Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability

Stephan Bode and Matthias Riebisch

Ilmenau University of Technology
P.O. Box 10 05 65, 98684 Ilmenau, Germany
{stephan.bode, matthias.riebisch}@tu-ilmenau.de

Abstract. Quality goals have to be under a special consideration during software architectural design. Evolvability constitutes a quality goal with a special relevance for business critical systems. Architectural patterns can significantly contribute to the satisfaction of quality goals. But architectural design decisions regarding these goals have to be made in a systematic, methodical way and concerning the patterns' influence on quality properties. Unfortunately, pattern catalogs do not well support quality goal-oriented design decisions. This paper presents a systematic refinement and mapping of the quality goal evolvability to properties for good architectural design. A set of architectural patterns is evaluated regarding these properties. Furthermore, a calculation scheme is provided that enables the evaluation of the patterns to support design decisions. The results have been developed, revised, and evaluated in a series of applications based on industrial expertise.

1 Introduction

For the development of many types of software systems, the satisfaction of quality requirements and the appropriate options for future changes are among the major goals of software architectures, even more important than functional requirements [12]. Business critical systems demand for the constant provision of the business services and for a long lifetime for the return of the investment, while changes have to be performed with a high frequency. As a consequence, the rank of evolvability often is higher compared to many other quality goals. Quality attributes have been considered by recent architectural design methods and approaches, for example QASAR [7], Siemens' 4 Views [23], ADD [4], and QADA [28]. Their activities can be classified to the phases architectural analysis, synthesis, and evaluation [22], of which synthesis creates the candidate solutions balancing the quality and functional requirements.

According to the importance of quality goals for architectural design, a high risk is related to them. As a consequence, an effective guidance is needed during the development, especially for the implementation of goals such as evolvability, flexibility, and variability. Quality goals often compete or even conflict with each other and with functional requirements. A refinement of quality goals to quality properties eases the resolution of conflicts and the identification of compromises [25]. For balancing between functional and quality requirements, the utilization of patterns [20] or tactics [4] for architectural structuring constitute an effective way. Architectural decisions between the several solutions have to be made according to their impact on the quality properties. This shall result in a goal-oriented way of selecting patterns and tactics.

The architect's set of solution elements is usually contained in a toolbox representing a knowledge base of design knowledge. There are suggestions to structure a toolbox into two parts [30]: *(a)* a catalog of approved methods and solution templates (e.g. patterns), as well as *(b)* a catalog of fundamental technologies and tools (e.g. frameworks).

To enable the intended goal-oriented way of selecting solution elements, the impact of the toolbox elements on quality properties is required as a decision criteria. Usually, pattern catalogs (e.g. [13, 18]) provide descriptions for context, problem, and solution. Influences on quality properties of the resulting architecture are considered to a lesser extent, and qualitatively rather than quantitatively. Classification is related to pattern types instead of quality properties. Therefore, the catalogs do not sufficiently guide the architect in a pattern selection related to quality goals. Unfortunately, to the best of our knowledge there is no quantitative evaluation nor classification of architectural patterns regarding their impact on quality attributes, which is required for a goal-oriented pattern selection process. This is especially the case for the goal evolvability.

This paper presents an approach for the quantitative evaluation of the impact of architectural solution elements on quality goals, which provides all necessary means for a goal-oriented decision-making for architectural design. As described in prior works [5, 8], we refine quality goals to subcharacteristics to facilitate conflict resolution. The quality subcharacteristics are mapped to properties for good architectural design. Architectural solution elements such as patterns are then related to these properties, based on evaluations of their impact on the latter. We utilize our concept of the Goal Solution Scheme [5] to structure these relations and to form a knowledge base. A sequence of evaluations of the approach has lead to revisions of previous schemes, thus, achieving a higher degree of maturity. The results provide the means for the different steps of a goal-oriented design process, such as refining the goals, prioritizing the quality subcharacteristics, and providing a ranked list of candidate solution elements during architectural synthesis. The presented work is focused on the quality goal evolvability, however, it is intended for other quality goals as well.

The results have been developed, revised, and evaluated in a series of applications based on industrial expertise. Here we explain them with a case study of a software system for collective orderers, which additionally confirmed and improved our evolvability model from an earlier work [8].

The rest of the paper is organized as follows. We introduce the fundamentals for our evaluation in Section 2: the evolvability model with the subcharacteristics and the quality properties. Section 3 describes our procedure for the evaluation of the impact of architectural patterns. Then, in Section 4 the results of the evaluation are discussed. Section 5 deals with related work. Finally, Section 6 concludes the paper and gives an outlook on further work.

2 Evolvability Subcharacteristics and Design Properties

This section provides the fundamentals for our approach. Three elements form the base for our approach of goal-oriented decision support on architectural solutions: (1) A quality model with a refinement of quality goals to subcharacteristics and properties, (2) a process for selecting architectural solutions, and (3) an evaluation of solutions regarding their impact on quality goals—in this case evolvability. We use a definition of evolvability based on Breivold et al. [9] and Rowe et al. [32]:

Definition. *Evolvability is the ability of a software system throughout its lifespan to accommodate to changes and enhancements in requirements and technologies, that influence the system's architectural structure, with the least possible cost while maintaining the architectural integrity .*

2.1 The Evolvability Model

Evolvability of a software system is a property referring to the effort concerning different aspects of its evolution. This effort can be determined by the help of several subcharacteristics of evolvability, which we define by a quality model. This model is an extension of the works of Breivold et al. [9, 10] and Cook et al. [15] and was introduced earlier in [8].

