Alignment of Software Specifications with Quality- and Business Goals in the SIKOSA Methodology

Andrea Herrmann[1], Daniel Weiß[2]

[1]Software Engineering Group, Faculty of Mathematics and Computer Science
University of Heidelberg
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
herrmann@informatik.uni-heidelberg.de

[2]Universität Hohenheim
Forschungszentrum Innovation und Dienstleistung (795)
70593 Stuttgart
daniel.weiss@uni-hohenheim.de

Abstract: Business-IT alignment for software specifications means that the specifications have to be aligned with business goals. In the SIKOSA research project, we developed the SIKOSA methodology which supports the integrated assurance of quality during the whole software development process. In the present work, we present these aspects of the SIKOSA methodology, which especially align specification decisions to quality goals and indirectly to business goals. Such goals play a role in the following activities: the derivation of software requirements from quality goals, the prioritization of these software requirements, and the definition of decision criteria for architectural design decisions. The results of these three activities influence architectural decisions.

1 Introduction

Business-IT alignment for software development means that the software (and all other artefacts) have to be designed in a way to support the business goals. Assuming that the software functions as specified, the software specifications have to be aligned with the business goals as well. Specifications are the result of a complex decision-making process which involves a variety of interdependent decisions on different levels of specification granularity, involving diverse stakeholders. Therefore, we here treat the question how decisions concerning specifications can be consistently aligned to business goals during different phases of the software development process.

The SIKOSA methodology supports the integrated assurance of quality and of Business-IT alignment during the whole software development process. This methodology has been published in [HPK06] and [WKK07]. In the present work, we highlight these aspects of the SIKOSA methodology which align specifications to quality- and business goals. The SIKOSA methodology consists of several modules: ProQAM (Process-oriented Questionnaires for Analyzing and Modeling Scenarios) [DOK05], TORE (Task

Oriented Requirements Engineering) [PK03], MOQARE (Misuse-oriented Quality Requirements Engineering) [HP05], [HP07] and ICRAD (Integrated Conflict Resolution and Architectural Design) [HPP06].

Software properties and how well they are aligned with the business goals are defined by the decisions made during the software specification process. Decisions and specifications can belong to the problem space or to the solution space [HPP06]. In this spirit, the requirements specification belongs to the problem space and describes the needs, while the architectural design (specification) belongs to the solution space and describes what will be implemented. In the problem space, decisions are made during the following activities: the software requirements specification, the definition of decision criteria for architectural design decisions and the prioritization of software requirements. (Priorities support many decisions like conflict solution or test decisions.) The results of these three activities influence the fourth activity: architectural decisions in the solution space.

In the SIKOSA methodology, these activities produce the following artefacts (Figure 1):

1. Software requirements (here: MOQARE countermeasures) are derived from business goals by ProQAM and MOQARE.

2. Decision criteria for architectural design decisions with ICRAD are derived from business goals.

3. Software requirements priorities are attributed to the software requirements, taking into account the ICRAD decision criteria and the business goals.

4. Architectural design decisions are made with ICRAD.

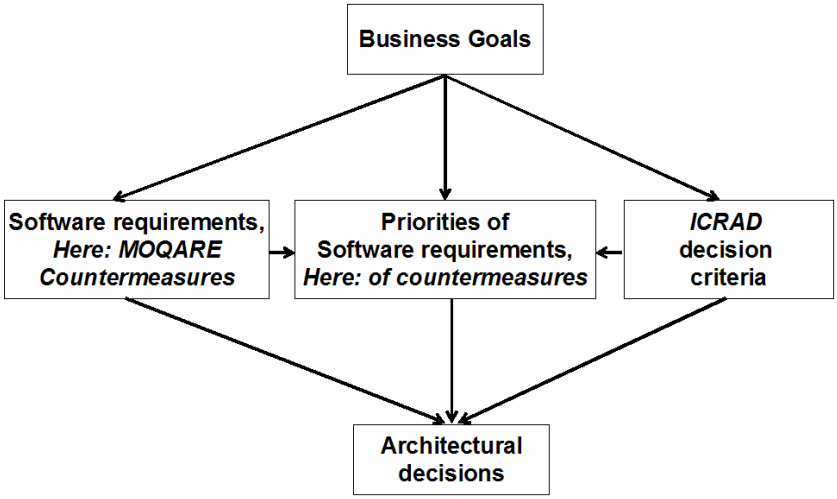

Figure 1: Business goals indirectly influence architectural decisions via three intermediate artefacts; the arrows signify relationships of the type “influences” between the artefacts

Section 2 cites related work. The subsequent sections treat the four above mentioned activities: Section 3 describes how countermeasures are derived from goals by MOQARE, section 4 discusses ICRAD´s architectural design decision criteria, section 5 treats requirements prioritization and section 6 presents how architectural decisions in ICRAD are indirectly aligned with goals, when they are based on software requirements, their priorities and the architectural decision criteria as defined in the preceding sections. Section 7 summarizes this article.

