Usability-Focused Architectural Design for Graphical User Interface Components

Stephan Bode, Matthias Riebisch
Technical University of Ilmenau
{stephan.bode, matthias.riebisch}@tu-ilmenau.de

Abstract

Although in recent years some progress in software engineering (SE) and human-computer interaction (HCI) has been made, there is still a gap between the two research areas and their methodologies. Today, from the engineering point of view, the specification and design of graphical user interfaces and their corresponding software architectural components is still a challenging task. The advanced component design methods are not yet integrated with HCI design methodologies to bridge the two fields. This paper presents a methodology, which extends a category-based software architectural design method by integrating HCI approaches. The methodology aims at a better design of Graphical User Interface (GUI) components in terms of both the architectural quality and usability. The methodology has been successfully evaluated in the development and partial reengineering of an e-commerce system.

1. Introduction

During the last years, the requirements for flexibility and usability of software systems have increased significantly, for example because of their bigger value for businesses and service. On the other hand, the systems' complexity has grown dramatically. As a consequence, the demand for software architectures and especially for architectural quality has intensified. Concerning the quality of software architectures several non-functional properties have to be considered, e.g. maintainability, efficiency, reliability, security, or usability [10]. Since the acceptance by the users is critical for the success of many business-critical software systems, usability has got more and more attention. Unfortunately, the software engineering research has not focused enough on usability.

Usability is defined as a property describing the ease-of-use of systems. In the research field of human-computer interaction (HCI) several methods for improving this property during the design process have been developed. Already in the early 1990s Nielsen and Shneiderman decomposed usability into several subgoals—*learnability, task efficiency, memorability, low error rate*, and *user satisfaction*—and gave advice for a high quality user interface design [16, 17]. Nevertheless, many existing interactive software systems fail to provide good user interfaces and the way they work is unpleasant for their users. Usability engineering as an integrated methodology becomes more and more important. In this paper, we develop contributions to usability engineering, which are related to the field of software architectural design.

Current software engineering (SE) methods consider use cases as the primary artifacts representing user requirements. However, regarding usability they insufficiently represent the design data for user interfaces. Furthermore, the user interface design is regarded as a secondary activity, separated from object-oriented analysis [4, 21]. Design methods of the HCI field, on the other hand, focus on usability. However, they are not well integrated into SE methods addressing a high architectural quality, entitled as gap between object-oriented (OO) and HCI methods; see [21].

Software architectures for systems with high complexity and a strong demand for flexibility are mostly built according to the component-based software engineering (CBSE) approach [20]. In this approach components are the main building blocks of the software architecture. The CBSE approach has superseded pure OO development techniques, but still has shortcomings in addressing user concerns for interactive systems. There are no widespread CBSE methodologies that deal with both, building high quality software systems, and also concentrate on the usability of a software system—which mainly depends on a good user interface design. Quasar [19, 18] can serve as an example, because it is an advanced method for building component-based software architectures, which assures important architectural quality criteria like separation of concerns, high cohesion, and low coupling. However, Quasar gives insufficient assistance for the GUI design. In this paper we introduce an integrated approach for the specification of GUI components as an extension of Quasar. For the

integration of HCI approaches to a comprehensive methodology, usability goals and GUI design principles have to be considered, for example the addition of user interface prototypes into the development life cycle.

For the evaluation of our approach we applied it in the development of an e-commerce trading system. The system named *Vendorbase* manages vendors, contracts with them including discounts, controlling and calculation. The system is of a middle size and is based on a framework for providing services for different architectural layers. Herpel [9] performed a reengineering of the business logic layer, while the GUI components were developed by Bode [1], together with an evaluation of our methodology. In this paper, parts of this project are used as a case study for illustration purposes.

The contribution of this paper consists in the integration of a high quality design method for software architectures with steps and models regarding the HCI design. The resulting extensions to the method represent the architectural principle of implementing non-functional requirements by functional solution elements. From the point of view of usability engineering, the resulting methodology constitutes a contribution related to the field of software architectural design.

The rest of the paper is organized as follows: The next section introduces the design methodology Quasar. Section 3 discusses some related HCI approaches. Section 4 in detail illustrates the steps of our methodology with the case study of the e-commerce system. Finally the last section states some conclusions and issues for future work.