For a goal-oriented way of decision-making during architectural synthesis, the impact of a decision on the quality goal has to be determined or predicted. Expert estimations constitute an effective way of impact determination. An expert in this regard should have experience with the implications of architectural patterns on quality properties in a certain class of software systems. The subjective character of expert estimations can be reduced by performing them on a detailed level and then aggregating the results.

We discovered that properties for good architectural design provide a proper refinement of the quality subcharacteristics to determine the impact of architectural elements on the different aspects of evolution effort. We modeled the refinement by a mapping between subcharacteristics and properties.

The subcharacteristics of evolvability are described in Table 1. These subcharacteristics are based on the ISO 9126 [24] and other works on evolvability [9, 10, 15]. They also strongly correlate to what Matinlassi et al. [27] call evolution qualities and additional characteristics (e.g. traceabiliy, variability) for specifying the quality goal maintainability.

The design properties used for refinement are listed in Table 2. The mapping between subcharacteristics and design properties is shown in Figure 1 and Table 3. In the figure we left out some direct dependencies and show the aggregated ones for a better visualization. In Table 3 an existing influence relation is represented by 1 if positive and by -1 if negative. Indentations in the tables express the refinements of subcharacteristics and properties. For example modularity aggregates cohesion and loose coupling. The indentations correspond to the refinement links in Figure 1.

Many evolvability subcharacteristics can be influenced by the architectural structure and behavior. However, there are important influence factors on evolvability as for example qualification and motivation of the team members, process maturity, or quality management activities. The development process with roles, phases, communication paths, and traceability has a large influence as well. Architectural structures cannot control these factors; they will be considered partly in the calculation scheme in Section 3.3. For the subcharacteristics (Table 1) and the design properties (Table 2) we marked the ones with a direct influence from architectural patterns by an * in the first columns. They are applied as evaluation criteria for the architectural patterns in the sequel.

The mapping relations have been developed in an iterative way, starting with hypotheses [8] and multiple steps of revision during application in case studies [31, 33]. Meanwhile, the relations and the way of calculating impact values can be

Table 1. Evolvability subcharacteristics.

Subcharacteristic	**Description**
Analyzability, Ease of comprehension, (Understandability)*	The capability of the software product to be diagnosed for deficiencies or causes of failures in the software and to enable the identification of influenced parts due to change stimuli (based on [24] and [9]).
Changeability/ Modifiability*	The capability of the software product to enable a specified modification to be implemented quickly and cost-effectively (based on [24] and [27]).
Extensibility*	The capability of a software system to enable the implementation of extensions to expand or enhance the system with new capabilities and features with minimal impact to existing system [9].
Variability*	The capability of a software system or artifact to be efficiently extended, changed, customized, or configured for use in a particular context by using preconfigured variation points (based on [34]).
Portability*	The capability of the software product to be transferred from one environment or platform to another [24].
Reusability*	The system's structure or some of its components can be reused again in future applications [27].
Testability*	The capability of the software system to enable modified software to be validated [24].
Traceability*	The capability to track and recover in both a forwards and backwards direction the development steps of a software system and the design decisions made during on-going refinement and iteration in all development phases by relating the resulting artifacts of each development step to each other (based on [19]).
Compliance to standards*	The extent to which the software product adheres to standards or conventions relating to evolvability (based on [24].
Process qualities	Additional process quality characteristics are for example Project Maturity and Community Quality, which are recognized as characteristics that influence the evolvability of open source software projects [17].

Table 2. Properties of good architectural design.

Property			**Description**
Low complexity*			The extent to which the amount/number of elements and their interdependencies are reduced.
	Abstraction*		The extent to which unnecessary details of information are hidden to build an ideal model and the extend to which a solution is generalized (based on [6]).
	Modularity*		The property of a software system to be decomposed into a set of coherent and loosely coupled elements with subsumption of abstractions (based on [6]).
		Cohesion*	The strength of the coupling between the internals of an element (based on [6]).
		Loose coupling*	The extent to which the interdependencies between elements are minimized (based on [6]).
	Encapsulation*		The extent of hiding the internals of an element for example by separation of interface and implementation (based on [6]).
	Separation of Concerns*		The extent to which different responsibilities are mapped onto different elements with as little as possible overlap, at which ideally one responsibility is assigned to exactly one specific element. The violation of this property is called tangling and scattering.
	Hierarchy*		The arrangement or classification of related abstractions ranked one above the other according to inclusiveness and level of detail (based on [6] and [29]).
	Simplicity*		The quality or condition of being easy to understand or do [29].
	Correctness		The property of an element to be complete and consistent resulting in a fulfillment of its responsibilities.
		Consistency	The absence of contradictions and violations between related elements.
		Completeness	The coverage of all relevant responsibilities by an element without lacking any necessary detail.
	Conceptual integrity		The continuous application of ideas throughout a whole solution, preventing special effects and exceptions (based on [11]).
Proper granularity*			The size and complexity of an element is appropriate to its responsibilities and to the particular situation.
Coherent mapping to concepts*			The way to map elements to ideas and mental pictures so that they are easy to understand, for example by proper names.

considered as rather mature. As an additional benefit of the evolvability model the refinement of the quality goals by mapping to properties enables a conflict resolution between competing quality goals, as discussed earlier in [5].

