Introducing Impact Analysis for Architectural Decisions

Matthias Riebisch, Sven Wohlfarth
Department of Software Systems / Process Informatics
Technical University Ilmenau, 98684 Ilmenau, Germany
{ sven.wohlfarth | matthias.riebisch }@tu-ilmenau.de

Abstract

Architectural quality constitutes a critical factor for contemporary software systems, especially because of their size and the needs for frequent, quick changes. For success-critical business systems, architectural decisions are of high risk for the market share and even for the existence of enterprises. These decisions are important for design processes as well as for refactoring. Because of the complexity of the decisions, e.g., uncertain, contradicting goals, unknown effects and risky conditions, decision-making is a difficult and risky task. Risks can be minimized if the decisions are made systematically. In an earlier paper, we introduced methods of Decision Theory to perform such decisions in a rational way. This paper introduces a method for evaluating alternatives of architectural decisions, for both architectural design and refactoring. This method adopts elements of the scenario-based evaluation method ALMA [1]. A practical example illustrates the application of the improved decision process.

***Keywords**: Refactoring, Reengineering, Software Architecture, Impact Analysis, Decision Theory*

1. Introduction

The complexity of software-intensive systems is increasing rapidly. Many of them are critical for the existence of an enterprise. If such a system could not adapted to changed market needs, this would lead to a loss of market share. Missing flexibility and maintainability represent high risks for mission-critical software. Some of the architectural decisions during the development and evolution of systems of this kind are frequently called strategic ones, because they determine future decisions and measures. Examples for such decisions are the definition of a middleware platform, of a database management system (DBMS) or an architectural style for distribution and cooperation between subsystems. Such decisions bear a high risk, because later changes require high effort and – even more important – delays. Unfortunately, these decisions are hard to make:

- The objectives and conditions are mostly complex, partly contradicting, and not completely known;
- Solution alternatives are hard to evaluate, because their consequences are mostly uncertain.

To reduce risks and to facilitate such decisions, a methodical and rational procedure was developed [2], based on the Decision Theory. Additionally, Impact Analysis is required to forecast the consequences of an alternative e.g., quality improvements, side effects and implementation effort. The identification of the consequences of a particular change in the software engineering domain is called Impact Analysis [4],[5]. Even if such a way of decision-making requires some effort, it increases the overall efficiency of the development processes by reducing risks. As a result, architectural decisions are made less subjectively.

The formerly developed procedure of rational decision-making [2] includes the following phases, as shown in fig. 2: identification of objectives, description of the architecture, and selection, analysis and evaluation of the alternatives. The benefit of this process results especially from the reduction of ambiguity, uncertainty and fuzziness of objectives and conditions, and from the way of weighting the consequences related to a broad variety of aspects.

Even if made in a rational way, the decisions are based on a proper establishment, selection and evaluation of architectural alternatives. Especially the Impact Analysis has to be done in a systematic way, because it is highly complex: Accurate positive *and* negative quality impacts have to be determined, and various side effects and conditions have to be considered, e.g., restrictions resulting from previous decisions. Furthermore, the Impact Analysis bears an uncertainty because it is related to a future – currently not existing – architecture, which has to be managed. The Decision Theory as the basis of the decision process does not provide methodical support for the Impact Analysis of architectural decisions, it just gathers the results.

Analyzing the Impact of alternatives, based on an evaluation of several executable prototypes of the resulting software system is too effort and time consuming for most cases. Estimations would require less effort, but pure subjective estimations bear the risk of being vague, too optimistic or too pessimistic. Wrong or unrealistic estimations would lead to wrong decisions. However, additional effort for an Impact Analysis is justified if it is related to the risk of wrong decisions. If a wrong decision for a business critical system would lead to an organizational-wide delay of new products and services, e.g., if the distribution of a product is impossible for some days, it would have been much better to spend one or two days in making the best decision.

The goal of this paper is to extend the decision process on architectural alternatives by a scenario-based Impact Analysis. This Impact Analysis is developed with strong similarities to scenario-based methods of architectural assessment, especially ALMA [1]. Since ALMA was developed with different goals, it could not simply be integrated.

This paper is organized in the following way: At first, three major State of the Art approaches for Impact Analysis are compared and evaluated in section 2. In section 3, the decision process [2] is explained in brief. In section 4, the case study is introduced, which is applied in section 5 to illustrate the introduction of the an Impact Analysis to the decision process in detail, by showing its activities in a refactoring decision. In the conclusion, evaluation results and open issues are discussed.

2. State of the Art in Impact Analysis Approaches

The Impact Analysis was introduced to the software engineering domain because side effects related to a change constitute highly critical issues [4][5]. Side effects can occur in unchanged components, which are related to the changed ones, e.g., by dependencies, constraints or import-export relations. An Impact Analysis can be done before or after an implementation of a change.

