annotation base by using different colors or markers

**R27** Provide an adequate visualization of parallel annotations for one annotation base, e.g. by using the layer- or stack-metaphor

**R28** Provide an optional XML-view of the annotated data for advanced users

## 6 Outlook and future work

While human-computer interaction has been and still is the subject of extensive research, the subgenre of *humanist-computer interaction* has been treated with significantly less attention. Fortunately, usability is increasingly perceived as a key factor in the entire corpus creation process (Santos and Frankenberg-Garcia, 2007), which besides annotation includes the digitization of primary data and the querying and visualization of the annotated data (Culy and Lyding, 2009).

The recommendations derived from the usability evaluation of three existing annotation tools may serve as a starting point for subsequent studies which point toward a more structured and validated set of usability patterns for the design of annotation tools[9]. Such a collection of patterns (Borchers, 2001) can help tool developers to systematically engineer usability for future tools, or to refactor the usability (Garrido et al., 2011) of existing tools.

[9]The evaluation study described in this paper accompanies an ongoing dissertation project on usability patterns for annotation tools.

## Acknowledgments

I would like to thank Isabella Hastreiter and Florian Meier for taking part in the HW study, and Tim Schneidermeier and Prof. Christian Wolff for feedback and helpful advice throughout the project.

## References

Apple Computer. 1992. Macintosh human interface guidelines. Addison-Wesley.

Carol M. Barnum. 2011. Usability Testing Essentials: Ready, Set...Test . Morgan Kaufmann Publishers.

Jan Borchers. 2001. A pattern approach to interaction design. Wiley & Sons.

Manuel Burghardt and Christian Wolff. 2009. Werkzeuge zur Annotation diachroner Korpora. In: *Proc. GSCL-Symposium Sprachtechnologie und eHumanities*, 21–31.

Chris Culy and Verena Lyding. 2009. Corpus clouds - facilitating text analysis by means of visualizations. In: *LTC'09 Proceedings of the 4th conference on Human language technology: challenges for computer science and linguistics*, Springer-Verlag, 351–360.

Sandipan Dandapat, Priyanka Biswas, Monojit Choudhury, and Kalika Bali 2009. Complex linguistic annotation - no easy way out! A case from Bangla and Hindi POS labeling tasks. In: *Proceedings of the Third Linguistic Annotation Workshop, Morristown, NJ*, 10–18.

Stefanie Dipper, Michael Götze, and Manfred Stede. 2004. Simple annotation tools for complex annotation tasks: an evaluation. In: *Proceedings of the LREC Workshop on XML-based Richly Annotated Corpora*, 54–62.

Gülsen Eryigit. 2007. ITU treebank annotation tool. In: *Proceedings of the First Linguistic Annotation Workshop, Prague*, 117–120.

Daniela Fogli, Giuseppe Fresta, and Piero Mussio. 2004. On electronic annotation and its implementation. In: *Proceedings of the working conference on Advanced visual interfaces - AVI '04*, 98–102.

Alejandra Garrido, Gustavo Rossi, and Damiano Distante. 2011. Refactoring for usability in web applications. In: *IEEE Software* vol. 28, 60–67.

GATE. 2009. GATE online brochure http://gate.ac.uk/sale/gate-flyer/2009/gate-flyer-4-page.pdf, accessed in February 2012.

ISO 9241-11. 1999. Ergonomic requirements for office work with visual display terminals – Part 11: Guidance on usability. ISO.

ISO 9241-210. 2010. Ergonomics of human-system interaction – Part 210: human-centred design process for interactive systems. ISO.

Jeff Johnson. 2007. GUI Bloopers 2.0: common user interface design don'ts and dos. Morgan Kaufmann Publishers.

Claire-Marie Karat, Robert Campbell, and Tarra Fiegel. 1992. Comparison of empirical testing and walkthrough methods in user interface evaluation. In: *CHI '92 Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, 397–404.

Jakob Nielsen. 1992. Finding usability problems through heuristic evaluation. In: *Proceedings of the ACM CHI'92 Conference*, 373–380.

Jakob Nielsen. 1993. Usability Engineering. Morgan Kaufmann Publishers.

Jakob Nielsen. 1994. Heuristic evaluation. In: Jakob Nielsen and Robert Mack (eds.): *Usability Inspection Methods.* John Wiley & Sons, 25–62

Dennis Reidsma, Natasa Jovanovic, and Dennis Hofs. 2004. Designing annotation tools based on properties of annotation problems. *Report for the Centre for Telematics and Information Technology. University of Twente: Dept. of Computer Science, HMI Group*.

Mary Beth Rosson and John M. Carroll. 2002. Usability Engineering. Scenario-based development of human-computer interaction. Morgan Kaufmann Publishers.

Diana Santos and Ana Frankenberg-Garcia. 2007. The corpus, its users and their needs: a user-oriented evaluation of COMPARA. In: *International Journal of Corpus Linguistics*, 12(3), 335–374.

Andrew Sears. 1997. Heuristic walkthroughs: finding the problems without the noise. In: *International Journal of Human-Computer Interaction*, 9(3), 213–234.

Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. 1994. The cognitive walkthrough method: a practitioner's guide. In: Jakob Nielsen and Robert Mack (eds.): *Usability Inspection Methods.* John Wiley & Sons, 105–140.