Journal of the Chicago Colloquium on Digital Humanities and Computer Science
Volume 1 Number 3 (2011)

Usability of Generic Software in e-Research Infrastructures

Patrick Harms, Göttingen State and University Library, University of Göttingen
Jens Grabowski, Institute of Computer Science, University of Göttingen

Abstract

In e-Research Infrastructures (eRIs), software is used in diverse application contexts. To support this software is often implemented generically. The usability of software is strongly context dependent. Therefore, the use of generic software in different application contexts results in varying degress of usability depending on the concrete usage scenario. This paper focuses on the challenges of implementing usability-oriented generic software. First, we provide an introduction to generic software in the context of eRIs. Next, we offer an overview of usability and appropriate considerations in the software development process. Based on this, we demonstrate discrepancies between good usability and the application of generic software in distinct contexts. Finally, we provide a first architectural concept to address the identified challenges.

1. Introduction

e-Research Infrastructures (eRIs) provide new scientific possibilities. Having formerly been limited to technically oriented sciences, their application nowadays also spreads into other, less technical research areas. An example is the computer-aided analysis of large text corpora in the humanities. Therefore, eRIs are gaining more and more importance.

The software in eRIs is often implemented in a generic fashion. The goal is to provide flexibility and extensibility to make the software applicable in different research areas. But in our experience eRI software is often criticized by its users. The software does not match their expectations, e.g., regarding functionality or understandability. One reason for this is the bad *usability* provided by generic eRI software. Because good usability is always related to a specific application context and generic software is utilized in distinct contexts, the usability of generic software varies between different usage scenarios.

The paper is structured as follows. In section 2, we start by determining types of software in eRIs regarding their generic applicability. In section 3, we provide an overview of usability and related concepts. Based on this, we discuss the extent of the usability of generic software that is achievable using standard methods for usability engineering in section 4. In section 5, we then describe a concept for improving the usability of generic software through the generation of application-context-specific user interfaces. We conclude with an outlook on future research in this area in section 6.

Note: For larger, higher quality versions of the figures reproduced here, please refer to the *Supplementary Data* section accompanying this article online at http://jdhcs.uchicago.edu

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

2. Generic Software in e-Research Infrastructures

The term *eRI*, also referred to as e-infrastructures or cyberinfrastructures, includes any information and communication technology utilized to conduct research.[1, 2, 3] Researchers use these systems indirectly through software interfaces. In the back-end, the software utilizes computer hardware, networks, and other technologies. An example of such software is TextGrid.

2.1. TextGrid - A Software in e-Research Infrastructures

TextGrid[4] was developed as part of the German D-Grid initiative.[5] It is a virtual research environment for the arts and humanities, in which researchers can store, edit, process, and publish their research data. The intended user groups of TextGrid include researchers from literature, linguistics, musicology, and art history.

The data management facilities of TextGrid encompass file oriented and Grid-based data storage as well as XML-based metadata and data relationship storage. Through metadata and full text search functionalities, TextGrid provides efficient data discovery and retrieval. Several editors and web-services support manual data editing and automated data processing. TextGrid also includes tools for user and access rights management. Further details can be found on TextGrid's website under http://www.textgrid.de.[6]

The different functionalities of TextGrid are generically applicable to different extents. Examples of fully generically applicable functionalities are the data management facilities. TextGrid allows researchers to store and manage any kind of data and file format. The files' metadata include a basic set of information elements, such as identifiers, file names, authors, and content-types. The metadata structures can be extended with research project specific elements. The data management facilities can be used by any intended user group of TextGrid.

Less generically applicable functionality is provided by the editors that are delivered with TextGrid. An editor is an embedded tool that is able to display files of one or more specific types. It allows users to edit the file contents and provides assistance to ensure consistency in the file format. Examples of such editors include TextGrid's text editor and its XML editor. The least generically applicable editor in TextGrid is the music sheet editor for musicologists. This editor only supports

---

[1] Ralph Schroeder, "e-Research Infrastructures and Open Science: Towards a New System of Knowledge Production?," *Prometheus* 25, no. 1 (2007): 8, accessed June 1, 2011, doi: 10.1.1.115.6926.

[2] Community Research and Development Information Service, "e-Infrastructure," *European Commission*, accessed June 1, 2011, http://cordis.europa.eu/fp7/ict/e-infrastructure/home_en.html.

[3] National Science Foundation, "Cyberinfrastructure Vision for 21st Century Discovery," *National Science Foundation*, accessed June 1, 2011, http://www.nsf.gov/pubs/2007/nsf0728/nsf0728_2.pdf.

[4] TextGrid, "TextGrid," *Georg-August-Universitaet Goettingen*, accessed June 1, 2011, http://www.textgrid.de.

[5] D-Grid, "D-Grid," *D-Grid GmbH*, accessed June 1, 2011, http://www.d-grid.de.

[6] TextGrid, "TextGrid," *Georg-August-Universitaet Goettingen*, accessed June 1, 2011, http://www.textgrid.de.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

editing specific files for storing music sheets. It is therefore only useful for TextGrid users that work with music sheets, such as musicologists.
There are further examples of TextGrid functionalities that provide different levels of generic applicability. For a better distinction, TextGrid's functionalities can be classified into two different kinds of software regarding generic applicability:

- fully generic software that is useful for any intended user group, and
- less generic software that is useful only for some user groups.

This distinction is shown in Figure 1. The ellipses represent sets of functionalities. The grey ellipse includes all functionality provided by TextGrid. The white sets show the functionality useful for two specific user groups, such as musicologists and linguists. The overlapping region of the white sets represents all TextGrid functionality that is generically applicable for both user groups. This includes, e.g., TextGrid's data management facility. The non-overlapping parts of the white sets represent TextGrid functionality that is useful for only one of the user groups. An example is the music sheet editor for musicologists. In the next section, we provide a more abstract view on types of software to be found in eRIs.