Any Impact Analysis approaches have to reduce uncertainty and risks of the architectural decisions. In the background of this paper, the different approaches are therefore compared according to the following criteria:

Degree of reduction of uncertainty and risk: There are different way to reduce uncertainty and risks related to the design process, e.g., a prevention of subjective influences and uncertainty, as well as a systematic, rational procedure, an explicit manifestation of dependencies and constraints in models, and an analyses of dependencies such as side-effects.

Ability to forecast quality improvements, effort and functional changes: Quality properties of an software architecture are usually regarded as more important than functional properties. An Impact Analysis has to evaluate quality properties e.g., maintainability, portability, robustness, performance and scalability. Furthermore, the implementation effort and the impact on the functional behavior are important criteria.

Ability to manage the complexity of architectural decisions: Software systems and their architectures are highly complex. They are characterized by strong interrelations and constraints. Decision-making by humans requires for abstractions and simplifications to master this complexity.

Appropriate for usual development processes: The Impact Analysis has to fit into the decision process [2] as well as into development processes seamlessly.

Additional effort: The extra effort of the decision-making – including the one of the Impact Analysis – should be as low as possible. However, the higher the risk of a decision, the more effort is justified.

Degree of tool support and automation: Tool support and automation of the Impact Analysis would reduce its extra effort and would increase the efficiency of software and system development.

The following three major Impact Analysis approaches are investigated, described, and analyzed in the following section, whether they fulfill the criteria or not.

Graph-Based Analysis

The impact of architectural changes to a system is caused by dependencies between its parts. To describe an architecture formally, a graph-based model represents one option. In a graph model, the system's components and dependencies are expressed by nodes and edges. Graph analyses [4] can then be applied to investigate the effects of an architectural change: the more nodes are effected by a change, the higher is the effort. Path analyses can be applied to identify potential effects to the functional behavior. The establishment as well as the analysis of a graph-based model can be performed in a rather systematic way, thus the risks of subjectivity and uncertainty do not apply. However, most quality properties such as maintainability, reliability and robustness of a system are hardly to model within the same graphs representing the structure of that system, therefore this approach is not suitable to all relevant criteria. Performance can be mentioned as a non-functional criterion which can be evaluated well using graph-based analyses.

The formal description of a complete architecture including the architectural alternatives requires a much higher effort than a semi-formal one as usual for documentation purposes, because the complete system including all details of its parts and its environment has to be described formally to enable an analysis. Graph-based analyses offer some potential for integration into usual development processes because they are based on models which could be established within the design phase. Their application requires special skills. Tool support is available for performing analyses, however the establishment of a graph model has to be performed manually.

Graph-based analyses are to some extend suitable to perform an Impact Analysis for architectural decisions, however they cannot be applied for most criteria as well as for low effort analyses.

Simulation

The simulation of the runtime behavior represents another method to analyze the consequences of architectural changes. Most simulation approaches focus on events, which are triggered by rules, and the traces of their effects.

To simulate an architecture, the relevant parts of the architecture, e.g., subsystems or components, are modeled by an Architecture Description Language (ADL), e.g., Rapide [16]. Components, interfaces and dependencies are modeled to express the structure of a system. Additionally, events, rules and trigger have to be defined to enable a simulation of the runtime behavior of the architecture after a change. The events are triggered through the defined rules and a tool captures and traces the effects [16]. If all relevant parts of an architecture are described, the resulting model can be executed to perform simulations.

To analyze architectural changes, the model has to be evolved prototypically according to the necessary changes. Establishing and evolving such architectural models requires a high effort, because the both the model and the prototype have to be modeled up to deep details. As a result, the developer can check if the events are processed correctly. Additionally, the impact on the performance can be analyzed.

The simulation is useful, if the performance of the software system is the most critical aspect of an architectural decision. Furthermore, the simulation eases the analysis if the functional behavior of the software system before and after changes. However, a concept is missing to analyze the quality-related consequences, like the impact on the portability or maintenance, in a systematic way.

Similar to graph-based analyses investigated above, simulation approaches offer some potential for integration into usual development processes because based on models. Furthermore, their application requires special skills, especially for establishing the models. Tool support is available for performing the simulations, however the establishment of the models requires high manual effort.

This high effort is only justified, if the simulation does help to solve very high risks concerning correctness and performance. The approach is less suitable if other non-functional criteria are more important.

Scenario-Based Analysis

A scenario-based architectural analysis consists of a formal review for the assessing a software architecture. The analysis is performed by developers in the way of a walkthrough following to a scenario, which is a peer-review following to that scenario. The Architecture Tradeoff Analysis Method (ATAM) [6] is one of the most popular methods, offering advantages in comparison to other scenario-based analysis methods [7]. ATAM describes a complete process of preparing and performing the walkthrough (see fig. 1). According to the relevant evaluation criteria, a scenario has to be established. For evaluating the correctness, a use case scenario is applied. Maintainability can be checked by following a change scenario. The selection of a scenario represents the most critical issue, because a wrong scenario does not help to discover deficiencies.