2 Related Work

Business goals are "high-level reasons for getting the new product" [La02] and a "non-operational objective to be achieved by the […] system" [DVF93]. A lot of research activities focus on the business goals of software systems, projects or organizational units dealing with their classification and identification. Business goals can be categorized according to the five dimensions: product size, quality, staff, cost, and (calendar) time [Wi02]. Orthogonally to these dimensions, business goals can be classified according to the four perspectives of the Balanced Scorecard [KN92]: financial, customer, internal processes, learning & growth. For details, we refer the interested reader to the relevant business literature.

Software requirements and software requirements decisions can be described on different levels of granularity and with different focus. Aurum et al. [AWP06] distinguish four levels of requirements decisions: business, stakeholders, project, and product level. Lamsweerde et al. [La01] discern business goals, project goals, and software system goals.

The goal-oriented requirements engineering methods [La01] have been using software (product) goals successfully as a starting point for software requirements specification. In [He07], we have discussed how goal-orientation and hierarchical top-down detailing from goals to software requirements ideally supports decision-making during requirements elicitation. Other authors also emphasize the importance and multiple roles of goals for requirements elicitation, alignment of requirements with business goals, requirements validation, conflict solution and architectural design [YM98], [RS05]. Nevertheless, no integrated method for the alignment of all types of decisions with the business goals exists.

It seems logical to derive software requirements from business process requirements. Nevertheless, there are only few approaches to do so [BE01], [KL06a]. Business process modelling and software requirements modelling still use different notations and semantics. Approaches to their integration are presented by [SH00], [No04], [BCV05], [KL06b]. However, some weaknesses of the integrations remain [BE01] [KL06a]. Especially former work concentrates on functional requirements (FR). Non-functional requirements (NFR) are neglected, although they are gaining more and more relevance, as the competition on the market cannot be won by a software´s functional scope alone, but also quality is crucial.

3 Derivation of software requirements from software goals

The distinction between business goals and software goals is also made in the SIKOSA methodology. Software goals can be functional or non-functional goals. In the SIKOSA methodology, the functional goals are described by the business processes to be supported, while the non-functional goals are called quality goals.

We integrate the ProQAM business requirements modelling with software requirements specification based on quality goals and countermeasures (NFR described with MOQARE) and use cases (FR described with TORE). Usually, goal-oriented analyses proceed from high-level goals down to requirements [He07]. This is supported systematically by the modules of the SIKOSA methodology, as presented in Figure 2.

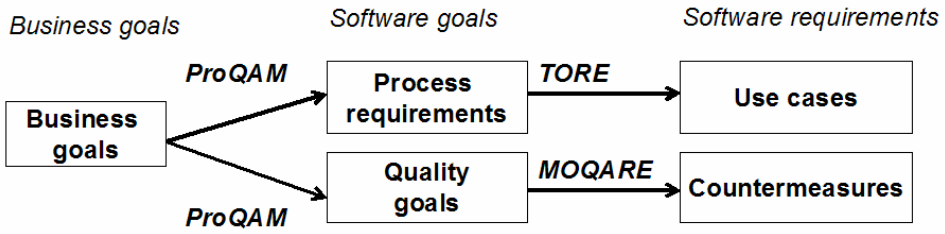

Figure 2: ProQAM, TORE and MOQARE derive software requirements from business goals

The five concepts shown in Figure 2 are defined as follows:

- Business goal: ProQAM identifies the stakeholders´ business goals. These can be formal or technical and express situations to be achieved and results respectively modes of action by means of decisions.

- Process requirements: The process requirements describe the process which is to be executed. Not only does it contain the steps which are to be supported by software, but also staff needed or relevant competences. In ProQAM, such elements are described by central constructs of event-driven process chains (EPCs [Sc01]). The central element, the function, is defined in a way to support the business goals.