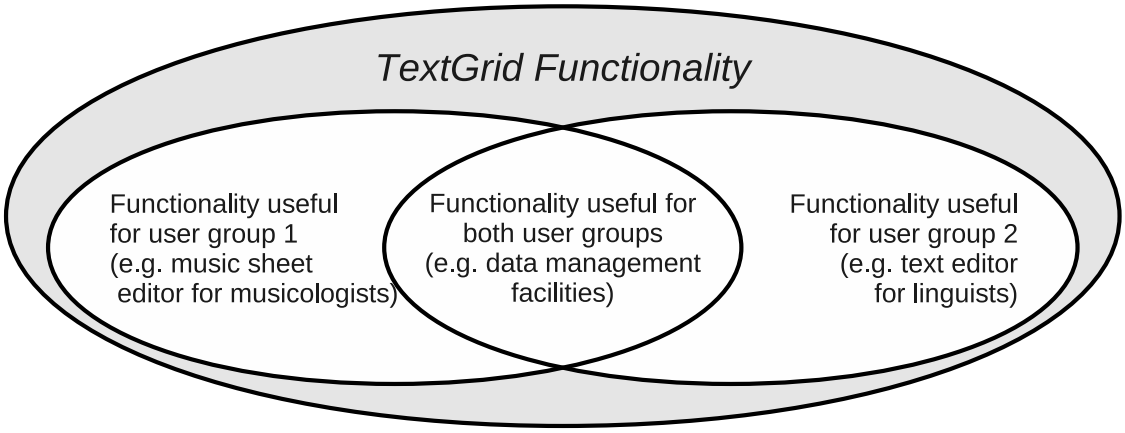

Figure 1. Generic applicability of TextGrid functionality.

2.2. Categories of Software in e-Research Infrastructures

Software that per definition belongs to eRIs is any kind of software that researchers use to conduct their research. This also includes standard software, such as e-mail clients or web browsers. We distinguish among three categories of software in eRIs:

- software not limited or dedicated to research,
- generic research software, and
- specific research software.

The first category, software not limited or dedicated to research, includes software that does not focus on research but provides general purpose functionality also needed and used outside research. Examples include offices tools, e-mail clients, operating systems, and instant messaging tools.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

Generic research software is dedicated to conducting research. It offers functionality needed for research in general or in a specific research domain. It has no focus on a specific research project. Examples include research data repositories and catalogs, as well as data processing tools and frameworks. Referring to TextGrid, this category includes TextGrid's data management facilities.

The third category, specific research software, implements research-project specific solutions. It has a strong focus on a specific research project and is usually not generically applicable for other projects. Examples include special data processing algorithms or project specific processing chains. Within TextGrid, the music sheet editor belongs to this category.

These three categories differ in their concreteness and their focus on specific research projects. The more software focuses on a specific research project, the less it can be generically used in others. This relationship is shown in Figure 2. The boxes represent the identified software categories. They are sorted depending on their generality. The arrows represent the level of generality and concreteness of the categories regarding research projects.

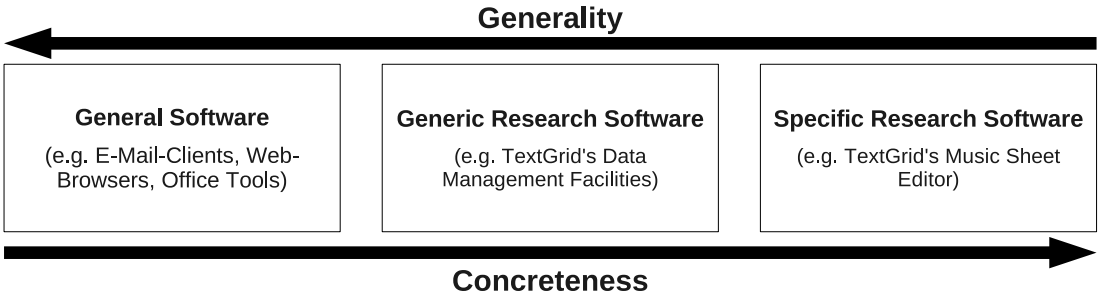

Figure 2. Categories of software in eRIs.

Specific research software can become generic research software, if its purpose and its implementation allow for reuse in other research projects. This usually accompanies adaptations and extensions to make the software more generic. As an example, TextGrid's music sheet editor can be extended to support different file formats for storing and reading music sheets, thus becoming applicable in several distinct research projects.

Specific research software often directly utilizes the facilities of generic research software. Here also TextGrid's music sheet editor is a good example. It utilizes TextGrid's data management facilities to read and store music sheet files.

2.3. General Software Categories regarding Generic Applicability

The categories of software identified in the previous section focus on research and eRIs. But in other domains similar classifications can be done. Therefore, we see the following categories for software in general:

- general software,

- domain-specific software, and
- application-specific software.

The term *domain* here refers to a bounded set of applications that belong together. Research is one example for a domain. Another domain is internet shops. For both domains, examples for each of the identified software categories are given in Table 1. As an example, a framework for internet shops is domain-specific software of the internet shop domain. The table also shows that general software is used beyond the borders of a specific domain.

	Research Domain	**Internet Shop Domain**
General Software	e.g. E-Mail-Clients, Web-Browsers, Office Tools	
Domain-Specific Software	e.g. TextGrid's Data Management Facilities	e.g. Framework for Internet Shops
Application-Specific Software	e.g. TextGrid's Music Sheet Editor	e.g. Configurations and Extensions for a Framework for Internet Shops

Table 1. Examples for software types in the domains research and internet shops.

Throughout this paper, we concentrate on the research domain. We exclude general software and focus on the other two software categories. We refer to the domain-specific software, e.g., generic research software, as *generic software* and to application-specific software, e.g., specific research software, as *specific software*.