2.2 The Selection Procedure for Architectural Decisions

Architectural decisions concern design changes or the introduction of architectural solution elements, for example from the architect's toolbox. In order to implement a goal-oriented development we embed our approach into a two step procedure of selecting architectural solutions: (1) Architectural constraints are used to determine the set of applicable solutions by eliminating all unsuitable

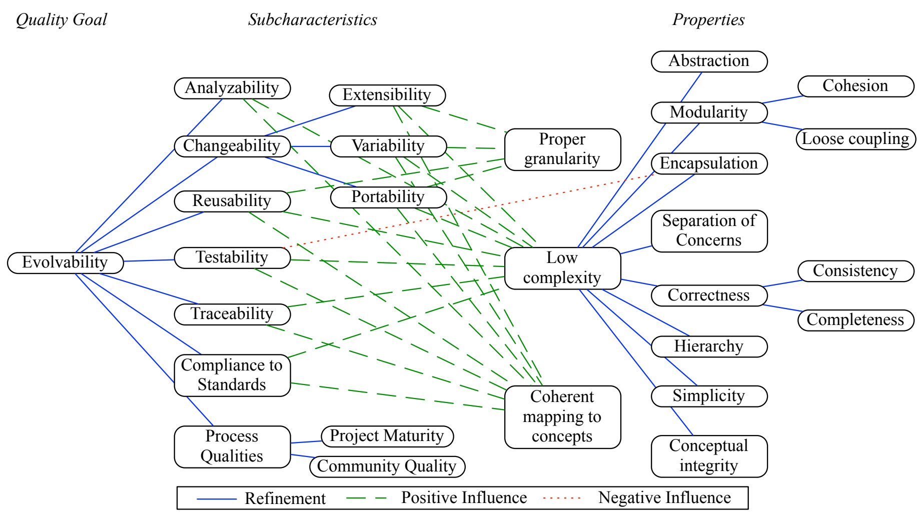

Fig. 1. Graphical representation of the evolvability model.

Table 3. Mapping of subcharacteristics to properties.

Property \ Subcharacteristic	Analyzability	Changeability	Extensibility	Variability	Portability	Testability	Reusability	Traceability	Compliance
Low complexity*								1	
Abstraction*	1		1	1	1		1		1
Modularity*									
Cohesion*	1		1	1	1	1	1		
Loose coupling*	1		1	1	1	1	1		
Encapsulation*	1		1	1	1	-1	1		
Separation of concerns*	1		1	1	1	1	1	1	
Hierarchie*	1		1	1	1	1			1
Simplicity*	1		1	1	1	1	1		
Correctness									
Consistency	1		1	1	1	1	1	1	
Completeness				1		1	1	1	
Conceptional integrity	1		1	1	1				1
Proper granularity*			1	1	1		1		
Coherent mapping to concepts*	1		1	1	1	1	1	1	1

1 – Positive influence; -1 – Negative influence

ones. (2) All solutions in the set are evaluated and ranked regarding the relevant quality goals. According to the ranking the architect selects and implements a solution. The evaluation needed for step 2 is presented in the next chapter.

3 Evaluation of Architectural Solution Elements

In this section we describe our approach for the evaluation of the impact of architectural patterns on evolvability. The concept of the approach is based on the evolvability model and the evaluation criteria presented in the last section. The evaluation itself is presented with a case study of a software system for collective orderers of mail order companies.

Case Study: Collective Ordering System. *Mail order companies prefer to work together with collective orderers, who accumulate orders of several customers and submit them as a collective order to the mail order company. The mail order company delivers the goods in one shipment to the collective orderer, who in turn distributes them to the customers. There are several advantages: The collective orderer knows the formalities and processes for rare procedures such as reshipment, complaint, deferred payment, etc. better than the average customer. The personal, familiar contact to customers has positive effects on the business volume. The mail order company can delegate communication activities with customers to the collective orderer. These procedures belong to the core business in the domain and are affected by frequent changes. Therefore, they were chosen for this case study.*

The software system of the company shall enable collective orderers to submit orders, manage their customers, and deal with complaints. We applied our approach for the task of enhancing this system. First, we selected some architectural patterns as explained in Section 3.1. Second, we determined the impact of the patterns on the properties for good architectural design (Section 3.2). For evaluation purposes, a suitable part of the collective ordering software was designed for each of the considered architectural patterns. This architectural design was used for the impact determination. Based on the results, the impact on the subcharacteristics was determined as discussed in Section 3.3. Finally, we aggregated the values to determine the impact on the quality goal evolvability (Section 3.4). The resulting values are stored together with the patterns in the architect's catalog. They can be used for future design decisions regarding this quality goal.

3.1 Selection of Patterns

For the impact determination regarding the quality goal evolvability, architectural and design patterns with an influence on the software architecture constitute interesting candidates. There is a high number of patterns available. For the evaluation with this case study we selected a set of patterns from the entirety which have an influence on the architecture, which are well documented, and which are expected to have an impact on evolvability. According to step 1 of our decision procedure, they have to fulfill the constraints of the software system of the case study. The selected patterns are listed in Table 4.