- Use case: Use cases [Co01] describe the requirements for the interaction between user (or other, maybe non-human actors) and the software, including pre-conditions, interaction steps and post-conditions. They can be derived from the process requirements. Deriving FR from business processes in the form of such use cases is supported by the method TORE.

- Quality goal: A quality goal is a goal which is to be satisfied by the software and therefore is a high-level NFR. In MOQARE, quality goals are expressed by the combination of an asset plus a quality attribute, like “usability of the user interface”. An asset can be any protectable part of the system. A quality attribute describes an aspect or characteristic of quality.

- countermeasure: A countermeasure is an operational requirement which supports the quality goal. Countermeasures can be FR, exception scenarios of use cases, NFR constraining use cases, architectural constraints, user interface constraints, constraints on project and software development, constraints on administration or maintenance, or another quality goal.

The five concepts above describe desired properties of the business, the business processes and the software. From the security field, the idea of negative, undesirable concepts has been adopted in the SIKOSA methodology. The most famous concept based on this principle is the misuse case [SO00], [SO01], [Al02]. Like use cases, misuse cases describe the interaction of the software system with an actor, but misuse cases describe unwanted scenarios (e.g. attacks, user errors, accidents) which threaten goal satisfaction. Misuse cases help to define and to complement the software requirements and also to document the justification of these requirements. This principle is used in the SIKOSA methodology with respect to business goals and quality goals.

For lack of space, here, we only focus on the realization of IT-alignment in the SIKOSA methodology. For a complete description of the methods, we refer the reader to the publications cited in the introduction. In the remainder of this section, we describe how countermeasures are derived from functional and non-functional process requirements by MOQARE. These process requirements are output of ProQAM[1].

To illustrate our methods, we describe a case study performed during the Sysiphus enhancement[2]. Sysiphus is a tool which is developed and used at the University of Heidelberg and the Technical University of Munich to teach software engineering and to document the results of case studies [Sy07]. Sysiphus implements TORE and MOQARE. It also supports design according to Brügge and Dutoit [BD04] and ICRAD. The case study objectives were: We wanted to test and to measure the usability of Sysiphus and to propose requirements on potential improvements. These requirements had to be prioritized in order to be integrated into plans for the further enhancement of Sysiphus. Finally, a workshop was held to discuss strategies of how to implement the improvements and a decision was made.

To meet these objectives, this case study included the following steps:

1. definition of the usage context and the business goal
2. description of the FR
3. detailing of the quality goal „usability of the user interface“ and derivation of countermeasures, in order to define what usability means for this system
4. definition of the reference system, then benefit estimation for the FR and countermeasures
5. usability test and evaluation of the software to measure how well the countermeasures and the quality goal “usability” are satisfied

[1] We want to remark that one of the strengths of the SIKOSA methodology is that it integrates modular methods which can be applied independently of the others as well as in combination.
[2] Further MOQARE case studies have been published here: [HRP06], [HKD07], [HP07]. However, most industry case studies we performed are confidential.

6. prioritization of the countermeasures for release planning
7. decision on implementation alternatives

The results of the steps 4 to 6 are presented in section 5, and step 7 in section 6.

Step 1: We restricted the scope of the analysis to the requirements engineering (RE) and architectural design (AD) parts of Sysiphus. Their business goal is "efficient support of RE and AD". The analysis started with the quality goal "usability of user interface", which in a former analysis (not presented here) had been identified to contribute to this business goal. We assumed a usage context where Sysiphus is applied in a small company by ten IT professionals. They are irregular users, had only short Sysiphus training and are offered no helpline support. They must use the tool during RE and AD.

Step 2: The FR supported by the RE & AD part of Sysiphus are described by 27 use cases, such as "specification of misuse cases" or "review of design".