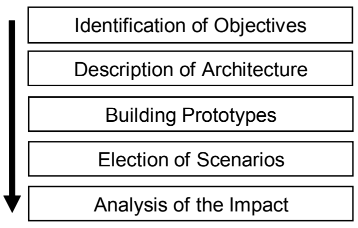


Figure 1. Phases of the scenario-based analysis by ATAM

The Modifiability Analysis Method ALMA [1] was developed based on ATAM, because ATAM is a general evaluation method, and an Impact Analysis requires specialized scenarios. ALMA includes some pre-defined scenarios for an evaluation regarding modifiability and therefore it can be applied here.

By performing the defined procedure by humans, a walkthrough is less objective than a graph-based analysis or a simulation. As a consequence, there are only limited opportunities for tool support and automation. To reduce the subjectivity, it has to be decreased by a proper organization of the team interaction. The effort can be reduced by investigating only the relevant parts of an architecture, affected by a change. However, the changes of concern have to be implemented to some extend to provide a basis for the walkthrough. In terms of effort reduction, only prototypical implementations are developed. In contrast to the aforementioned approaches, the developers can deal with syntactically and semantically incomplete models thus reducing the modeling effort prior to the Impact Analysis.

As an advantage in comparison to the aforementioned approaches, all required quality criteria can be assessed by applying proper scenarios.

According to the risk level, analyses can be performed in a more or less sophisticated way, thus increasing or decreasing the required effort correspondingly.

The scenario-based analysis is able to manage the complexity of architectural decisions in a good way, because abstraction, hierarchical structuring and views can be used both in modeling and in performing the walkthrough. The approach fits well into usual development processes, because the establishment of prototypes can be performed within the iteration cycles.

It can be stated that a scenario-based approach is suitable in the best way for analyzing the impact of alternatives within an architectural decision process. In section 5, some central elements of the ALMA method will be applied in our process.

3.The Decision Theory Based Process

Decision-making for architectural development and reengineering is unable to reduce risks of mission-critical projects if performed in a subjective way and without a methodic procedure. To reduce the risks and to facilitate such decisions, a methodical and rational procedure was developed [2], based on the Decision Theory. Decision Theory facilitates a systematic and rational decision making, especially if competing objectives, side effects and uncertainty have to be considered [8]. Such decision making is broadly applied to solve economic decision problems, e.g., for financial investments. To improve the comprehension of the reader, an overview over the extended decision process is given here. Fig. 2 shows the overall structure of the process. In this section, the phases are explained in brief, and the introduction of the Impact analysis to phase 3 is explained in the next section. Only this phase is affected by the extension presented in this paper.

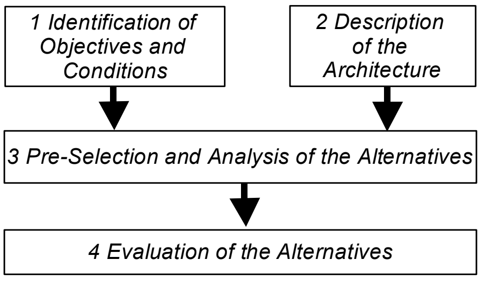

1 Identification of Objectives and Conditions
2 Description of the Architecture
3 Pre-Selection and Analysis of the Alternatives
4 Evaluation of the Alternatives

Figure 2: Phases of the Decision Process [2]

Identification of Objectives and Conditions

At first, all objectives and conditions relevant for a decision have to be identified. They are the central criteria for the whole architectural decision. In the case of competing, uncertain or fuzzy objectives this phase is a critical one.

The objectives are derived from the yet unsatisfied non-functional and functional requirements, which are relevant for an architectural artifact, e.g., a subsystem or a component. In the frequent cases of being incomplete, vague, inconsistent or competing, the objectives are refined and classified into two categories: mean objectives and fundamental ones. The fundamental objectives are the primary ones; they bear a high value for the stakeholder. Means objectives are those contributing to the fundamental ones; their results help to achieve the fundamental objectives. In addition to the classification, the objectives are structured and prioritized. The priorities help to solve conflicts. Furthermore, the priorities are applied for reducing the number of objectives under review, to reduce the overall effort for a decision.

In addition to the objectives, the conditions have to be identified. There are three types of conditions: organizational and technical ones as well as conditions caused to previous decisions. Organizational conditions are defined by an enterprise or by a customer, e.g., enterprise strategies, know-how of the staff, deadlines, budget. Usually they are hard to change. Technical conditions are represent the technical environment a product is developed for or developed in, e.g., standards, tools and protocols. These conditions tend to be variable over time. Conditions of both types are usually identified using checklists. The third type of conditions result from previously made operational or strategic decisions and from their implementation. Both objectives and conditions are in phase 3 applied as the criteria for evaluating of the architectural decisions.