3. Usability

Usability is a quality characteristic of software.[7] It considerably influences the handling of, and the user's attitude towards a software product. It also plays an important and decisive role in the selection process between different software alternatives for the same application scenarios. Thus usability affects economic aspects of software development.

3.1. Introduction to Usability

Usability is context-sensitive.[8] This means that software that provides good usability in one application context can have bad usability in a different application context. The application context includes:

- the tasks to be executed with the software,
- the environment in which the software is used, and
- the users that fulfill tasks with the software.

[7] Werner Schweibenz and Frank Thissen, *Qualität im Web: benutzerfreundliche Webseiten durch Usability Evaluation* (Berlin: Springer, 2003): 12.

[8] ISO, *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten: Teil 11: Anforderungen an die Gebrauchstauglichkeit – Leitsätze* (Brüssel: Beuth, 1999): 5.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

The tasks to be executed with the software are user-oriented. The software supports the execution of the task. An example is the task to write a letter. The software helps the user in writing the letter, but it does not write it.
The environment of the software includes the operating system and the hardware it runs on, the rooms or buildings it is used in, as well as to the lightning, sounds, weather, or other similar conditions that influence the user during the utilization of the software.

Regarding the characteristics of users, human psychology is also closely related to the application context. Psychological aspects, such as cognitive skills, expectations, or typical human behavior, must be considered to ensure good usability. But these factors are not user group specific. Therefore, they do not belong to the concrete application context. Instead, they influence usability as a stable confounding variable.

The usability of software is assessed indirectly by utilizing quantitative and qualitative measures. Quantitative measures include effectiveness, efficiency, and error rate.[9] They are measured indirectly through the observation of people using software. The error rate, for example, is quantified by counting the number of mistakes a user makes during the usage of the software. Such mistakes are, e.g., data entries in invalid formats or mouse clicks that are not needed to fulfill a task. Qualitative measures include satisfaction and attractiveness. They are obtained using user questionnaires and interviews after a user has fulfilled specific tasks with software. In addition, they can be the result of analyses of experts.

The definitions of usability are based on the application context as influencing factor, and the measures used for its assessment. A product is usable, i.e., has a good usability, if, for a representative set of tasks to be fulfilled in a specific environment by the envisaged user group, the obtained values for qualitative and quantitative measures, such as effectiveness, efficiency, and satisfaction, match predefined, positive criteria.[10, 11, 12] If for example the software shall support the execution of a task in a defined time slot, and the measurements of the efficiency show, that the task is executable in that time slot, then the software provides a good usability regarding that aspect.

Figure 3 shows both the influences on usability caused by the application context, as well as the measures for usability assessment. It is based on ISO 9241 part 11.[13] The figure includes the subdivision of the application context into (1) the tasks to be done with the software, (2) the usage environment, and (3) the user. For each of these parts, the figure provides examples of detailed aspects to be considered. For the tasks, such aspects include the goals to be achieved with the tasks and the steps that are executed for fulfilling the tasks. Furthermore, the figure distinguishes between

---

[9] The terms *effectiveness*, *efficiency*, and *error rate* are used in both contexts: software development as well as usability. In software development they represent characteristics of software. For usability, they characterize users during the execution of tasks instead. In this paper we will use the meaning of the terms in the context of usability.

[10] ISO, *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten: Teil 11*: 3.

[11] Andreas Holzinger, "Usability Engineering Methods for Software Developers," Communications of the ACM 48, no. 1 (2005): 71, accessed June 1, 2011, http://portal.acm.org/citation.cfm?id=1039539.1039541.

[12] Florian Sarodnick and Henning Brau, *Methoden der Usability Evaluation: wissenschaftliche Grundlagen und praktische Anwendung* (Bern: Huber, 2006): 17, 135.

[13] ISO, *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten: Teil 11*: 1ff.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

qualitative and quantitative measures. For both groups it shows appropriate examples, like effectiveness for quantitative measures.

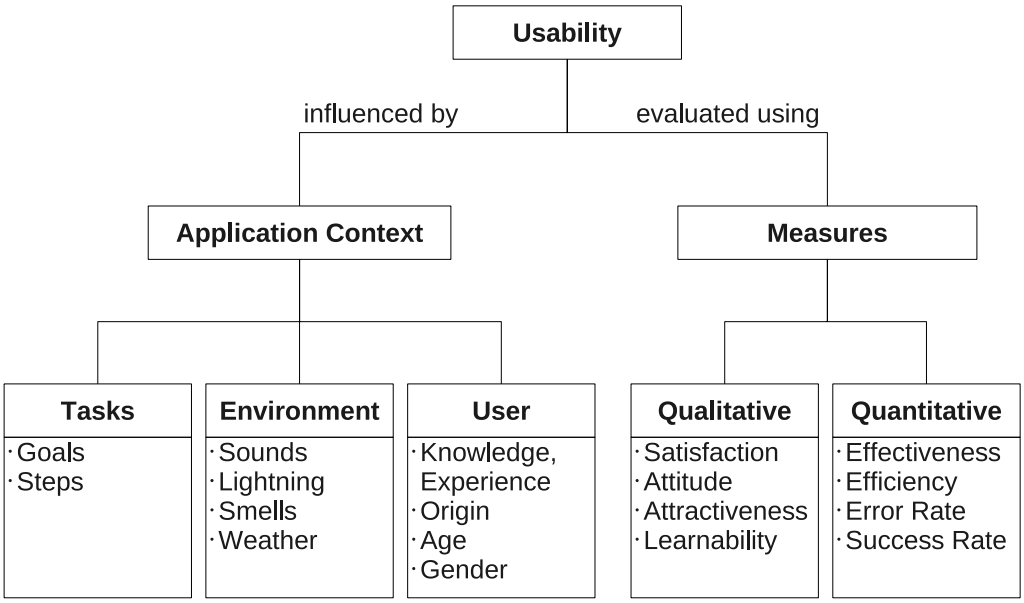

Figure 3. Influences and measures of usability.