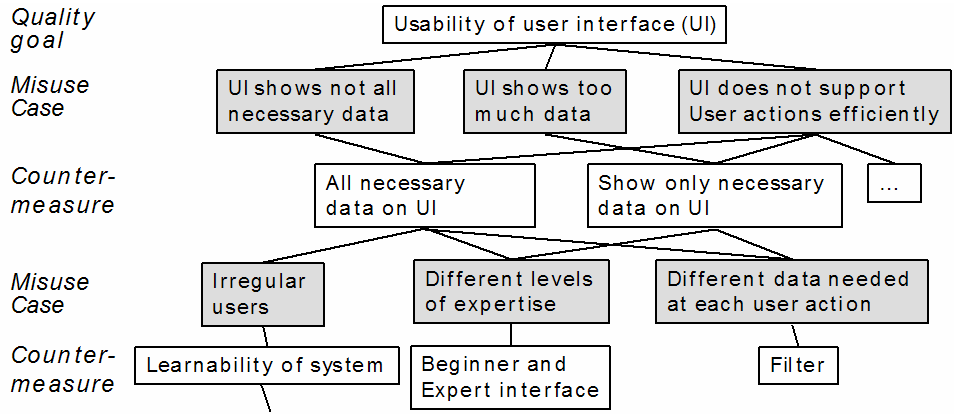

Figure 3: Section of the Misuse Tree resulting from the case study

Step 3: From quality goals, MOQARE derives misuse cases and countermeasures. The misuse cases threaten the quality goals. A countermeasure reduces the probability of a misuse case or reduces its predicted negative consequences. By analyzing the quality goal with MOQARE, 22 misuse cases and 31 countermeasures were identified. Two ISO standards [ISO13], [ISO92] supported the identification of usability requirements, which then were chosen and detailed specifically to the context and its needs. Figure 3 shows a section of the resulting Misuse Tree. System specific misuse cases and countermeasures should be worded in a way to apply to the 27 use cases individually, but we did not do so here because so many details would have complicated the Misuse Tree and all later treatments of the countermeasures.

4 ICRAD decision criteria for architectural decisions

As business-IT alignment is our objective, the decision criteria for architectural decisions have to be defined in a way to support the business goals. The business goals usually can not be used as decision criteria directly. For instance, it might be difficult to estimate

how well an architectural solution supports the business goal “high market share” or “efficient process support”, as their satisfaction does not depend on the software alone. It is easier to predict how well the quality goal “usability of user interface” is supported.

Frequently used decision criteria for architectural decisions are benefit, cost, complexity and risks, or combinations of these factors, like net value and benefit-cost-ratio [XMC04] [KAK01], [IKO01]. Therefore, in ICRAD these are the four standard evaluation criteria for architectural alternatives (see section 6). If necessary, ICRAD can be adapted in order to use other or additional criteria, like the satisfaction of goals [GY01], of non-functional goals [KAK01], [IKO01] or of functional goals [CB95], [KAB96]. But usually, if the benefit of these goals is known, their satisfaction in taken into account by considering their contribution to the benefit.

5 Prioritization of requirements

In [HPP06], we discussed that some requirements conflicts can only be solved in the solution space, by selecting one of the architectural alternatives. However, some decisions about requirements can be solved in the problem space, based on requirement priorities. Such priorities support decisions like: One out of three types of requirements conflicts can be solved within the problem space [HPP06]. Presorting of requirements is useful. Davis calls it “requirements triage” [Da03] and also the “Planning Game” of Extreme Programming [Be00] classifies requirements according to which ones have to be implemented, which can be postponed and which have to be analyzed in more detail. Such a classification facilitates other decisions like release planning. During architectural decisions, must-requirements can be an exclusion criterion: Those architectural alternatives which do not satisfy the must-requirements will not be considered further. Some decisions later in the software development process can use requirement priorities, like testing (where requirements priorities support test case prioritization) or – as in the case study – the assessment of the overall level of quality.

Step 3 of the case study identified requirements which Sysiphus should satisfy in order to support the quality goal “usability of user interface”. Some countermeasures are currently not satisfied and others are (at least partly) satisfied by Sysiphus. As we have shown in [He07], different decisions can demand different criteria. Therefore, the architectural design decision criteria from section 4 cannot automatically be used for the requirements prioritization. In the case study, we in fact used two prioritization criteria: With respect to the overall assessment of the usability of Sysiphus, our main criterion for requirements prioritization was its benefit relative to the usability quality goal. For the planning of later software releases it was important whether and how well a countermeasure is already satisfied; its implementation cost also played a role.

In MOQARE, we derive a countermeasure´s benefit from the risk reduction which it causes with respect to the misuse case risk[3]. Among misuse cases and countermeasures,

[3] Misuse Case risk is defined as the product of probability and caused damage [ISO02], [XMC04].

n-m-relationships can be found, when a misuse case is counteracted by several countermeasures, and when countermeasures work against several misuse cases. For instance, there are several countermeasures against the misuse case "user interface does not support user actions efficiently".