Description of the Architecture

In the second phase, a description of the current state of the software architecture in form of an architectural model is developed. This model serves as the basis for establishing prototypes for the later evaluation of the alternatives. The model includes all parts of the architecture, which are relevant for the actual architectural decision, like components, interfaces and relations. For example, if improving maintainability of the database access is an objective, then all parts of the architecture have to be described, which are related to the database access. The more risky a decision, the more parts of the architecture are important enough to be described. To reduce uncertainty and risks, the described model has to be consistent and clear to understand – with respect to the criteria relevant to a certain decision. Usually, an architectural model is separated into views and described by a set of diagrams, in some cases using an Architectural Description Language (e.g., ACME [9], xADL [10]), or UML.

Preselection and Analysis of the Alternatives

In the third phase of a decision, suitable alternatives are established and their consequences are analyzed. To reduce the effort of analysis and evaluation, the number of alternatives under review is reduced stepwise, while consulting the objectives, the conditions and the risks.

The alternatives for architectural decisions are established based mostly on the experience of the architect. They stem from a variety sources of two main categories: abstract and technological solutions. Abstract solutions as architectural and design patterns, architectural styles or heuristics belong to the first category. These solutions have to be instantiated and implemented to provide a solution and to achieve the objectives.

Technological solutions as the second category consists of reusable building blocks, solution parts from previous projects, third-party components, frameworks as well as technological standards, e.g., component models like those of COM or CORBA.

For each suitable alternative, the consequences according to the objectives and conditions have to be determined. Since the alternatives are not yet implemented at this time, it is only possible to establish a hypothesis or estimate the consequences of each alternative. The decision process [2] suggests a preselection of the alternatives to reduce the effort, and then a detailed estimation of the consequences for the preselected ones. The preselection is based on one hand on the properties and the general quality characteristics of the alternatives. If for example the Design Pattern Façade [14] is an alternative, it influences the architectural quality by increasing the maintainability, because the encapsulation of elements is improved. On the other hand, the alternatives are compared to the organizational and technical conditions. If there is no match, an alternative cannot be applied.

If performed subjectively by the architect, this selection and the estimations are vague and risky. For high-risk decisions, a more detailed analysis is necessary to identify the consequences and the benefits or disadvantages. As a contribution of this paper, this phase is improved by introducing a scenario-based Impact Analysis, as explained in section 4.

Evaluation of the Alternatives and Decision

The fourth and last phase of the decision process is an evaluation of the alternatives by a procedure adapted from the Decision Theory [15]. The several consequences are aggregated and weighted to enable a rational decision. In this way, the risk of subjective decisions is reduced, e.g., if the developer could focus more on positive consequences and would not consider the negative ones.

The evaluation consists in the following steps:

Normalization: In the first step the consequences according to different properties are normalized. A comparison of the consequences is facilitated in this way even if they are of different scales. The consequences are normalized to an interval between zero to one. The best consequences are normalized to one, the worst ones are normalized to zero. If there are only minor differences between consequences, some realistic values for the best and the worst values have to be estimated.

Weighting: In a second step the consequences are weighted according to the relevance of the different properties. Preferred properties are weighted higher.

Calculation of the Expected Value: This third step is performed to take probabilities of the consequences into account. For example, if there is a chance that an alternative requires additional effort, or if there is a risk that a wanted property is not achieved, the probabilities of these different consequences are multiplied with the weighted values to calculate an expected value.

The developer can then select the alternative with the highest value.

4. Typo3 Refactoring Case Study

For illustrating the Impact Analysis extension, a case study is now introduced in brief . The extended method is then described in section 5 using examples form this section.

As case study, a refactoring decision on the Typo3 system is applied. Typo3 is a popular, open source Content Management System for editing and publishing web pages. An author creates web pages by a simple and easy-to-use editor, and the created web page content stored in a MySQL database. If a client requests a web page, the content is retrieved from the database and the webpage is dynamically generated (see fig. 3). Inside the Content Management System Typo3, the Core provides the basic functions to manage the content and to generate the web pages. These basic functions are extended by plug-ins - so-called extensions - e.g. to integrate multimedia elements into the webpage. To store and retrieve the content from the database, a Database Abstraction Layer provides the required functions. Typo3 is a mission critical software system for web-based organizations, which have to share information in a very flexible way, like web-based customer operations.

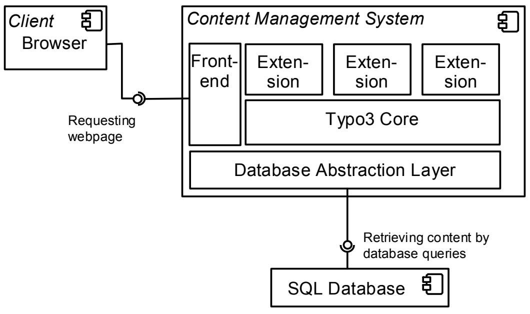


Figure 3. Overview over the Typo3 Architectural Model

4.1. Objectives and Conditions

For this case study decision, four objectives and conditions have to be considered: *portability, performance, minimal implementation effort and avoidance of functional changes*.