3.2 Usability Evaluation

There are several methods for evaluating the usability of software. They can be used for obtaining values for the measures mentioned in the previous section. Usually only a subset of measures is gained by a specific method. To be more effective, the methods must consider details of the application context, such as characteristics of the intended user groups or the tasks to be executed with the software. This is usually done by analyzing the application context of the software before an evaluation takes place, and utilizing the outcome of this analysis in subsequent evaluation procedures. Therefore, the evaluation results are context-sensitive. The methods can be divided into *expert-oriented* methods, which are also known as *analytical* methods, and *user-oriented* methods,[14, 15] which are often referred to as *empirical* methods.

Expert-oriented methods specify actions to be taken and issues to be considered to analytically assess the usability of the software. For the successful application of these methods, background knowledge and method-related experience is required. Therefore, they should be conducted by experts who match these criteria. A well known example of such methods is the cognitive

[14] Schweibenz, *Qualität im Web*: 43.

[15] Holzinger, *Usability Engineering Methods*: 72.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

walkthrough.[16] This method focuses on the easy learnability of a system. Based on four guiding questions,[17] it tries to determine at every step of a system's usage if an inexperienced user would intend to do the next correct action and if this action is available and executable.

The execution of expert-oriented methods usually starts with training a group of evaluators in the application of a specific method. Afterwards, these evaluators perform the same analysis with the same method on the same software. The more evaluators perform the evaluation, the higher the validity of the results. Finally, subsequent discussions among the evaluators result in a better understanding of the existing usability problems and provide initial ideas for improvements.

User-oriented methods concentrate on evaluating the usability of software with the help of end users. In this approach, users perform several selected tasks with the software. In the meantime, the evaluators collect usage data either through manual observation or through automated recording of the interaction using appropriate equipment. In addition, the evaluators ask the users to describe their personal usage experience. After the test execution, the evaluators analyze the collected data and material. Based on this, they draw conclusions regarding the software's usability and respective points of improvement. For such tests the software can still be in a prototypical state.

For the purpose of collecting test data and material, a variety of different methods can be applied. These are classified into *active* and *passive methods*. Passive methods are, e.g., video/audio recording, log file recording, and eye-tracking. They do not require a specific behavior of the user and therefore reduce test influences to a minimum. On the other hand, active methods require the user to perform unusual activities during the evaluation. These include verbalizing thoughts (thinking aloud) or answering test related questions either using questionnaires or by conducting interviews. These methods have an increased influence on the user and therefore on the test. Their advantage is a better detection and analysis of usability problems. The selection of the method strongly depends on the specific test scenario.

3.3. Usability Engineering

The development of software can be adapted to introduce usability awareness. This is referred to as usability engineering.[18] It can be divided into the following three categories:

- application of knowledge about human behavior and psychology,
- application of guidelines and heuristics, and
- iterative user interface assessment by applying usability evaluation methods.

The application of knowledge about human behavior and psychology is a minimalist approach of usability engineering. Here software developers and architects are made aware of general cognitive

---

[16] Peter Polson et. al., "Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces," *Institute of Cognitive Science, University of Colorado, Boulder*, accessed June 1, 2011, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.3404&rep=rep1&type=pdf.

[17] Schweibenz, *Qualität im Web*: 93.

[18] Sarodnick, *Methoden der Usability Evaluation*: 19.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

and perceptional skills of humans as described by, e.g., Schweibenz,[19] Sarodnick,[20] or Norman.[21] Based on this knowledge, the software design is adapted to provide theoretically better usability. The practically resulting usability is not evaluated.

Guidelines and heuristics provide best practices at different levels of detail for developing software with good usability. They can be applied without detailed knowledge of human psychology or behavior. Guidelines and heuristics often provide general statements, such as an interface must be consistent in its representation. But they can also focus on specific application domains and consider respective requirements. Guidelines and heuristics are assembled analytically or based on experience. The resulting usability strongly depends on their interpretation. Usually there is no subsequent usability evaluation.

The most substantial usability engineering is conducted through the iterative application of usability evaluation methods which are described in the previous section. The methods can be applied at any prototypical stage of the software varying from paper-based interface drafts, via interface mock-ups, up to pre-to-final versions. Based on the outcome of the intermediate assessments, actions for improving the prototype or pre-to-final product are derived and taken.

Both expert-oriented and user-oriented methods should be applied at different stages of software development.[22] Expert-oriented methods, for example, are able to assess rudimentary interface concepts and can therefore already be conducted very early in the development. Their disadvantage is that they do not involve the end user. One can compensate for this by applying user-oriented methods. However, these are limited to later development stages as they need a minimum of interface functionality and quality.

The usability problems observed through the application of the usability evaluation methods must be solved during the subsequent steps in the software development process. It must be recognized, that some usability problems are not easily eliminated through small changes in the user interface. Instead, solving them requires adaptations on the software architectural level.[23] There has been initial research on directly designing software architecture with usability in mind. The goal is to minimize architectural changes because of usability problems that are required late in a development

[19] Schweibenz, *Qualität im Web*: 24ff.

[20] Sarodnick, *Methoden der Usability Evaluation*: 48ff.

[21] Donald A. Norman, *The Design of Everyday Things* (New York: Perseus Books, 1992): 12ff.

[22] Holzinger, *Usability Engineering Methods*: 74.

[23] Eelke Folmer and Jan Bosch, "Architecting for usability: a survey," *Journal of Systems and Software* 70, no. 1-2 (2004): 2ff, accessed June 1, 2011, http://dissertations.ub.rug.nl/FILES/faculties/science/2005/e.folmer/c2.pdf.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

process.[24, 25, 26] However, to ensure good usability, these techniques need to be combined with the application of usability evaluation techniques. Especially the expert-oriented methods should be applied in early design phases.