Common methods for requirements prioritization[4] do not consider dependencies among the benefits of requirements at all or only superficially. In reality, however, such dependencies are frequent and critical. For instance, countermeasures can replace each other partly, when they mitigate the same misuse case. Or countermeasures may need each other for being effective against the same misuse case. We take into account such dependencies by bundling requirements and by relating all estimations to a reference system [HP06]. In many prioritization methods, it is common to bundle those requirements which depend on each other most to relatively independent bundles[5]. The reference system is the idea of a set of requirements which are imagined to be implemented. If perfect quality is the benchmark, the perfect system is the reference, i.e. a system in which all requirements are implemented [XMC04]. The reference system can also be the ensemble of all mandatory requirements [REP03], the former system version or a competitor´s software product.

When estimating a countermeasure´s benefit relative to a reference system, the risk of the corresponding misuse case(s) is estimated twice: Firstly, the "reference risk" in the reference system is estimated, secondly the "varied risk" if this countermeasure is not implemented or if it is implemented additionally. The benefit achieved by a countermeasure in relation to a misuse case equals the risk reduction [AH04], [XMC04].

This said, we can continue with the case study´s *Step 4:* The reference system was defined to support all the FR identified in step 2 plus all countermeasures defined in step 3. This means that our benchmark is the system with perfect usability. The benefit of this reference system is set to 100 benefit points. This benefit is defined to be achieved by the satisfaction of the FR alone. Then, the satisfaction of the usability goal does not add direct benefit, but only prevents risk. We use the unit "benefit points", because it is difficult in an example with fictitious usage context to estimate benefits in Euro.

The FR benefits were defined on two levels of granularity. On a high level, we identified three FR bundles defined according to the three methods supported. We assumed these bundles to be independent and simply distributed the 100 benefit points. On the use case level, within each bundle use case benefits were estimated.

For each misuse case – countermeasure pair, two risks were estimated as described above. Probabilities were estimated in percentage and damages in benefit points relative to the total system benefit of 100. Resulting benefits for the most important and some

---

[4] Such methods are the analytic hierarchy process (AHP) [Sa80], [KWR98], numeral assignment [Ka96] or cumulative voting (CV), also called "$100 test" [LW00], [BJ06]. According to [HP06], all methods which attribute one fixed priority value to each requirement can be said to neglect dependencies.

[5] These groups are then called features [RHN01], [Wi99], feature groups [RHN01], super-requirements [Da03], classes of requirements [REP03], bundles of requirements [PSR04], categories [XMC04], User Story [Be00], super attributes [SKK97] or Minimum Marketable Features [DC03].

less important countermeasures are shown in Table 1. The countermeasure "all necessary data on user interface" (which is a quality goal itself and further analyzed in Figure 3), refers to two misuse cases. In the reference system, the risk of both misuse cases is supposed to be 0. If the countermeasure was not implemented, then the user – as a workaround - can open several Sysiphus windows and this way get all necessary data. However, this does not work for all user actions and it is inefficient. The misuse case "User interface does not show all necessary data" causes a damage of 100 points, because it makes the system useless. However, this happens only in an estimated 40% of the user actions. Therefore, its varied risk without the countermeasure being implemented is 40 points. Without the countermeasure, the other misuse case – "the user interface does not support the user efficiently" - is true to 100%. As the users are obliged to use the system and because they are IT professionals, who can handle two windows on their screen, the damage was estimated with only 10 points (the value of loss of productive work time). Assuming that both misuse cases are independent of each other, the countermeasure´s benefit then is 40+10=50 benefit points. As can be seen in the table, all other countermeasures have a much lower benefit. There was no other misuse case in the analysis which caused such a high damage.

Table 1: Countermeasure benefits resulting from the case study (Remark: These benefits are specific to the case study and not generally valid.)

Benefit (in benefit points)	**Countermeasure**
50	All data necessary for one user action must be presented at the same time.
2.0	At any time, the currently executed user action must be obvious to the user.
1.8	The system allows filtering of data.
1.4	Automated check whether input data are within the valid range
1.1	Context sensitive help for any screen and data field
1.02	User training
1.0	Success notification after completion of each user action
1.0	Support of users for doing the user actions in the right order
1.0	Explanations on user interface + self-explanatory names
…	
0.1	The system allows to adapt the size of the work space.
0.0875	Data fields are initialized with default values.

Step 5: Usability test: To save time during the case study, we did not specify detailed test cases for evaluating the current satisfaction of the usability requirements by the system. Instead, we executed the 27 use cases as defined in step 2 and assessed how well each of them satisfies each of the countermeasures. The results of these tests were entered in a spreadsheet table where each column corresponds to a use case and each

row to a countermeasure. These results x_{ij} measure the degree of satisfaction of a countermeasure i during the execution of a use case j between 0 (not satisfied at all) and 1 (totally satisfied). These tests were performed by two testers and the results were discussed afterwards to obtain a shared judgement.