2. The design methodology Quasar

The architectural design methodology Quasar—an acronym for QUAlity Software ARchitecture—was developed in industrial practice and combines Best Practices and principles for a good component-based software architectural design. We consider the Quasar architectural design methodology as the best one for component-based systems. It integrates the contributions of all preceding OO design methods. Beyond this, it achieves an improved decomposition of the software into components, because so-called software categories are introduced to reduce dependencies by a separation according to the responsibilities and the knowledge covered by the components [19, 18]. The improved properties regarding modularization, decoupling and separation of concerns increase a system's architectural quality. Quasar's design methodology contributes to evolvability, maintainability and robustness. It uses the categories to structure components according the requirements and thus to bridge the gap between analysis and architectural design [1, 22].

According to Quasar, all components are based on the standard categories 0, A, T, and R. For a concrete application design, these categories are therefore refined in a category model. Software of the 0-category is neutral concerning the application's functionality and independent of technical aspects. Modules, classes and interfaces with universal applicability belong to it, e.g. class libraries like the Java Runtime Environment. 0-software has a high degree of reusability, a low probability of changes and it creates no undesired dependencies. A-components are application specific but independent of technical issues. They contain the application logic and entity classes for the realization of the domain functionality. T-components deal with the technical aspects of the system, for example the implementation of an application programming interface (API) for database connectivity or an API for the GUI. They are independent of concrete application functions. R-software refers to representation; it establishes a connection between A- and T-components, however, minimizing the dependencies between them. This is achieved by transformation, for example to external data presentation formats like XML. Other ways of mixing A and T, e.g. the so-called AT-software, are prohibited, because they would re-introduce stronger dependencies.

The Quasar methodology defines a sequence of steps for the design of components and their interfaces. In [1] we have developed it further to establish three basic steps: component identification, interface specification and inner structuring. For the design of the components' interfaces, their dependencies are analyzed to distinguish between loose and tight coupling. However, the methodology does not provide enough guidance for designers and developers for the implementation of non-functional, quality features.

For user interface design, Quasar provides a client architecture as reference architecture for graphical user interfaces [18, 6, 7, 8], which is the basis for the technical specification of the GUI components and their programming interfaces. Although this client architecture simplifies the GUI development by providing a framework for GUI components, there are some deficiencies concerning the GUI component identification. Therefore, in this paper we apply the Quasar methodology and extend its component design steps by usability engineering for user interface design.

3. Related user interface design approaches

In the field of HCI several excellent methods and techniques regarding usability have been developed. The ones based on use cases offer the highest potential for a seamless integration within widely accepted SE approaches.

Use case storyboards are part of the GUI design method of Kruchten et al. [13] used in the Rational Unified Process

(RUP). This design method is use case-based and thus fits well into our methodology. It consists of the two activities user interface modeling and user interface prototyping. For user interface modeling the designer uses the artifact named *use case storyboard*.

A *use case storyboard* describes a logical and conceptual view of how the user interface provides a use case and which interactions occur between the software system and its users. Such a *use case storyboard* consists of several parts: a *flow of events-storyboard* as a high-level textual description of the user-system interaction, as well as interaction and structure diagrams, and further usability requirements. A *user interface prototype* is an initial model of the user interface and is expressed by paper sketches, pictures, bitmaps from a drawing tool, or even an interactive executable prototype. We use this approach for our component-based methodology because it allows a consideration of user needs and usability requirements early in the development life cycle. However, we had to adjust the method for the use of components instead of RUP's boundary classes.

User Interface Modeling (UIM) is a user-centered design method developed by Gulliksen et al. [5] that emphasizes user participation stronger than pure use case-based approaches. UIM specifies an actor model, a goal model, and a work model as extension to the use case model. It is based on the *workspace metaphor* stated by Lif et al. [15]. The basic idea of the metaphor is a one-to-one mapping between work situations of actors and a workspace in the user interface. So an actor has access to all necessary information needed in one work situation to achieve his goals. We use this *workspace metaphor* in our methodology because it is as an important tool to effectively tailor the hierarchy of GUI components according to user needs. However, UIM does not describe steps for creating interfaces, because GUI design is seen as a creative process that cannot be described in a method.