4. Usability and Generic Software

Because usability of software is context-sensitive, there is a theoretical discrepancy between good usability and the generic use of software in different application contexts. This also applies to generic research software. To solve this issue, the three different categories of usability engineering are at most partially helpful. The reason for this is their level of application context dependency. Best usability is always reached through methods that take the application context into account. The utilization of methods that are not, or only partially, concerned with the application context achieve a lower level of usability. Therefore, the applicability of usability engineering methods for generic software decreases with an increasing consideration of an application context.

This overall relationship is shown in Figure 4. In this figure the boxes represent the different categories of usability engineering. They are sorted based on the dependency on an application context. The arrows indicate the degree of achievable usability, the dependency on an application context, and the expected applicability for generic software. The figure also indicates an overlapping of the intensity, in which the different method categories take the application context into account.

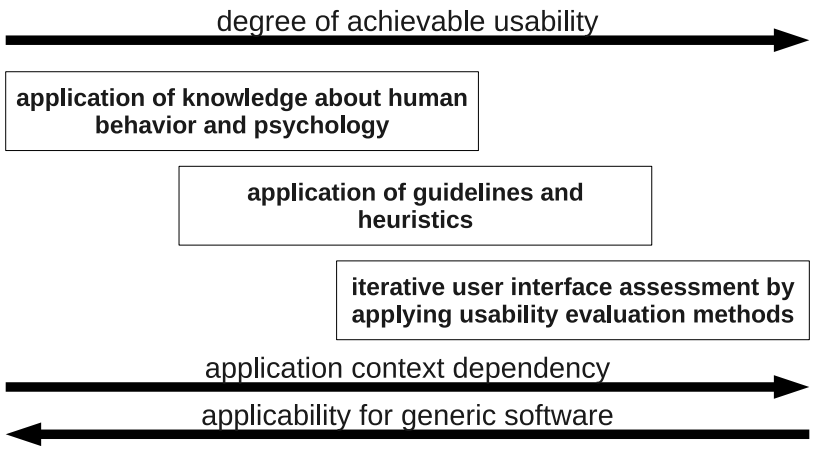

Figure 4. Applicability of usability engineering in the context of generic software.

---

[24] Jan Bosch and Natalia Juristo, "Designing software architectures for usability," *Proceedings of the 25th International Conference on Software Engineering* (2003): 1ff, accessed June 1, 2011, http://portal.acm.org/ft_gateway.cfm?id=776937&type=pdf&coll=GUIDE&dl=GUIDE&CFID=80172646&CFTOKEN=86699133.

[25] Len Bass et al., "Usability-Supporting Architectural Patterns", *International Conference on Software Engineering* (2004): 1ff, accessed June 1, 2011, http://doi.ieeecomputersociety.org/10.1109/ICSE.2004.1317502.

[26] Bonnie E. John et al., "A responsibility-based pattern language for usability-supporting architectural patterns," *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems* (2009): 1ff, accessed June 1, 2011, http://doi.acm.org/10.1145/1570433.1570437.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

The first category of usability engineering, i.e., the application of knowledge about human behavior and psychology, is most unaware of an application context in which the software is used. Therefore, the usability achieved through this method reaches only an application-context independent level. The method considers humans in general, but not, e.g., context-specific terminology. As a result, user interfaces designed through this method utilize a generic terminology but not one that is most helpful for the user group.

The application of guidelines and heuristics can have a closer relationship to the application context. This depends on the concrete heuristic or guideline chosen. In the case, that no concrete application context is envisaged, the same problems as described in the previous paragraph arise. The resulting software does not provide the usability that is achievable with a stronger focus on the application context. If a guideline or heuristic has an application context relationship, its utilization results in software with a better usability. But in this case, the usability of the software is linked to the application context. If the software is used in a different application context, it provides a worse usability. Therefore, the software is not generically usable with the same quality.

The application of usability evaluation methods has the same challenges because of their context-sensitivity. The usability of the resulting software is only good within the application contexts that were considered during the development and the application of the evaluation methods. For other application contexts the usability can be, and most probably will be, worse.

As a consequence, generic software cannot be developed with good usability for every application context using only the existing usability engineering methods. Either a universal, but unspecific, or an application-context-specific usability can be reached. It is not yet possible to implement software that is generically applicable and provides the same good usability for every application context, especially for contexts unknown during the development.

5. Generically Usable Software – A Draft Architectural Approach

A potential solution to achieve good usability of generic software is the implementation of distinct user interfaces, where each is dedicated to a specific application context. Such interfaces can provide the available functionality in a way that best fits the envisaged application context. But such an approach is economically infeasible.

Therefore, instead of their manual implementation, we propose to generate context-specific interfaces. As a result, context-specific interfaces can be made available with less effort than it would take to implement them. In this chapter we provide a concept for this generation process. We start by identifying the needs for adaptability of the access to available functionalities in order to support application context awareness of user interfaces. Then, we introduce the basics of our concept and explain them with TextGrid as a potential example. Finally, we perform a theoretical assessment of the achievable usability.

5.1. Adaptability of Access to Functionality

Section 2.1 shortly described TextGrid as an example of software in eRIs. TextGrid includes generic and specific software to provide its functionality. Considering a specific research project, only a subset of TextGrid's functionality is needed. For example, a literature project makes marginal use of the music sheet editor for musicology. Furthermore, available metadata structures of the data

management facilities are used and needed only partially. In addition to this, the research project may reuse functionality using different names for functions and data structures. Furthermore it may need tools not yet available in TextGrid.

Therefore, a research project using TextGrid selects parts of TextGrid's functionality, renames it, and develops functional additions. To be able to generate a user interface specific for such a project the generation process needs information about which functionalities are needed and how they shall be made available for the user. Therefore, the specification for an application-context-specific interface must include the following:

- specification of available generic and specific functionality,
- selection of functionality needed in a specific application context, and
- specification of application-context-aware adaptations of the access to functionality.