The satisfaction of each countermeasure i was calculated as weighted sum $x_i = \Sigma_j (x_{ij} \cdot$ *benefit* of use case j). If all countermeasures were satisfied, the total system benefit would have been 100 points. As some were only partly satisfied, the total usability risk (benefit loss) was the weighted sum = Σ_i [(1- x_i) · (*benefit* of countermeasure i)]. This risk was 18 points and consequently the effective benefit of the system 100-18 = 82. (We must remark here that we were very strict when evaluating the software!) This value will be especially interesting when we will re-assess the usability after a system enhancement to measure the usability improvement quantitatively.

Step 6: Countermeasure prioritization for release planning: For those countermeasures which are not yet satisfied to 100%, the cost of doing so was estimated in 1, 2 or 3 cost points. The priority of a countermeasure i with respect to release planning was defined to be proportional to "(1-x_i) · *benefit* of countermeasure i". Those countermeasures with the highest priority and those with cost = 1 were candidates to be scheduled for the next release.

6 Architectural decisions which are aligned with goals

ICRAD is an iterative and integrated process for the solution of requirements conflicts and for architectural design. In this section, we describe how architectural alternatives are compared and how the decision is made. Decisions among two or more alternatives and their justifications are documented in the template shown in Table 2. Each alternative is evaluated with respect to its benefit, risk, implementation cost and complexity cost. The reference system can be different for each decision, as it is modified by the decisions made before[6]. The *benefit* of an alternative is not equal to the sum of the benefits of the requirements realized by this alternative, due to dependencies. The *risk* of an alternative includes risks provoked by realizing risky requirements or provoked by the architectural alternative, as well as risks provoked by not realizing some countermeasures. *Cost* of implementation ideally is estimated in the same unit as the benefit, in order to be comparable. *Complexity* includes architectural and organizational complexity and will lead to maintenance and other cost. For being comparable to the other criteria, complexity is transformed into complexity cost. Complexity is caused by software complexity, e.g. by coupling of its components [KAB96], [CB95], [LRV99] and also by the complexity of the software´s integration into its environment. These estimations are done for both (respectively all) alternatives of the same decision and their results are documented in Table 2.

The *total benefit* is calculated as *benefit* minus *risk*. *Total cost* includes implementation and complexity *cost*. Two decision criteria are:

- net value = *total benefit* minus *total cost*

[6] This – together with requirements dependencies – is why the requirements benefits estimated in section 4 cannot be used directly here.

- Benefit-cost-ratio = *total benefit* / *total cost*

If the more expensive solution has a lower benefit, then it is logical to choose the cheaper and better solution. However, very often, the alternative with the higher benefit is the more expensive one, as is also the case in our case study. The value *[(B2-R2)-(B1-R1)] / [(CC2- CC1)+(C2-C1)]= ΔTB/ ΔTC* (see Table 2, in the lower right field) has shown to be a third good decision criterion. To interpret this value, different cases are to be distinguished:

- If *ΔTB/ ΔTC<0*:
  - If *ΔTB<0* and *ΔTC>0*, then the total benefit of alternative 1 is higher and the total cost below that of alternative 2, and alternative 1 is chosen.
  - If *ΔTB>0* and *ΔTC<0*, then the opposite is true and alternative 2 is chosen.
- If *ΔTB/ ΔTC>0* because both *ΔTB* and *ΔTC* are positive
  - and the absolute value of *ΔTB/ ΔTC>1*, then alternative 2 is chosen.
  - otherwise, the alternative with the higher benefit-cost-ratio is chosen.
- If *ΔTB/ ΔTC>0* because both *ΔTB* and *ΔTC* are negative
  - and the absolute value of *ΔTB/ ΔTC>1*, then alternative 1 is chosen.
  - otherwise, the alternative with the higher benefit-cost-ratio is chosen.

The three criteria above do not always lead to the same decision. How to proceed if they are in favour of different decisions is described elsewhere [HPP06].

Table 2: Template table used to compare alternatives in ICRAD.