For Typo3 version 3.8.1, the *portability* is limited because Typo3 does not support other database management systems (DBMS) than MySQL. This fact limits its applicability if the MySQL DBMS cannot be used because of security reasons or other restrictions. For improving the portability with respect to the DBMS a refactoring has to be performed. The main issue of this refactoring consists in a change of the architecture, so that Typo3 is able to cooperate with other DBMSs. This refactoring bears important risks because changes to the DBMS access affects the whole Typo3 functionality of storing and retrieving content. A detailed analysis of the consequences of the alternatives is necessary for a rational decision-making. Therefore, the extended decision process is applied here. The justification of the additional effort of the Impact Analysis is derived from the risk of failures and unconcerned side effects.

Performance of the DBMS access represent another objective. Usually performance and portability are competing objectives because they demand for contradicting solutions e.g. optimization of data transformation versus layering and encapsulation. Performance is a critical objective, because the storage of the content and the generation of web pages require a lot of database queries, and missing performance of the database access would lead to a delays of providing requested web pages. This would affect the usability and reduce acceptance of the web service and thus of Typo3. Therefore, the time to process database queries must not be lengthened.

4.2. Refactoring Alternatives

Among several possible alternatives available for a restructuring of the database access, the following two alternatives are suitable (see fig. 4 for an overview):

ADODB: The first alternative consists in the application of the ADODB library. ADODB provides functionality to transform SQL statements according to several DBMSs, e.g., to Oracle DBMSs. Even if ANSI SQL is a standard language, there are many proprietary extensions. For example, the data format of time stamps is not standardized and nearly every DBMS uses a different format. Therefore, some SQL statements require a transformation, which is done by the functionality of ADODB.

Wrapper: The second alternative consists in the development of several Wrappers, which are specialized to one DBMS. For example, if Typo3 has to work with the Oracle DBMS, an Oracle wrapper has to be developed and implemented. Such a wrapper includes native database functions, like ora_parse(<SQL-Statement>) for Oracle or the MySQL function, shown in the previous section. Due to space reasons, we decided to consider only two new Wrappers in our case study: one for Microsoft's SQL-Server and one for Oracle products.

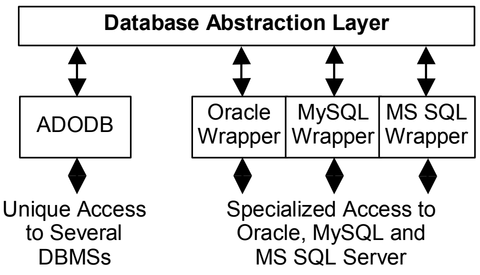

Database Abstraction Layer
ADODB
Oracle Wrapper
MySQL Wrapper
MS SQL Wrapper
Unique Access to Several DBMSs
Specialized Access to Oracle, MySQL and MS SQL Server

Figure 4. Access to different DBMSs by the two alternatives

The evaluation of the two alternatives within the decision process is now performed by the Impact Analysis in the next section.

5. Introducing Impact Analysis for Architectural Decisions

This section introduces a methodical Impact Analysis to the decision process described in section 3. The Impact Analysis extends phase 3 of the decision process (see fig. 2). It performs an analysis of the formerly established alternatives, according to the objectives and conditions from phase 1 of the process, and with respect to the architecture as described in phase 2. By the Impact Analysis, the consequences of the alternatives are determined, which are the basis for the evaluation and decision in the last phase.

The scenario-based approach ALMA was evaluated best in the State of the Art analysis in section 2; only for pure performance analyses a simulation-based approach would be appropriate as well.

ALMA was developed mainly for assessing complete architectures, but here only alternatives for architectural decisions have to be assessed. Therefore only a few core elements of ALMA have to be applied. The analysis of objectives e.g., is already covered by phase 1 of the process (fig. 2). For effort reduction, only a scenario-guided walkthrough of prototypical architectural models is performed. If performance is among the most critical objectives, the architectural models are developed in a way that enables a simulation.

5.1. Building Prototype Models

An Impact Analysis by a scenario-based walkthrough requires an object to investigate. To minimize the implementation effort, alternatives are implemented to certain extend depending on the risk.

Only in the case of high-risk objectives and conditions, alternatives are implemented by evolving the architectural model of phase 2 (see fig. 2); the result is here called a prototype model. To reduce this effort, the prototype models are only developed as necessary for an assessment according to the objectives and conditions of concern. Furthermore, only the few (2 or 3) of the preselected alternatives are developed into prototype models.

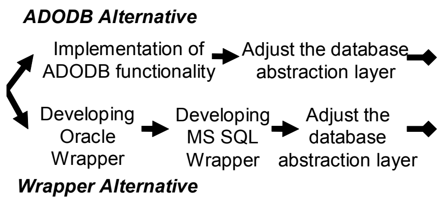
In the case of low risks, only estimations are performed.