Preprint Version: The original publication is available at www.springerlink.com
http://dx.doi.org/10.1007/978-3-642-15114-9_15

Table 4. Selected architectural patterns.

Name	**Sources**
Client-Server	see Avgeriou&Zdun [1]
Layers/Tiers	
Repository	
Blackboard	
Pipes and Filters	
Model View Controller (MVC)	
Presentation Abstraction Control (PAC)	
Event-Based, Implicit Invocation	
Broker	
Micro Kernel	
Reflection	
Facade	Gamma et al. [18]
Adapter	
Proxy	
Plug-in	Manolescu et al. [26]

3.2 Determination of the Impact on the Properties

This section explains the determination of the impact values for the selected patterns in the case study. First, we applied the patterns in an exemplary architectural design for the case study. The resulting pattern-based design was rated regarding the impact on the properties for good architectural design. The ratings were gained through an assessment of the impact for each property by experts. The value of the impact is expressed by values of -2 (strong negative), -1 (weak negative), 0 (neutral), 1 (weak positive) and 2 (strong positive).

Case study example for impact discussion A collective orderer has to enter orders into the software system, and then the orders have to be transmitted to the mail order company. Usually this is done via phone but should be supported by the new software system. As a possible solution in the case study, we utilized the Client-Server pattern (see Figure 2(a)). The server at the mail order company is connected to the collective orderer's client via internet. It provides an interface for the transmission of orders. The client is structured in three layers as shown in Figure 2(b). The presentation layer is responsible for the graphical user interface (GUI). It uses the application layer, which provides functions as calculations for deferred payment, a search for ordered but not delivered goods, or a reminder for the deadline for returning the goods. The data layer is responsible for the data persistence in a databank.

Now we discuss the evaluation of the patterns Client-Server and Layers regarding their impact on the properties for good architectural design. They both have a strong positive impact on several properties. For example they provide a good abstraction of internal details (rating 2). The resulting architecture is simple to understand (2). They provide good modularity due to high cohesion

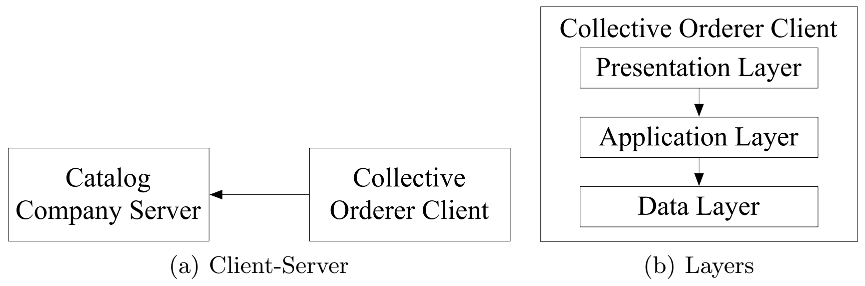

Fig. 2. Pattern application in the case study example

inside the layers, client, and server (2), as well as a loose coupling between the elements (2) for example for the deferred payment. Unnecessary details are hidden behind interfaces between the layers. Therefore, the encapsulation is improved (2). Regarding separation of concerns Client-Server and Layers have a positive impact, but they cannot completely prevent mixing different concerns (rating 1). Regarding the Hierarchy criteria the two patterns differ in their impact. The Layers pattern supports the ranking and grouping of abstractions on different levels very well due to the different layers (2). For the Client-Server pattern this cannot hold to this extend resulting in a lower rating (1). The same applies for the coherent mapping to concepts criteria. The Layers pattern has a weak positive impact on a proper granularity of an architectural design by structuring into layers instead of one big structural element (1). Overall, the Client-Server pattern and the Layers pattern reduce the complexity of an architecture through structuring.

Inside the client's presentation layer, the Model View Controller (MVC) pattern can be used to separate the data to be presented (e.g. a customer or order), from the different views and control mechanisms. For example there are views for editing the customers' contact information or for collecting and managing the orders.

The support for abstraction and cohesion as well as separation of concerns of MVC is very good (rating 2) as a result of the strict separation of model, view, and controller. This improves the simplicity of the design as well (1), although it is not so easy to use MVC with modern GUI libraries. The encapsulation is also good because the internals of each element are hidden behind interfaces (1). To build a hierarchy with MVC is not so well supported (0)—here Presentation Abstraction Control (PAC) would be better. Regarding coupling MVC is evaluated slightly negative (-1). Of course, the views can be decoupled from the model via a change-propagation mechanism, however, view and controller are coupled very tightly. Summed up, the complexity resulting from MVC is good but not excellent. The granularity that results from MVC can be quite good (1) if the models and views are properly designed. However, MVC's real strength is to provide a coherent mapping of concepts for the user interaction through the GUI (2).