	Alternative 1	**Alternative 2**	**Difference**
Cost	C1	C2	C2-C1
Complexity Cost	CC1	CC2	CC2-CC1
Risk	R1	R2	R2-R1
Benefit	B1	B2	B2-B1
Total benefit	B1-R1	B2-R2	(B2-R2)-(B1-R1)
Total cost	C1+CC1	C2+CC2	(CC2- CC1)+(C2-C1)
Net value	(B1-R1)- (C1+CC1)	(B2-R2)- (C2+CC2)	(B2-R2)-(C2+CC2) -(B1-R1) +(C1+CC1)
Total Benefit/ total cost	(B1-R1) / (C1+CC1)	(B2-R2) / (C2+CC2)	[(B2-R2)-(B1-R1)] / [(CC2- CC1)+(C2-C1)]

In the case study, we had identified a multitude of countermeasures which signify improvement ideas. One might have realized them in a series of subsequent releases improving the user interface´s usability incrementally. We also considered re-designing the user interface. This decision was fundamental and was discussed in a workshop of several hours with about ten participants. The workshop started with a discussion of the countermeasures. Then, architectural alternatives were identified. Without going into detail, we want to present the decision between two alternatives. Although in the preceding sections, a countermeasure´s benefit was a main criterion for its prioritization,

now the default criteria of ICRAD have all been taken into account, because the implementation cost, complexity cost and risks caused by a solution also play a role for the decision for or against the one or the other type of project. The must-requirement "All data necessary for one user action must be presented at the same time." (from Table 1) was realized in both alternatives. The other countermeasures were considered indirectly by estimating the benefit on the basis of which countermeasures can be realized by each of the alternatives. The benefit of a requirement was again measured in "benefit points", relative to the 100 value of the perfect system. Benefit should ideally be comparable to cost, yet in this case study they were not. The cost of each alternative here was estimated in person months (unlike cost estimation for individual requirements in step 6).

Table 3: Comparison of alternatives in the case study; "PM" stands for "person months" and "BP" for "benefit points"

	Alternative 1: incremental improvement	**Alternative 2: re-design**	**Difference (Alternative 2 – Alternative 1)**
Cost	3 PM	6 PM	3 PM
Complexity Cost	2 PM	1 PM	-1 PM
Risk	0 BP	2 BP	2 BP
Benefit	6 BP	14 BP	8 BP
Total benefit	6 BP	12 BP	6 BP
Total cost	5 PM	7 PM	2 PM
Net value	6 BP – 5 PM	12 BP – 7 PM	6 BP – 2 PM
Total Benefit/	1.20 BP/ PM	1.71 BP/ PM	3 BP/ PM

As can be seen in Table 3, the re-design has higher implementation cost than the incremental improvement, but lower complexity cost because it reduces the software´s complexity. (Or rather: the next release was defined in a way that at realistic cost a good improvement could be attained without any risk. A larger release would have caused more cost without significantly higher usability improvement.) It can be expected that the re-design achieves a much higher improvement of the usability and therefore more benefit, but also includes the risk to loose benefit. The re-design has the higher total cost and higher total benefit. The net values are difficult to compare, as cost was estimated in person months, while benefit was estimated in benefit points. But the benefit-cost-ratio is higher for the re-design and criterion $\Delta TB/\Delta TC$ (right bottom field) also is in favour of the re-design. Therefore, the re-design was chosen.

7 Summary

This work presents in which way the SIKOSA methodology aligns software specification and decisions to quality- and business goals. The following three activities align specification with goals and therefore are presented here: the software

requirements specification, the prioritization of these software requirements, and the definition of decision criteria for architectural design decisions. The results of these three activities influence the fourth activity: making architectural decisions. For aligning software specification and software with goals consistently, it is important to execute these four activities in an integrated way, as it is done by the SIKOSA methodology.

These four activities were executed for a case study where software usability requirements were defined, the usability was assessed, the most important improvements identified, and finally a decision was made between incremental improvement and re-design of the user interface.

Literaturverzeichnis

[AH04] Arora, A.; Hall, D.; Pinto, C.A.; Ramsey, D.; Telang, R.: An ounce of prevention vs. a pound of cure: How can we measure the value of IT security solutions? Lawrence Berkeley National Laboratory. Paper LBNL-54549. 2004.

[Al02] Alexander, I.: Misuse Cases Help to Elict Non-Functional Requirements. http://easyweb.easynet.co.uk/~iany/consultancy/ misuse_cases/misuse_cases.htm

[AWP06] Aurum, A.; Wohlin, C.; Porter, A.: Aligning Software Project Decisions: A Case Study. In: Int. J. of Software Eng. and Knowledge Eng., 16(6), 2006, pp. 795-818.