The specification of available functionalities will be needed only once for a software application. However, the application-context-specific selection of the functionalities, as well as the adaptation of the access to them, is needed for any generated user interface. The following section introduces a concept to do such specifications and to generate user interfaces based on them.

5.2. Generation of Application-Context-Specific User Interfaces

Most software has two basic layers: the user interface and the back-end. The user interface allows the users to work with the software. It is usually graphical or textual. In this paper we consider graphical user interfaces. The back-end implements the business logic of the software, i.e., the functionalities. The user interface provides access to these functionalities.

The generation of application-context-specific user interfaces for generic and specific software can be based on models. Three different models are needed for such an approach. They are shown in Figure 5. The first model is the functionality model. It specifies the low level functionality of the back-end, including data types and available functions. Therefore, it is the specification of the available generic and specific functionality. There are already languages that can be used for this. An example is the Web Service Description Language (WSDL). Through its integration with XML Schema it supports the definition of complex data structures. These can be combined with definitions and groupings of functions.

The second model is the user interface framework model. It describes the basic elements of the user interface as well as their arrangement. It aims at giving each of the generated user interfaces the same basic structure. It defines classes of interface components on a higher level. This means that the model identifies and names groups of related interface elements that together form a bounded interface component. One example of such a component are the menus of a window on the screen. Menus are usually located on the top left of a window. They are subdivided into several submenus, each identified by a label. Typical labels are "File", "Edit", and "Help". When clicking on the labels, the elements of the submenus appear on the screen. These elements are buttons that provide access to related functionalities of the back-end. When clicking on a button, the functionality is executed. The focus of the user interface framework model is to define such basic elements and their locations on the screen. But, the model is not too detailed. It leaves space for refinement. Typical refinements for a menu are specific labels to be used, as well as the concrete buttons available in a submenu.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

The third model, the user interface model, is a refinement of the user interface framework model. It provides more details about the high level interface components identified by the user interface framework model. It also links the functionality defined by the functionality model to specific elements of the user interface model. In the example of the menu, the user interface model defines the concrete labels of the menu and the buttons available in the submenus. It further links the buttons to functions of the functionality model so that a click on a specific button executes a specific functionality of the functional model. It does not need to map all functionality of the functionality model to user interface elements. Therefore, it is used to specify the functionality selection, as well as the application-context-aware adaptations requested in the previous section. The three models are further described in the next section using TextGrid as an example.

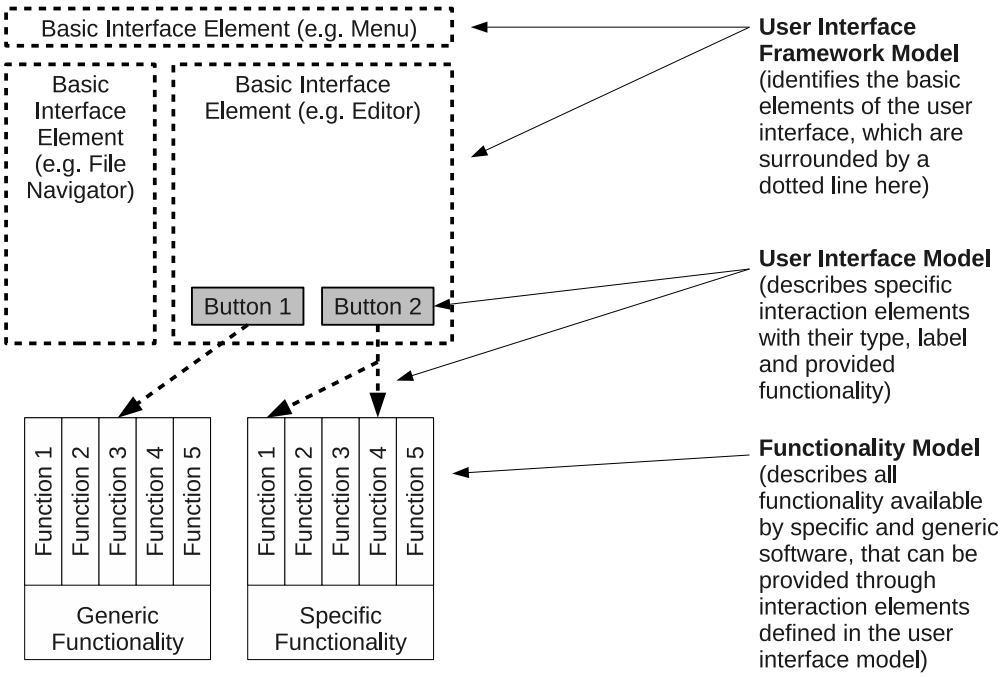

Figure 5. Models for user interface generation.

5.3. Model Examples for TextGrid

TextGrid is divided into two major components:

- the TextGrid-Laboratory as the graphical user interface (GUI): it provides the user with an integrated environment to access all TextGrid functionalities, and
- the TextGrid-Repository as the back-end: in addition to others, it stores and archives the research data and enforces user management, including access rights.

The TextGrid-Repository can be accessed mainly through web services described in WSDL. The web services provide functions, such as creating, reading, updating, and deleting files. The TextGrid-Laboratory calls these functions on behalf of the user. The user is unaware of these calls as the

TextGrid-Laboratory encapsulates them. For TextGrid, a functionality model as introduced in section 5.2 includes at least all functionality provided by the TextGrid-Repository.

The TextGrid-Laboratory is structured into several distinct groups of interaction elements. Each group provides access to functions that logically belong together. An example is the group for data management. It provides access to the files stored in TextGrid. For this, it displays the contents of TextGrid in the form of a file tree view similar to common file system browsers. Through buttons, context menus, and other interface components, the user gets access to functions such as storing, reading, updating, and deleting files. A user interface framework model for TextGrid, which describes basic structures of user interfaces as shown in section 5.2, therefore includes a definition for the data management group of interaction elements of the TextGrid-Laboratory. Furthermore, it already specifies several elements that will be part of the data management, such as a view for displaying the content of TextGrid as well as the location of buttons and a context menu. It does not define which concrete interaction elements, such as button types with labels or tree views, are used. This is done by the user interface model.