Table 5. Values for the patterns' impact on the properties

Property \ Pattern	Client-Server	Layers/Tiers	Repository	Blackboard	Pipes&Filters	MVC	PAC	Impl. Invoc.	Facade	Adapter	Broker	Proxy	Micro Kernel	Reflection	Plug-in
Low complexity	1,7	1,8	0,8	0,3	1,7	1,1	1,4	0,8	1,3	1,4	1,5	1,5	2	0,3	1,6
Abstraction	2	2	0	0	2	2	2	1	2	2	2	2	2	2	2
Modularity	2	2	0	0,5	0	0,5	0,5	1	1,5	1,5	2	2	2	1	1,5
Cohesion	2	2	1	1	2	2	2	0	1	1	2	2	2	0	2
Loose coupling	2	2	-1	0	-2	-1	-1	2	2	2	2	2	2	2	1
Encapsulation	2	2	1	2	2	1	1	1	2	2	2	2	2	0	2
Separation of concerns	1	1	2	0	2	2	2	0	0	1	1	1	2	0	2
Hierarchie	1	2	0	0	2	0	2	0	0	0	0	0	2	0	0
Simplicity	2	2	2	2	2	1	1	2	2	2	2	2	2	-1	2
Proper granularity	0	1	0	0	2	1	1	0	2	1	0	0	1	0	2
Coherent mapping to concepts	1	2	1	0	2	2	2	0	1	1	2	0	2	0	2

For the rest of the patterns the evaluation concerning the properties was done in the same way. It cannot be explained here in detail due to space limitations. Table 5 shows the determined impacts of all selected patterns on the properties for good architectural design. The ratings of the aggregated properties modularity and low complexity are calculated by arithmetic mean of the subordinates.

3.3 Calculation of the Impact on Evolvability Subcharacteristics

The patterns' impact on the quality subcharacteristics is primarily determined from the impact on the properties, as discussed above. They are considered in the first step of the calculation. Additional influences on the subcharacteristics—for example from efforts not related to the properties of good design—are considered by introducing adjustments in a second step.

We calculated the results in the following way. Let $\boldsymbol{R}$ be the matrix of the impact ratings for the properties (Table 5) and $\boldsymbol{r}_p$ be a column vector of this matrix for one element p of the set of patterns P. Let $\boldsymbol{M}$ be the mapping matrix of Table 3 and $\boldsymbol{M}^*$ be $\boldsymbol{M}$ reduced by the rows for which the properties were not evaluated (and are not marked with *). Further, let $\boldsymbol{m}_s$ be a column vector of $\boldsymbol{M}^*$ for one element s of the set of subcharacteristics S. Moreover, let $\boldsymbol{V}$ be the matrix with the impact values of the patterns on the subcharacteristics. Then, each element v_{sp} of $\boldsymbol{V}$ is calculated by

$$v_{sp} = \boldsymbol{r}_p \cdot \boldsymbol{m}_s / \left\| \boldsymbol{m}_s \right\|_1 .$$

Finally, the matrix $\boldsymbol{V}'$ in the top of Table 6 is obtained from $\boldsymbol{V}$ by calculating the impact values for changeability in row 2 by the arithmetic mean of the values for extensibility, variability, and portability. In this way we determine the patterns' impact values on the subcharacteristics from the direct ratings for the properties (Table 5) by evaluating and normalizing the influences of the interdependencies described by the mapping in Table 3.

However, through the calculation there is no discrimination regarding the subcharacteristics of changeability. The Reflection pattern for example contributes to extensibility and variability but reduces portability if the base technology does not support reflection. Furthermore, testability is decreased due to possible dynamic changes at runtime.

These effects are not represented by the aggregated impact values in $\boldsymbol{V}'$ of the first step. Therefore, we considered offset values o_{sp} shown in the middle of Table 6 for the determination of the patterns' impact on the subcharacteristics. To determine these offset values we also incorporated knowledge about consequences of the pattern application regarding quality properties mentioned in the literature (e.g. Buschmann et al. [13]). The final impact values are calculated as follows. Let $\boldsymbol{F}$ be the matrix for the adjusted impact values. Then each element f_{sp} of $\boldsymbol{F}$ is calculated by

$$f_{sp} = \begin{cases} v_{sp} & \text{if } o_{sp} \text{ is undefined} \\ (v_{sp} + o_{sp})/2 & \text{otherwise.} \end{cases}$$

The final impact values $\boldsymbol{F}'$ for the subcharacteristics including changeability shown in the bottom of Table 6 again are obtained as for $\boldsymbol{V}'$ before.

3.4 Determining the Impact on Evolvability

As the last step the overall impact of the patterns on the quality goal evolvability is to be determined by aggregating all subcharacteristics with equal weights. The resulting values of the patterns' impact on evolvability are shown in Figure 3 and in the lowermost row of Table 6. The experts gave feedback on the results. This feedback led to a minor revision of the offset values discussed in the last section. The changes resulting from the offset values can be seen by comparing the unadjusted and the final values shown in Figure 3. As a consequence of the evaluation, a plug-in-based architecture was selected as the best solution.

4 Discussion of the Results

The final results of the impact evaluation are illustrated by Figure 3. The chart shows that the patterns in general do have a positive impact on the quality goal evolvability. Some patterns turned out to be excellent, for example Layers, Plug-in, or Pipes and Filters; others are less supportive. However, the impact of the patterns on evolvability and on software quality in general is limited if process aspects are not taken into account. We have considered process qualities only partly by the adjustments. Traceability constitutes another aspect important for evolvability which depends on the development process rather than on patterns.