Typo3 Case Study:

The preselected alternatives interoperate with the Typo3 Core through the database abstraction layer (see fig. 4). The SQL statements are transformed by one of the two alternatives explained in section 4. Two prototype models are necessary, built by the following activities (see fig. 5):

ADODB alternative: The first step is the implementation of the ADODB library, which can be done as a system extension. The second step are adjustments to the database abstraction layer, so that the SQL commands are rerouted to the ADODB for transformation.

Wrapper alternative: The first both steps are the development of the wrapper for the Oracle and the MS SQL-Server DBMS. The third step is an adjustment of the database abstraction layer in a way, that the SQL commands are directed through the DBMS-specific Wrappers.


ADODB Alternative
Implementation of ADODB functionality
Adjust the database abstraction layer
Developing Oracle Wrapper
Developing MS SQL Wrapper
Adjust the database abstraction layer
Wrapper Alternative
Figure 5. Decision Tree of the implementation activities

5.2. Selection of the Scenarios

According to ALMA, the determination and selection of proper scenarios is performed according to the objectives and conditions. Four scenarios were established:

Criterion	Scenario
Flexibility, modifiability, effort	Change scenarios developed for ALMA
Reliability	Introduction of failures
Interoperation	Typical use cases
Performance, resource consumption	Stress scenario: high volume of requests or data

The set of scenarios is then customized according to the precise objectives and conditions assigned to that particular decision. Furthermore, the scenarios are modified in order to reduce the effort for building the prototype model, e.g. by reducing the scope of a scenario.

Typo3 Case Study:

According to the objectives and conditions, the following scenarios are developed:

Portability: To evaluate the alternatives with respect to portability, a migration scenario is established. The scenario analysis then focuses on the effort, which is required to migrate Typo3 to another DBMS, e.g., FoxPro.

Performance: To check, if the performance has changed, some stress scenarios regarding database access are established. These scenarios are developed from the use case "request webpage" with a frequency of 90 % of the original Typo3 stress threshold.

5.3. Application of the Scenarios and Analysis

The scenarios are now applied to the prototype models of the two alternative. According to the scenario-based approach, this step is performed by walkthroughs.

If performance is among the most critical objectives and the risk level justifies extra effort, the behavior of the prototype model can assessed by a simulation much better than by a walkthrough. To enable a simulation, the prototype model has to be transformed into a proper ADL, e.g., Rapide [16].

Identification of Quality-Related Consequences

For the identification of quality-related consequences, the impact of conditions resulting from the alternative under review has to be considered. For the later decision both, the degree and the probability of the fulfillment of an objective or condition, have to be identified.

Quality-related consequences of an architectural alternative can be determined by considering the positive and negative impact, introduced by properties and the side-effects. They can improve or decay the extend to which objectives are achieved. Major sources for side-effects and properties are the technical conditions to be identified in phase 1 (see fig. 2), resulting e.g., from formerly implemented patterns, styles or used protocols. Each of the identified technical conditions has to be checked, if there is an impact on the objectives. An example is an implementation of the Proxy design pattern [14] to improve the performance of an access to a remote component. A restrictive layer-architecture can hinder these performance improvements.

In the case of a negative impact, the alternatives have to be extended by additional activities to reduce the unwanted impact and to achieve the objectives. If the occurrence of an impact is uncertain, these additional activities are optional. In the latter case, the implementation of the alternative varies, either with or without the optional activities. The probability of the occurrence will later be used to the consequences. The estimation of the probabilities is based both on the experience of the developer and on methods of the Decision Theory [15]. In the case of the Proxy pattern example described above, it could be uncertain, if the layered architecture indeed has an negative impact. Therefore, the implementation of a bridge over the layers would be an optional activity with a certain probability.

Typo3 Case Study:

The identification of the quality-related consequences is determined by applying the scenarios to the prototype models in walkthroughs, as mentioned in section 5.2. For the **ADODB** alternative, the walkthrough shows that the portability is improved, because the flexibility is improved regarding all DBMSs which are supported by ADODB. Additionally, in the walkthrough it was stated that there is a negative impact on the maintainability: future extensions to the DBMS' functionality are not utilizable if they exceed the functionality provided by the ADODB adapter. Furthermore, the possibilities of bug tracking are considered to be reduced, because it is difficult to locate errors. Performance problems are expected due to the additional transformations to SQL commands and due to the fact, that some advanced database functions, e.g., caching mechanisms cannot be used.

For the **wrapper** alternative, the same scenarios are applied. The walkthrough shows that the portability is improved. Similarly to the ADODB alternative, the bug tracking was considered to be limited. As an advantage over the latter it is stated that the DBMS-specific wrappers make advanced database functions available and that additional transformation steps are not required. Therefore the performance should not affected, compared to the unchanged Typo3.

The analysis results of the quality-related consequences are shown in fig. 6. Grades from 1 (best) to 5 (worst) are used to quantify the extent the objectives are achieved. The consequences of the ADODB alternative are evaluated for portability to grade 1 and for performance to grade 3. The consequences of the wrapper alternative are evaluated both for portability and performance to grade 1.