As a refinement for the user interface framework model, the user interface model for a specific application context of TextGrid defines the concrete interface elements needed in that context. For the example of data management, it specifies the specific representation of the contents in TextGrid, i.e., a file tree. It also defines the specific buttons, labels, and elements in the context menu of that view to create, read, update, and delete files. Furthermore, the user interface model links the defined buttons and context menu entries to functions provided by the TextGrid-Repository. In case of a button for creating a new file, it links this button to the appropriate function of the web services of the TextGrid-Repository. If this function requires parameters, the user interface model defines where the parameters can be retrieved. Sources for parameter values can be other interaction elements in the user interface, specific dialogues to request the parameters from the user, or constant values. The functionality model can also define combinations of functionalities of the functionality model to appear as one functionality, such as one button, in the user interface.

5.4. User Interface Generation

Based on the three different models, user interfaces can be generated through model transformation. The user interface framework model is in this case the basis for an overall structuring of the user interface. This structuring can be filled with detailed interaction elements defined by the user interface model. Because of the linking of these interaction elements to functionalities in the functionality model, the user interface can be linked to the appropriate back-end software providing these functionalities.

The result of the model transformation can be any kind of model that is capable of fully specifying a user interface. An example is Java source code utilizing classes provided by Java Swing. Such a model is platform-specific and therefore restrictive. There are more abstract, platform-independent models that provide more flexibility for subsequent use of the resulting model. An example is the DiaMODL language described by Trætteberg.[27]

[27] Hallvard Trætteberg, "Model-based User Interface Design," *Norwegian University of Science and Technology*, accessed June 1, 2011, http://www.idi.ntnu.no/~hal/_media/research/thesis.pdf?id=research%3Athesis&cache=cache.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

The basic concept for the generation of the user interface is illustrated in Figure 6. A model transformation routine, which is based on transformation rules, takes the user interface framework model, the user interface model, and the functionality model as input and produces an executable user interface model. We call this executable, as it should be in a form that is either directly or indirectly presented to the user. Directly means that the model is in the form of executable source code. Indirectly means that the model is in a form that is interpreted by another entity, e.g., by software that reads such models and renders the described user interface.

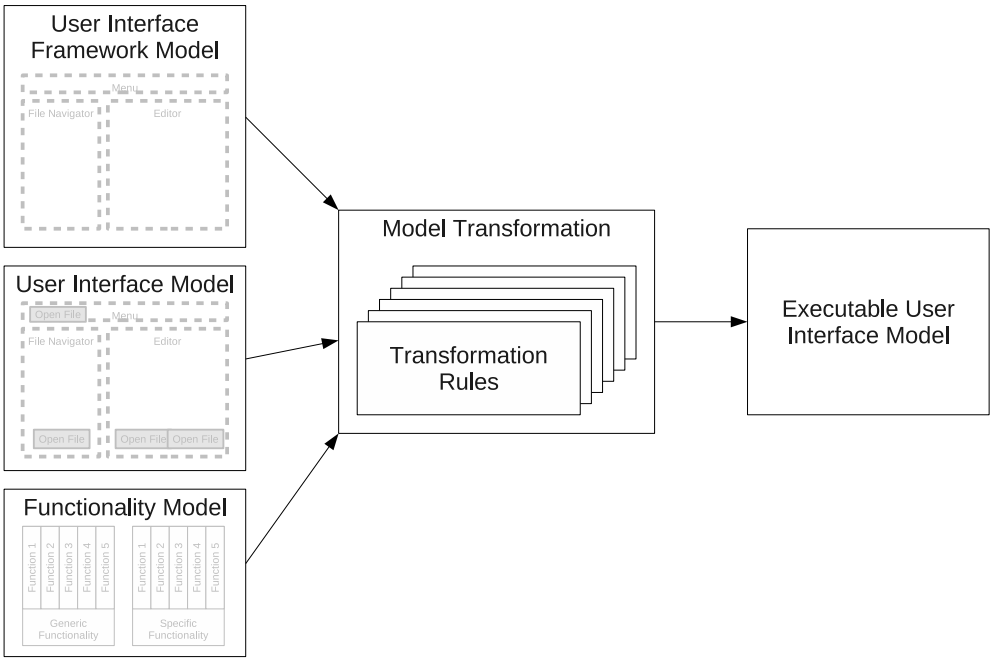

Figure 6. User interface generation concept.

The transformation rules add significant value, in that they map elements of the input models to specific elements visible in the resulting user interface. An example is a button defined in the user interface model. A button can be represented in many different variants, colors, and sizes. The transformation rules generate the specific representation and provide the visualization details. Therefore, they are an important part of the generation process.

5.5. Support for Improved Usability

Through the approach described in the previous sections, several usability-related issues can be addressed. The user interface framework model is the basis for a logical structuring and grouping of functionalities. Referring to Norman,[28] this is one important aspect necessary to make interfaces easy to use. Furthermore, the provided functionalities can be combined and renamed to better match the tasks of the given application context and the knowledge and expectations of the users. It is also possible to hide system complexity that results from generic implementations (such as

[28] Norman, *The Design of Everyday Things*: 174.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

complex metadata structures) through functionality selection. This allows for simplification and a focusing on relevant elements for a specific application context and user group.

Another positive aspect is that user interfaces generated based on models tend to provide better usability than programmed user interfaces.[29] One reason for this is the consistent representation of interface elements that results from the application of the transformation rules. But the transformation process may also produce interfaces with bad usability. An example is the generation of interaction elements that are out of sight for the user. This must be considered and solved in the implementation of the transformation rules as well as in the specification of the input models.