Virtual Windows are a way to design user interfaces presented by Lauesen [14]. His comprehensive methodology helps to overcome the semantic gap between software engineering and HCI. Lauesen shows a systematic way to design the GUI and is very concerned with usability. Unfortunately, this holistic methodology does not consider high-quality component-based software architecture design. However, Lauesen states some valuable rules and guidelines for the design of his Virtual Windows. These are [14]:

1. *Few window templates.*
2. *Few window instances per task.*
3. *Data in one window instance only.*
4. *Rooted in one thing.*
5. *Virtual windows close to final screen size.*
6. *Necessary overview of data.*
7. *Things—not actions.*
8. *All data accessible.*

These rules and some further principles for control elements in user interfaces can perfectly be used for the design of GUI components, too. This is the reason why we adopted them for the UI design of our methodology.

4. Designing GUI components

In this section we describe the design activities for components and their programming interfaces of the GUI. Because not covered by Quasar, we introduce the identification of GUI components. Quasar defines a fundamental sequence of the three design steps for business logic components: component identification—interface specification—inner structuring. For the GUI related components we adopt both, these steps and the UI modeling approaches presented in the last section. This extension for GUI components represents a contribution of this work for supporting a systematic implementation of usability in component-based quality software architectures.

Preceding design activities. In advance to the GUI design steps the designer must perform some activities according to the Quasar method. First of all, he specifies a use case model with textual use case descriptions, which for example can be achieved via the use case templates of Cockburn [3]. After the specification of the use cases the architectural design follows. The GUI design can be accomplished parallel to or after the design of the application kernel. Only the programming interfaces between both have to be specified before or in parallel. One constraint for the identification of GUI components consists in the following rule: Every concrete use case that is connected with an actor must be assigned to a particular user interface, but one user interface can handle multiple use cases. To support the establishment of this mapping and, thus, the identification of the right GUI components, the following steps are performed. Starting from the use cases the software designer decomposes the systems functionality by building a function tree or feature model [12]. Then, the designer builds the category model to separate the concerns [18, 9, 1]. All artifacts—the use case model, the function tree, and the category model—serve as input for the following GUI design steps.

4.1. GUI component identification

For the identification of the GUI components according to the client architecture of Quasar, the dialogs representing

the user interface have to be identified. Therefore, the functional decomposition of the software system constitutes a first point of reference. As explained above, the criteria for GUI component identification are not defined by Quasar. So we introduce here the *use case storyboard* approach by Kruchten et al. [13] to enable a consideration of the usability goals. Thus, the textual use case descriptions from the use case model are extended by usability requirements building a so-called *flow of events-storyboard*, which covers user needs and user-system interactions. Then, the *flow-of-events storyboard* is analyzed for information concerning the visualization. As an example, each entity object in an information system that is mentioned in the *flow of events-storyboard* must be visualized via a dialog. In order to fulfil the usability subgoals *task efficiency* and *learnability* the total number of dialogs should be minimized. For that reason, common dialogs have to be identified for reuse in several use cases. In the case study we identified the MainDialog that serves as a root for all other GUI components and is reused in all *use case storyboards*. This issue shows the integration of Lauesen's rules listed in section 3.

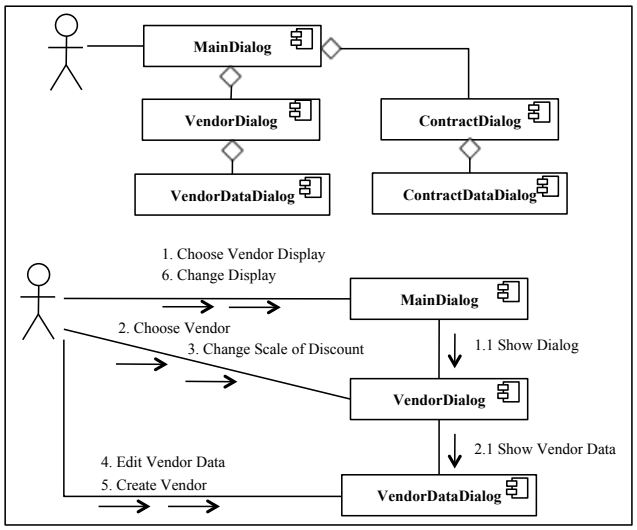

Figure 1. Use case storyboard diagrams.