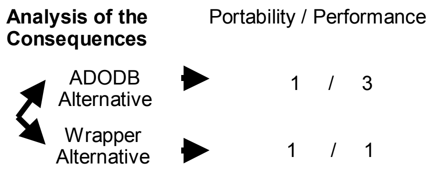


Figure 6. Quality-Related Consequences as Decision Tree

Identification of Functional Changes

Furthermore, the functional behavior of the future architecture has to be checked, either if it remains unchanged in the case of refactoring or if the desired behavior is achieved in the other cases. These checks are performed by walkthroughs based on use case scenarios. If there are unwanted or missing functional changes, the alternative has to be modified.

Typo3 Case Study:

As typical for refactoring activities, changes to functional behavior are unwanted. For the case study no additional scenarios are applied, but during the walkthrough mentioned above the issue of functional changes is analyzed. As typically for adapters, the **ADODB** alternative hides special database functions from use, for example the searching algorithms of MySQL. As a consequence, some search operators, e.g., NEAR, cannot be used. This leads to a change of the functional behavior. To make such functions to operate, additional refactoring steps have to be performed. Potentially, there are more specific database functions, which have to be supported. However, exploring this issue would require a detailed source code analysis, which is not covered by this architectural Impact Analysis itself. The developer estimates the probability of these additional steps by 40% (see fig. 7). The quality-related consequences would not be affected if such additional steps would be performed.

The **wrapper** alternative is functionally comparable to the original database access in Typo3 version 3.8.1. If there are no implementation errors of the wrappers, the functional behavior should be unaffected.

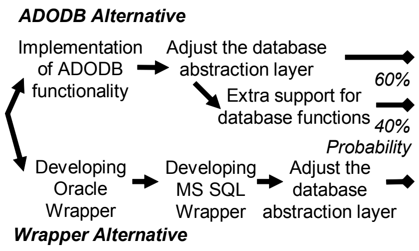

ADODB Alternative
Implementation of ADODB functionality
Adjust the database abstraction layer
60%
Extra support for database functions
40%
Probability
Developing Oracle Wrapper
Developing MS SQL Wrapper
Adjust the database abstraction layer
Wrapper Alternative

Figure 7. Activities and Consequences Concerning the Functional Behavior as a Decision Tree

Estimation of the Implementation Effort

For effort as a critical condition, an effort estimation is carried out during the third phase of the decision process. Efforts for the later implementation of an alternative, for changes and for a necessary modification of the rest of the architecture can be performed by developers during the elaboration of the prototype model and during the walkthrough.

Typo3 Case Study:

The **ADODB** alternative requires are two major and one optional development activity. Furthermore, version conflicts with the remaining development activities of the Typo3 core will likely occur and have to be managed. An effort of 35 person-days is estimated. In the case that specific database functions have to be supported, the effort rises to 45 person-days.

The implementation effort of the **wrapper** alternative is higher, because each wrapper has to be developed and tested separately. Similarly to the other alternative, version conflicts will likely occur. An effort of 50 person-days is estimated.

5.4. Typo3 Case Study Results: Evaluation of the Alternatives

The Impact Analysis provides the required data for the evaluation of the alternatives. To conclude the case study, the succeeding steps of normalization, weighting and calculating of the expected value is described here briefly.

Normalization: The best alternative requires an effort of 35 person-days, the value of the corresponding consequence is normalized to 1. The best alternatives concerning portability and performance are of grade 1, their corresponding consequences get value 1. The worst alternative concerning effort requires 50 person-days, its consequence is normalized to 0. The worst alternative concerning performance is of grade 3, but an even more worse performance of grad 5 is possible. Therefore, the consequence is normalized to the value 0.5. In the case of the additional refactoring activities of the ADODB alternative, the effort rises to 45 person-days. This effort is near the worst consequence, so we normalize it to 0.3.

Weighting: In our case, a high portability and a low implementation effort are most important. Therefore, the weighting factors are determined to 0.4 for the portability and 0.3 for the effort since Typo3 is not a real-time application but performance is important, the weighting factor is defined to be 0.3 for the performance. The normalized consequences (see fig. 8) are multiplied by these weighting factors and added together. The total value of the ADODB alternative is 0.85 in the first case without additional refactoring activities. In the second case with extra activities for refactoring additional database functions, the resulting total value is 0.64. The total value of the wrapper alternative is 0.70.

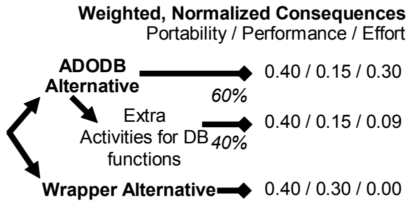

Weighted, Normalized Consequences
Portability / Performance / Effort
ADODB Alternative
0.40 / 0.15 / 0.30
60%
Extra Activities for DB functions
40%
0.40 / 0.15 / 0.09
Wrapper Alternative
0.40 / 0.30 / 0.00

Figure 8. The Weighted Consequences

Calculation of the Expected Value: The consequences of the first variant of the ADODB alternative are multiplied with its probability of 0.6; the consequences of the second one are multiplied with 0.4. The results are added to get the expected value of 0.77. for the wrapper alternative there are no probabilities.