5.6. Process Model for the Development of Generic Software

The described approach does not always generate usable interfaces. Instead, the provided flexibility can also decrease the usability. For example, it is still possible to select counterproductive terminology for the naming of user interface elements in the user interface model. Therefore, it is necessary to combine the proposed approach with usability engineering methods as described in section 3.3. Through iterative assessment of the generated user interfaces, usability defects can be observed and linked to elements in the underlying models or in the transformation rules.

An implementation of this approach should also allow the generation of interface prototypes without underlying functionality. This can be done by ignoring mappings of functionalities in the user interface model to functionalities in the functionality model. As a result, non-functional, but graphically working, user interface prototypes can be created already early in a software development process. This also allows for early usability evaluations.

6. Conclusion and Outlook

Generic software is applied in different application contexts. We showed that this is in conflict with the context-sensitivity of usability. Therefore, generic software cannot provide the same level of usability for every application context as it is possible with specific software.

To address this issue, we provided a concept for generating application-context-specific user interfaces based on models and model transformations. The models describe generic and specific functionality of software, as well as basic and specific elements of user interfaces that are needed within a specific application context. Through model transformation based on transformation rules, user interfaces can be generated. These are expected to provide better usability in comparison to generic user interfaces, as they are application-context-specific. This is ensured through complexity reduction and focusing on elements that are relevant for that specific context.

Further research needs to be conducted to validate our approach. For this, the usability of existing generic software, such as TextGrid, needs to be evaluated. Then, an equivalent for the same software should be provided through the application of our proposed approach. The resulting user interface can then be evaluated again to check whether the measured usability shows an improvement.

[29] Silvia Abrahão et al., "Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool," in *Maturing Usability*, eds. John Karat et al. (London: Springer, 2008): 27.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago

During this scenario, we expect to find best practices for our approach that should be followed to achieve the best possible usability. This also includes guidelines for the definition of user interface framework models, user interface models, and functionality models. Furthermore, we will provide best practices for the needed transformation rules. As we plan to implement the approach, we will also be able to provide an assessment of the utilized tools and languages for their applicability in our approach.

7. Acknowledgments

I want to thank Prof. Jens Grabowski and his software engineering research group for reviewing and proof-reading this paper. Furthermore, I would like to thank all colleagues of the WisNetGrid project and the TextGrid project. They provided much of the necessary information and knowledge about project details and kindly discussed usability related issues with me. In particular, I would also like to thank Heike Neuroth, whose efforts have made it possible to for me to effectively combine my fulltime employment with my graduate studies.

Bibliography

Abrahão, Silvia et al. "Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool." In *Maturing Usability*, edited by John Karat et al., 3-32. London: Springer, 2008.

Bass, Len et al. "Usability-Supporting Architectural Patterns." *International Conference on Software Engineering* (2004): 716-717. Accessed June 1, 2011. http://doi.ieeecomputersociety.org/10.1109/ICSE.2004.1317502.

Bosch, Jan and Natalia Juristo. "Designing software architectures for usability." *Proceedings of the 25th International Conference on Software Engineering* (2003): 757-758. Accessed June 1, 2011. http://portal.acm.org/citation.cfm?id=776937&dl=ACM&coll=DL&CFID=29740826&CFTOKEN=74809744.

Community Research and Development Information Service. "e-Infrastructure." *European Commission*. Accessed June 1, 2011. http://cordis.europa.eu/fp7/ict/e-infrastructure/home_en.html.

D-Grid. "D-Grid." *D-Grid GmbH*. Accessed June 1, 2011. http://www.d-grid.de.

Folmer, Eelke and Jan Bosch. "Architecting for usability: a survey." *Journal of Systems and Software* 70, no. 1-2 (2004): 61-78. Accessed June 1, 2011. http://dissertations.ub.rug.nl/FILES/faculties/science/2005/e.folmer/c2.pdf.

Holzinger, Andreas. "Usability Engineering Methods for Software Developers." *Communications of the ACM* 48, no. 1 (2005): 71-74. Accessed June 1, 2011. http://portal.acm.org/citation.cfm?id=1039539.1039541.

ISO. *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten: Teil 11: Anforderungen an die Gebrauchstauglichkeit – Leitsätze*. Brüssel: Beuth, 1999.

John, Bonnie E. et al.. "A responsibility-based pattern language for usability-supporting architectural patterns." *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (2009): 3-12. Accessed June 1, 2011. http://doi.acm.org/10.1145/1570433.1570437.

National Science Foundation. "Cyberinfrastructure Vision for 21st Century Discovery." *National Science Foundation*. accessed June 1, 2011. http://www.nsf.gov/pubs/2007/nsf0728/nsf0728_2.pdf.

Norman, Donald A.. *The Design of Everyday Things*. New York: Perseus Books, 1992.

Sarodnick, Florian and Henning Brau. *Methoden der Usability Evaluation: wissenschaftliche Grundlagen und praktische Anwendung*. Bern: Huber, 2006.

Schroeder, Ralph. "e-Research Infrastructures and Open Science: Towards a New System of Knowledge Production?" *Prometheus* 25, no. 1 (2007): 1-17. Accessed June 1, 2011. doi: 10.1.1.115.6926.

Schweibenz, Werner and Frank Thissen. *Qualität im Web: benutzerfreundliche Webseiten durch Usability Evaluation*. Berlin: Springer, 2003.

TextGrid. "TextGrid." *Georg-August-Universitaet Goettingen*. Accessed June 1, 2011. http://www.textgrid.de.

Trætteberg, Hallvard. "Model-based User Interface Design." *Norwegian University of Science and Technology*. Accessed June 1, 2011. http://www.idi.ntnu.no/~hal/_media/research/thesis.pdf?id=research%3Athesis&cache=cache.

Source URL: http://jdhcs.uchicago.edu
Published by: The Division of the Humanities at the University of Chicago