Buschmann et al. [13] argue that a classification of patterns into groups is necessary to help the architect with the utilization of a system of patterns. We

Table 6. Impact values for subcharacteristics and evolvability

Subcharacteristic \ **Pattern**	Client-Server	Layers/Tiers	Repository	Blackboard	Pipes&Filters	MVC	PAC	Impl. Invoc.	Facade	Adapter	Broker	Proxy	Micro Kernel	Reflection	Plug-in
Calculated Rating															
Analyzability	1,6	1,9	0,8	0,1	1,5	1,1	1,4	0,8	1,3	1,4	1,6	1,4	2,0	0,4	1,6
Changeability	1,4	1,8	0,7	0,2	1,6	1,1	1,3	0,7	1,3	1,3	1,4	1,2	1,9	0,3	1,7
Extensibility	1,4	1,8	0,7	0,2	1,6	1,1	1,3	0,7	1,3	1,3	1,4	1,2	1,9	0,3	1,7
Variability	1,4	1,8	0,7	0,2	1,6	1,1	1,3	0,7	1,3	1,3	1,4	1,2	1,9	0,3	1,7
Portability	1,4	1,8	0,7	0,2	1,6	1,1	1,3	0,7	1,3	1,3	1,4	1,2	1,9	0,3	1,7
Testability	1,4	1,8	0,8	-0,2	1,2	1,0	1,4	0,6	0,8	1,0	1,4	1,0	2,0	0,2	1,4
Reusability	1,5	1,8	0,8	0,3	1,5	1,3	1,3	0,8	1,5	1,5	1,6	1,4	1,9	0,4	1,9
Traceability	1,2	1,6	1,3	0,8	1,9	1,7	1,8	0,3	0,8	1,1	1,5	0,8	2,0	0,1	1,9
Compliance to standards	1,3	2,0	0,3	0,0	2,0	1,3	2,0	0,3	1,0	1,0	1,3	0,7	2,0	0,7	1,3
Offset															
Analyzability			0	-1				-1	2	0	1		0		
Changeability															
Extensibility	1	1	2	-1		2	2	2	2	2	2			2	2
Variability	1	1				2	2	2		2				2	2
Portability	2	2				2	2			2	2	2		-1	
Testability				-2				-2	-1			2	0	-1	2
Reusability								2	2	2				1	
Traceability								2							
Compliance to standards											2		0		2
Final Values															
Analyzability	1,6	1,9	0,4	-0,4	1,5	1,1	1,4	-0,1	1,6	0,7	1,3	1,4	1,0	0,4	1,6
Changeability	1,4	1,6	0,9	0,0	1,6	1,6	1,7	1,1	1,4	1,7	1,6	1,4	1,9	0,7	1,8
Extensibility	1,2	1,4	1,3	-0,4	1,6	1,6	1,7	1,3	1,7	1,7	1,7	1,2	1,9	1,2	1,8
Variability	1,2	1,4	0,7	0,2	1,6	1,6	1,7	1,3	1,3	1,7	1,4	1,2	1,9	1,2	1,8
Portability	1,7	1,9	0,7	0,2	1,6	1,6	1,7	0,7	1,3	1,7	1,7	1,6	1,9	-0,3	1,7
Testability	1,4	1,8	0,8	-1,1	1,2	1,0	1,4	-0,7	-0,1	1,0	1,4	1,5	1,0	-0,4	1,7
Reusability	1,5	1,8	0,8	0,3	1,5	1,3	1,3	1,4	1,8	1,8	1,6	1,4	1,9	0,7	1,9
Traceability	1,2	1,6	1,3	0,8	1,9	1,7	1,8	1,1	0,8	1,1	1,5	0,8	2,0	0,1	1,9
Compliance to standards	1,3	2,0	0,3	0,0	2,0	1,3	2,0	0,3	1,0	1,0	1,7	0,7	1,0	0,7	1,7
Evolvability	1,4	1,8	0,7	-0,1	1,6	1,3	1,6	0,5	1,1	1,2	1,5	1,2	1,5	0,3	1,8

agree to this argument for general categories like architectural pattern or design pattern, structural or behavioral pattern, or regarding problems like concurrency or distribution. However, for effort-related quality properties a quantitative evaluation is more effective than a categorization because the impact on effort varies within an interval. In our decision procedure, step 1 results in a very similar effect as Buschmann's categories. The ranking of step 2 supports the architect in selecting the most appropriate solution element.

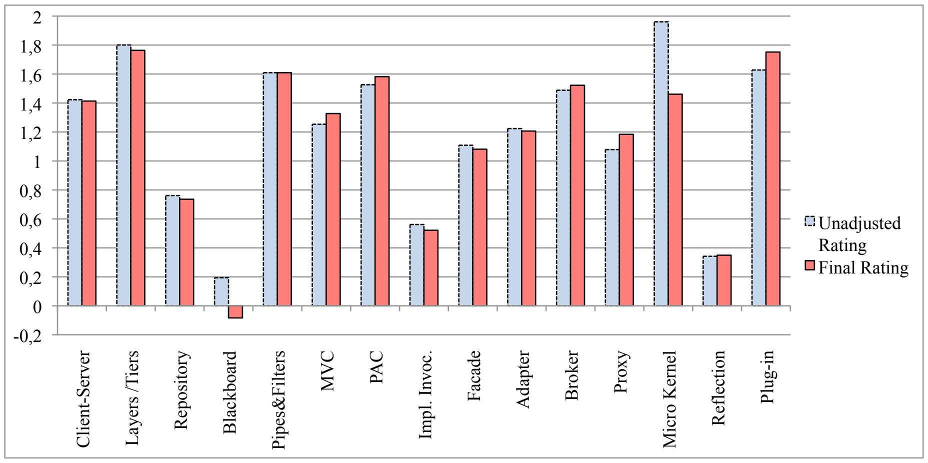

Fig. 3. Resulting Impact of Patterns on Evolvability

The impact values have been derived from a concrete case study. We must admit that they are subjective by nature because they were determined from expert opinion. The results of an expert survey also depend on the application conditions as the experience of the development teams. However, an improvement regarding objectivity is possible by including several experts. The values shall be applicable and applied in further projects. The degree of universality of the impact values can be improved by revising them in a series of projects.