Finally, both alternatives are evaluated. The ADODB alternative gets a total value of 0.77, the wrapper alternative one of 0.70, thus there is only a minimal difference. The rational decision consists in a preference of the ADODB alternative against the wrapper.

6. Summary and Conclusion

In this paper, an architectural decision process is extended by a methodical Impact Analysis, in order to reduce risks by more systematic and confident decisions . The architectural decision process was previously improved by introducing elements of decision theory [2]. Risk reduction is achieved by methodically guiding the decision maker in the evaluation of the consequences of alternatives of an decision. For this evaluation, a scenario-based approach has been chosen to analyze the consequences of the alternatives according to systems behavior, quality and effort. This scenario-based Impact Analysis adopts elements of the ALMA method for architecture assessment. The phases of the extended decision are described in detail and the application of process is shown by a refactoring case study.

The application of the scenario-based Impact Analysis facilitates the decision-making regarding the architectural alternatives by evaluating the consequences of each alternative in a methodical, rational way. Without this Impact Analysis, no guideline for performing evaluations about the consequences was given, with many cases of subjective evaluations and remaining uncertainty as a result. By the introducing of the Impact Analysis step, the risk of side effects, additional effort and a changed behaviour is significantly decreased.

Nevertheless, the application of the Impact Analysis as well as the whole decision process introduces some additional effort. Therefore, the risk of each architectural decision has to be investigated if it justifies that effort. Examples for risks are missing flexibility of a system with increased time to market and effort, missing integration and collaboration opportunities to neighbouring systems, tremendous additional effort for later refactoring due to wrong basic architectural decisions. The issue of effort is tackled by suggesting different levels of analysis and decision effort according to the level of risk.

7.Acknowledgements

We wish to thank our colleagues Alexander Pacholik and Patrick Maeder for their helpful comments to an earlier version of this paper.

8. References

[1] P. Bengtsson, N. Lassing, J. Bosch and H.v. Vliet, *Architecture-level modifiability analysis (ALMA)*, The Journal of Systems and Software, Vol. 69, 2004, pp. 129-147.

[2] S. Wohlfarth and M. Riebisch, *Evaluating Alternatives for Architecture-Oriented Refactoring*, Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2006), IEEE Press, Los Alamitos (CA), 2006, pp. 73-79.

[3] M. Fowler, K. Beck and E. Gamma, *Refactoring: Improving the design of existing code*, Addison-Wesley, Boston, 2005.

[4] Robert S. Arnold and S.A. Bohner, *Software Change Impact Analysis*, IEEE Computer Society Press, Los Alamitos, CA, 1996

[5] S.L. Pfleeger and S.A. Bohner, *A Framework for Software Maintenance Metrics*, IEEE Transactions on Software Engineering, 1990, pp. 320-327.

[6] R. Kazman, M. Klein and P. Clements, *ATAM: Method for Architecture Evaluation*, Technical Report CMU/SEI-2000-TR-004 ESC-TR-2000-004, http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf, 2000.

[7] M.A. Babar, I. Gorton, *Comparison of Scenario-Based Software Architecture Evaluation Methods*, 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004, pp. 600-607

[8] E. Forman and M.A. Selly, *Decision By Objectives*, World Scientific Publishing Company, 2002.

[9] D. Garlan, R.T. Monroe und D. Wile, *Acme: Architectural Description of Component-Based Systems*, In: Gary T. Leavens und M. Sitaraman (eds.), *Foundations of component-based systems*, Cambridge University Press, New York, pp. 47 – 67.

[10] E.M. Dashofy, A.V.d. Hoek, R.N. Taylor, *A Highly-Extensible, XML-Based Architecture Description Language*, Proceedings of the IEEE/IFIP Working Conference on Software Architecture (WICSA'01), IEEE Computer Society, pp. 103 – 112.

[11] C. Hofmeister, R. Nord and D. Soni, *Applied software architecture*, Addison-Wesley, Reading (Mass), 2000.

[12] N. Medvidovic, *ADLs and Dynamic Architecture Changes*, Joint Proceedings of the SIGSOFT'96 Workshops, ACM Press, 1996, pp. 24-27.

[13] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product line approach*, Addison-Wesley, Harlow, 2000.

[14] E. Gamma, R. Helm, R. Johnson and J. Vlissides: *Design Patterns – Elements of reusable object oriented Software*, Addison-Wesley, Munich, 2004.

[15] R. Clemen and T. Reilly, *Making Hard Decisions with Decision Tools*, Pacific Grove, Brooks Cole, 2004.

[16] Rapide Design Team: *Guide to the Rapide 1.0 - Language Reference Manuals*, URL: 'http://pavg.stanford.edu/rapide/lrms/overview.ps', last visited 2006-09-10.