Next, the dialogs are arranged within a hierarchy of dialogs. Originally, the *use case storyboards* dealt with boundary classes in its diagrams due to the use in the RUP. Instead of Kruchten's classes we introduce components, which can handle similar responsibilities. In this way the approach fits well in the component-based methodology. A component diagram is created for showing a structural view with aggregations of subdialogs (Figure 1, top). As a next step, the communication between the dialogs is defined based on the sequence of events of the *flow of events-storyboard*. The resulting communication diagram is used for the assignment of the responsibilities to the GUI components (Figure 1, bottom) and as a behavioral view to represent the interactions between them. All GUI components can exist independently of each other and communicate via a service hierarchy provided by the dialog frame of the client architecture of Quasar. As a rule, the identified GUI components in these diagrams have to cover all functional features from the function tree to enable an access to every use case.

In our case study the VendorDialog in Figure 1 is arranged as a subdialog of the MainDialog. The dialogs are structured in a hierarchy as shown in the component diagram in Figure 1. In the case study, the *workspace metaphor* [15] is chosen as a principle for the usability subgoal *task efficiency*. Following this principle, the VendorDialog and the ContractDialog are designed for an independent use. The *workspace metaphor* causes a similar arrangement of the ContractDialog in the dialog hierarchy, and later a similar layout in relation to the VendorDialog (Figure 2).

4.2. Definition of the component interfaces

In this step we apply the Quasar client architecture. It defines the style of interaction between the dialogs by services, and a dialog frame for managing it. For the interaction between the dialogs and the dialog frame special interfaces have to be defined—IDialog and ISession (Figure 2).

The interface design of the GUI components consists of two activities: *a*) interface design to fulfill the architectural style mentioned above; *b*) specification of the required interfaces according to the provided interfaces of the application kernel, where the business logic is implemented.

In the case study this is illustrated in the left part of Figure 2. The interfaces IDialog and ISession are added to all GUI components (activity *a*). Since the MainDialog constitutes a super-dialog for the VendorDialog, the IEmbedToolbarButtonService is added for embedding a button to the MainDialog's toolbar. Furthermore, services for accessing the details of a selected vendor are introduced. The ISelectedItemDisplayService and the IOverviewUpdateService for this purpose are shown between VendorDialog and VendorDataDialog in the left part of Figure 2. Moreover, the required interface IVendorManagerService is assigned to the GUI components as a service-oriented interface that is provided by the application kernel (activity *b*).

4.3. The definition of the GUI components' inner structure

According to the client architecture of Quasar, GUI components are separated into two parts, the dialog kernel belonging to category A and the presentation belonging to R.

The dialog kernel part defines user actions, dialog states, and the data to be presented. The presentation builds the