The presented results have been developed as hypotheses, later revised, and evaluated in a series of applications based on industrial expertise. They can be considered as rather mature. Even if forthcoming revisions might result in smaller modifications of the impact values, the revisions of the relations for refinement and mapping (Fig. 1) can be expected to be minor ones.

5 Related Work

There is a lot of literature with works about patterns and their classification in pattern catalogs, some of which were already mentioned in the preceding sections. A good start to read are Avgeriou and Zdun [1], Buschmann et al. [13], or the seminal work of Gamma et al. [18]. Unfortunately, there is no catalog of patterns for evolvability. For security such a thing exists already for a quite long time [35].

A similar work to ours is the one of Harrison and Avgeriou [21]. They give an overview about the strengths and liabilities of a set of common architectural patterns regarding their impact on the qualities of the ISO 9126. They also present a design method for architectural design, which describes how to select the appropriate patterns for balancing quality and functional requirements in

[20]. However, they do not discuss the impact of the patterns on the properties for good architectural design as we do. Furthermore, our work has a strength in focussing on evolvability.

Architectural tactics discussed in several works by the SEI (e.g. [3], [2]) can be other means for a proper selection of patterns during architectural design. They are used for example in the Attribute Driven Design (ADD) method [4]. The tactics give some qualitative hints for choosing patterns concerning several quality attributes as e.g. modifiability or performance. But they do not consider design principles either, only partly deal with evolvability through modifiability, and cannot be interpreted quantitatively as the ratings in our approach.

POSAAM [16] is a method for quantitative architectural evaluation that relates patterns to quality attributes and design principles in an ontology. In this it is akin to our approach, though not for architectural synthesis.

Furthermore, the NFR-Framework of Chung et al. [14] has some similarities to our evolvability model. They link quality goals, so-called softgoals, and their subgoals via contribution links and operationalize them with solutions to perform an evaluation of the quality goals' satisfaction. However, the NFR-Framework has a requirements-oriented viewpoint, and therefore, does not consider architectural design principles or properties important for architectural synthesis.

6 Conclusion and Further Work

In this paper as the major contribution we presented an approach for the quantitative selection of architectural elements regarding quality goals. It consists of a quality model for evolvability as a basis for the evaluation of architectural patterns regarding their impact on the quality goal evolvability. We defined subcharacteristics of evolvability and mapped them to properties for good architectural design in order to be able to determine the impact on evolvability. Additionally, we presented our calculation scheme for the evaluation together with the results of the evaluation. The evaluation is embedded in a decision procedure on architectural elements in a toolbox. With a case study we explained how to determine the patterns' impact on the properties, to calculate the impact on the subcharacteristics through our mapping, then to consider additional influences by offset values to fit expertise knowledge, and finally, to aggregate the ratings to the final impact values on evolvability. The results show a considerable impact of architectural patterns on evolvability, although this quality goal cannot be addressed by merely using architectural patterns.

Another contribution of this work consists in a quantitative evaluation of architectural patterns regarding their impact on the quality goal evolvability and its subcharacteristics. In our opinion this is a valuable mean for supporting the decision-making process of a software architect. Using the patterns' impact values, a software architect can enrich his toolbox and rank the patterns according to their quality impact. This eases the search and selection of appropriate solutions for quality goals during architectural design. Furthermore, the mapping of quality goals on subchararcteristics facilitates the resolution of conflicts between competing quality goals.

In further works, the mapping relations between properties and subcharacteristics will be investigated in more detail, to elaborate a weighting of the mapping relations. Moreover, the results can be combined with knowledge on patterns' impact on other quality attributes, in addition to evolvability. Furthermore, with our approach additional solution concepts of an architect's toolbox can be evaluated, for example architectural refactorings or frameworks. Tool support for the approach is currently developed.