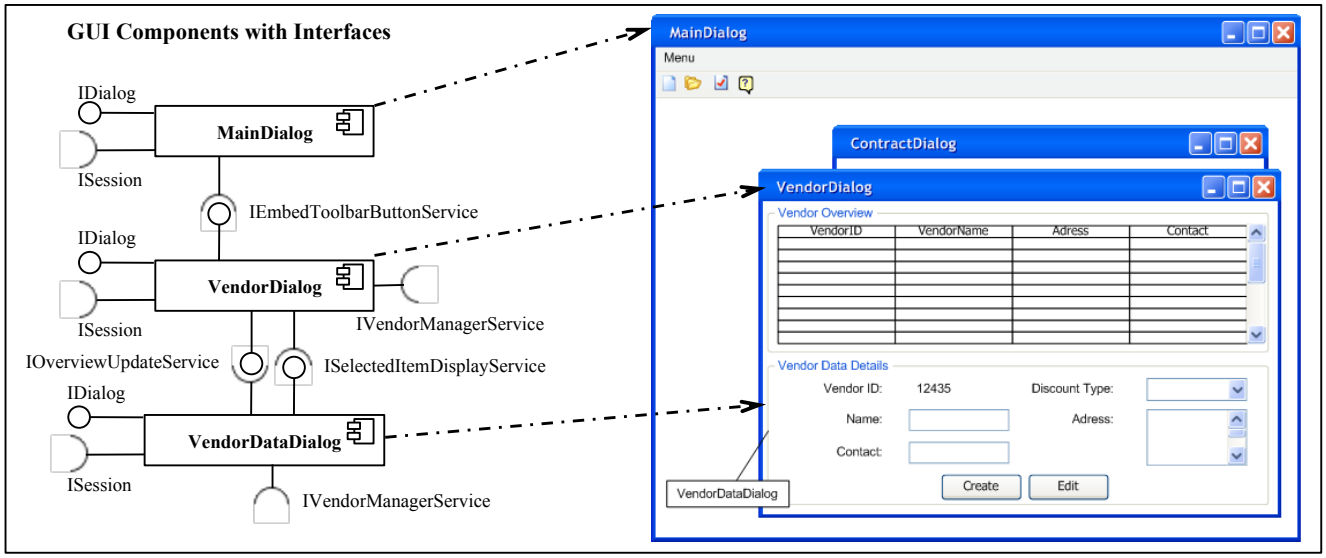

Figure 2. Correspondence between component structure and the visual dialog structure.

visual layout, connects user actions with GUI framework-events—so-called *action binding*—and transforms the data of the dialog kernel to the formats used by the GUI framework—so-called *data binding*.

In the case study, the presentation as the second part of the VendorDialog manages a Create button, among others. This means it has to connect the button to the corresponding user action; this is *action binding*. An example for *data binding* is to connect the name of the vendor, held in the dialog kernel, with the corresponding text field.

The application of the client architecture improves the architectural quality by separating the concerns while specifying the components and their categories precisely, and by using loose coupling through services. This for example enables the developer to easily exchange the GUI framework by modifying the presentation parts of the GUI components; while the dialog kernel parts with their application specific structure and behavior are preserved.

4.4. Developing the visual representation

In the last step, the designer decides about the visual layout and the control elements used. In this step usability is of particular importance. Therefore, several aspects of quality requirements as well as aesthetics, style and fashion have to be considered.

While designing the layout of the GUI many decisions have to be made on the arrangement of GUI elements and dialogs. To achieve the usability subgoals, the GUI design principles have to be weighed against each other. For the principles, which represent non-functional requirements, suitable functional solutions have to be found, thus, Bosch's architectural design method [2] is applied. All those usability-engineering decisions about solutions are traced back to the considered principles and further to the main goals. In the following the usability principles *workspace metaphor, consistency, clarity, aesthetically pleasing, balance*, and *symmetry* are considered.

A way to elaborate a good visual layout is to build prototypes and to discuss them with prospective customers and users. This is intended by the *use case storyboards* approach from section 3, too. While performing its user interface prototyping activity all primary windows must be identified. The identified GUI components are candidates for the primary windows. However, the navigation path between the windows should not become too long. This again corresponds to Lauesens rules from section 3, which can be a helpful utility for the designer when prototyping the user interface.

For our case study Figure 2 shows a scheme of the visual layout with its GUI elements for the GUI components MainDialog, VendorDialog, and VendorDataDialog. In order to realize a high *task efficiency* and *learnability*, the principles *workspace metaphor* and *consistency* were implemented using non-modal dialogs with a similar layout for VendorDialog as well as ContractDialog. Furthermore, shortcuts were included for advanced users. We decided to visualize the VendorDataDialog as an integrated panel instead of a separate subwindow as shown in Figure 2. This supports both *clarity* and *learnability*.

The usability subgoal user satisfaction especially can be achieved by following the HCI principles *aesthetically pleasing* using *balanced* and *symmetric* dialog layouts. In support of these principles, JGoodies [11] is utilized in the

case study as a technical component. It helps to accomplish the principles, for example by paying special attention on element alignment when using Java Swing.

5. Conclusion and future work

This paper presents a new design methodology for component-based software architectures targeting both on architectural quality and usability aspects. Its innovation is seen in an extension of the—in the opinion of the authors—most progressive architectural design methodology Quasar by HCI methods and techniques. The strengths of the Quasar method consist in the modularization, the decoupling and the independency achieved by distinguishing software by categories, leading to a high architectural quality. Furthermore, it provides an architectural style for the interface design of components and a process for the architectural design, summarizing other new approaches. Since it does not sufficiently support the HCI design, the most adequate design methods from this field have been integrated, e.g. the design approach *use case storyboards* by Kruchten et al. and the *workspace metaphor* by Lif et al. Moreover, several principles and guidelines concerning the usability of a GUI have been integrated, for example those of Lauesen. As a result, the methodology assures the development of components, which offer both a high architectural quality and a high usability. Additionally, the detailed description of the design activities facilitates the traceability of the design decisions.