References

1. Avgeriou, P., Zdun, U.: Architectural patterns revisited – a pattern language. In: 10th European Conf. on Pattern Languages of Programs (EuroPlop 2005), Irsee. pp. 1–39 (2005)
2. Bachmann, F., Bass, L., Nord, R.: Modifiability tactics. Tech. Rep. CMU/SEI-2007-TR-002, CMU/SEI (September 2007)
3. Bachmann, F., Bass, L., Klein, M.: Deriving architectural tactics: A step toward methodical architectural design. Tech. Rep. CMU/SEI-2003-TR-004, CMU/SEI (Mar 2003)
4. Bass, L.J., Klein, M., Bachmann, F.: Quality attribute design primitives and the attribute driven design method. In: Revised Papers from 4th Int. Workshop on Software Product-Family Engineering. LNCS, vol. 2290, pp. 169–186. Springer (2002)
5. Bode, S., Fischer, A., Kühnhauser, W., Riebisch, M.: Software architectural design meets security engineering. In: Proc. 16th Int. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS '09). pp. 109–118. IEEE (2009)
6. Booch, G.: Object Oriented Analysis and Design. With Applications. Addison-Wesley Longman, Amsterdam (Oct 1993)
7. Bosch, J.: Design and use of software architectures: Adopting and evolving a product-line approach. ACM Press/Addison-Wesley, New York, NY, USA (2000)
8. Brcina, R., Bode, S., Riebisch, M.: Optimization process for maintaining evolvability during software evolution. In: Proc. 16th Int. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS '09). pp. 196–205. IEEE (2009)
9. Breivold, H.P., Crnkovic, I., Eriksson, P.: Evaluating software evolvability. In: Proc. of the 7th Conf. on Software Engineering Research and Practice in Sweden (SERPS'07). pp. 96–103. IT University of Göteborg, Göteborg, Sweden (2007)
10. Breivold, H.P., Crnkovic, I., Land, R., Larsson, S.: Using dependency model to support software architecture evolution. In: 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. pp. 82–91. IEEE (Sept 2008)
11. Brooks, F.P.: The Mythical Man-Month : Essays on Software Engineering. Addison-Wesley (1995)
12. Brown, A.W., McDermid, J.A.: The art and science of software architecture. In: Oquendo, F. (ed.) Proceedings First European Conference on Software Architecture (ECSA 2007). LNCS, vol. 4758, pp. 237–256. Springer (September 2007)
13. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons (1996)
14. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional Requirements in Software Engineering, Int. Series in Software Engineering, vol. 5. Kluwer (2000)

15. Cook, S., Ji, H., Harrison, R.: Dynamic and static views of software evolution. In: 17th IEEE International Conference on Software Maintenance (ICSM'01). pp. 592–601. IEEE, Los Alamitos, CA, USA (Nov 2001)
16. da Cruz, D.B.: POSAAM – Eine Methode zu mehr Systematik und Expertenunabhängigkeit in der qualitativen Architekturbewertung. Ph.D. thesis, TU München (2009)
17. Deprez, J.C., Monfils, F., Ciolkowski, M., Soto, M.: Defining software evolvability from a free/open-source software perspective. In: Proceedings of the Third International IEEE Workshop on Software Evolvability. pp. 29–35. IEEE (Oct 2007)
18. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Softwaresystems. Addison-Wesley (Nov 1994)
19. Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, Colorado Springs, CO, USA. pp. 94–101. IEEE (Apr 1994)
20. Harrison, N., Avgeriou, P.: Pattern-driven architectural partitioning: Balancing functional and non-functional requirements. In: Second International Conference on Digital Telecommunications, 2007. ICDT '07. IEEE (July 2007)
21. Harrison, N., Avgeriou, P.: Leveraging architecture patterns to satisfy quality attributes. In: Proc. ECSA 2007. vol. 4758/2007, pp. 263–270. Springer (2007)
22. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software 80(1), 106–126 (Jan 2007)
23. Hofmeister, C., Nord, R., Soni, D.: Applied software architecture. Addison-Wesley, Boston, MA, USA (2000)
24. ISO/IEC: ISO/IEC 9126-1 International Standard. Software Engineering - Product quality - Part 1: Quality models (June 2001)
25. van Lamsweerde, A.: From system goals to software architectures. In: Formal Methods for Software Architectures. LNCS, vol. 2804/2003, pp. 25–43. Springer (2003)
26. Manolescu, D., Voelter, M., Noble, J.: Pattern Languages of Program Design 5. Addison-Wesley Professional (May 2006)
27. Matinlassi, M., Niemelä, E.: The impact of maintainability on component-based software systems. In: Proc. 29th Euromicro Conf., 2003. pp. 25–32. IEEE (2003)
28. Matinlassi, M., Niemelä, E., Liliana, D.: Quality-Driven Architecture Design and Quality Analysis Method. A Revolutionary Initiation Approach to a Product Line Architecture. Tech. Rep. 456, VTT Technical Research Centre of Finland (2002)
29. McKean, E.: The New Oxford American Dictionary. Oxford University Press, 2 edn. (May 2005)
30. Posch, T., Birken, K., Gerdom, M.: Basiswissen Softwarearchitektur: Verstehen, entwerfen, wiederverwenden. dpunkt.verlag, 1 edn. (2004)
31. Riebisch, M., Bode, S.: Software-Evolvability. Informatik-Spektrum 32(4), 339–343 (Aug 2009)
32. Rowe, D., Leaney, J., Lowe, D.: Defining systems architecture evolvability - a taxonomy of change. In: Proceedings of the 11th International Conference on the Engineering of Computer Based Systems (ECBS'98). pp. 45–52. IEEE (1998)
33. Stollberg, R.: Klassifikation von Architekturstilen und -mustern hinsichtlich qualitativer Ziele für den Softwarearchitekturentwurf. Bachelor thesis, Ilmenau University of Technology, Ilmenau, Germany (2010)
34. Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software: Practice and Experience 35(8), 705–754 (2005)
35. Yoder, J.W., Barcalow, J.: Architectural patterns for enabling application security. In: 4th Conf. on Patterns Languages of Programs (PLoP '97) (1997)