The assumptions and limitations of the paper are related to an application for the design of client server systems rather than web services. Requirements are supposed to be specified as use cases. Addressees of the method are software designers and HCI developers. The paper describes especially those architectural design steps in detail that are related to the user interface.

Based on this work researchers can also consider to integrate other HCI methods to enhance the methodology for usability or consider its improvement for further non-functional properties. A classification of HCI principles and technical solutions according to their support for goals and subgoals related to usability would be useful for a systematic design. Another issue of future work consists in an explicit support for the establishment of traceability links for all design activities. For an overall traceability, the methodology's detailedness facilitates a tool support for an automated establishment of traceability links.

References

[1] S. Bode. Traceability und Entwurfsentscheidungen für Softwarearchitekturen mit der Quasar-Methode. Diploma thesis, Technical University of Ilmenau, Ilmenau, Germany, 2008.

[2] J. Bosch. *Design and use of software architectures: Adopting and evolving a product-line approach*. ACM Press/Addison-Wesley, New York, NY, USA, 2000.

[3] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, Boston, MA, USA, 2000.

[4] L. L. Constantine and L. A. D. Lockwood. Structure and style in use cases for user interface design. In van Harmelen [21], chapter 7, pages 245–279.

[5] J. Gulliksen, B. Gransson, and M. Lif. A user-centered approach to object-oriented user interface design. In van Harmelen [21], chapter 8, pages 283–312.

[6] M. Haft, B. Humm, and J. Siedersleben. Quasar reference interfaces for business information systems. Technical report, sd&m Research, 2004.

[7] M. Haft, B. Humm, and J. Siedersleben. The Architect's Dilemma – Will Reference Architectures Help? In *Proceedings Quality of Software Architectures and Software Quality, QoSA-SOQUA*, LNCS, pages 106–122. Springer, 2005.

[8] M. Haft and B. Olleck. Component-based client-architecture (in German: Komponentenbasierte Client-Architektur). *Informatik-Spektrum*, 30(3):143–158, 2007.

[9] K. Herpel. Refactoring und Identifikation von Komponenten. Diploma thesis, Technical University of Ilmenau, Ilmenau, Germany, 2007.

[10] ISO/IEC 9126-1 International Standard. Software Engineering - Product quality - Part 1: Quality models, June 2001.

[11] JGoodies. http://www.jgoodies.com/, 2008.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, SEI Institute, Carnegie Mellon University, USA, 1990.

[13] P. Kruchten, S. Ahlqvist, and S. Bylund. User interface design in the rational unified process. In van Harmelen [21], chapter 5, pages 161–196.

[14] S. Lauesen. *User Interface Design: A Software Engineering Perspective*. Addison-Wesley, Boston, MA, USA, 2005.

[15] M. Lif, E. Olsson, J. Gulliksen, and B. Sandblad. Workspaces enhance efficiency – theories, concepts and a case study. *Information Technology & People*, 14(3):261–272, 2001.

[16] J. Nielsen. *Usability Engineering*. Interactive Technologies. Academic Press, Boston, USA, 1993.

[17] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley, Boston, MA, USA, 2nd edition, 1992.

[18] J. Siedersleben. *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar*. dpunkt.verlag, Heidelberg, Germany, 2004.

[19] J. Siedersleben (ed.). Quasar: Die sd&m Standardarchitektur. Parts 1 and 2. Technical report, sd&m Research, 2003.

[20] C. Szyperski, D. Gruntz, and S. Murer. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley/ ACM Press, USA, 2nd edition, 2002.

[21] M. van Harmelen, editor. *Object Modeling and User Interface Design: Designing Interactive Systems*. Addison-Wesley, Boston, MA, USA, 2001.

[22] S. Wendler. Entwurfsentscheidungen bei der Entwicklung von Software-Architekturen. Diploma thesis, Technical University of Ilmenau, Ilmenau, Germany, 2007.