[BCV05] Bleistein, S.J.; Cox, K.; Verner, J.: Strategic Alignment in Requirements Analysis for Organizational IT: an Integrated Approach. In: ACM Symposium on Applied Computing, Santa Fe, 2005.

[BD04] Bruegge, B.; Dutoit, A.H.: Object-Oriented Software Engineering - Using UML, Patterns, and Java, Prentice Hall, 2004.

[Be00] Beck, K.: Extreme programming explained, Addison-Wesley, Upper Saddle River, 2000.

[BE01] Brücher, H.; Endl, R.: Erweiterung von UML zur geschäftsregelorientierten Prozessmodellierung. In: Proc. Referenzmodellierung RefMod2001, http://www.wi.uni-muenster.de/is/Tagung/Ref2001/Kurzbeitrag13.pdf

[BJ06] Berander, P.; Jönsson, P.: Hierarchical Cumulative Voting (HCV) - Prioritization of Requirements in Hierarchies. In: Int. J. of Software Eng. and Knowledge Eng., 16(6), 2006, pp. 819-849.

[CB95] Clements, P.; Bass, L.; Kazman, R.; Abowd, G.: Predicting Software Quality by Architectural-Level Evaluation. In: Proc. 5th Int. Conf. on Software Quality ICSQ, Maribor, Slovenia, 1995.

[Co01] Cockburn, A.: Writing effective use cases, Addison-Wesley, 2001.

[Da03] Davis, A.M.: The Art of Requirements Triage. In: IEEE Computer 36(3) 2003, pp. 42 - 49.

[DC03] Denne, M.; Cleland-Huang, J.: Software by Numbers: Low-Risk, High-Return Development. Prentice-Hall, 2003.

[DOK05] Dietrich, A.; Otto, S.; Kirn, S.: Simulationsmodell für logistische Prozesse in Mass-Customization-Szenarien. In: Kirn et al. (Hrsg.): Kundenzentrierte Wertschöpfung mit Mass Customization. 2005, pp. 118-147.

[DVF93] Dardenne, A.; van Lamsweerde, A.; Fickas, S.: Goal-Directed Requirements Acquisition. In: Science of Computer Programming 20, 1993, pp. 3-50.

[GY01] Gross, F.; Yu, E.: Evolving system architecture to meet changing business goals: An agent and goal-oriented approach. In: Proc. Fifth IEEE Int. Symposium on Requirements Engineering, 2001, pp.316 - 317.

[He07] Herrmann, A.: Entscheidungen bei der Erfassung nicht-funktionaler Anforderungen. Workshop "Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung", SE 2007, Hamburg, Germany, 2007.

[HKD07] Herrmann, A.; Kerkow, D.; Doerr, J.: Exploring the Characteristics of NFR Methods – a Dialogue about two Approaches. In: Proc. 13th Int. Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality – REFSQ 07, Trondheim, Springer, 2007; pp. 320-334.

[HP05] Herrmann, A.; Paech, B.: Quality Misuse. In: Proc. 11th Int. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 05, Essener Informatik Beiträge, Band 10, 2005; pp. 193-199.

[HP06] Herrmann, A.; Paech, B.: Benefit Estimation of Requirements Based on a Utility Function. In: Proc. 12th Int. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 06, Essener Informatik Beiträge, Band 11, 2006; pp. 249-250.

[HP07] Herrmann, A.; Paech, B.: MOQARE: Misuse-oriented Quality Requirements Engineering. In: Requirements Engineering Journal, to be published.

[HPK06] Herrmann, A.; Paech, B.; Kirn, S.; Kossmann, D.; Müller, G.; Binnig, C.; Gilliot, M.; Illes, T.; Lowis, L.; Weiß, D.: Durchgängige Qualität von Unternehmenssoftware. In: Industrie Management, 6, 2006; pp. 59-61.

[HPP06] Herrmann, A.; Paech, B.; Plaza, D.: ICRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design. In: Int. J. of Software Eng. and Knowledge Eng. 16(6) Dec. 2006, pp. 917-950.

[HRP06] Herrmann, A.; Rückert, J.; Paech, B.: Exploring the Interoperability of Web Services using MOQARE. IS-TSPQ Workshop “First International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems”, Bordeaux, 2006.

[IKO01] In, H.; Kazman, R.; Olson, D.: From Requirements Negotiation to Software Architectural Decisions. In: Proc. From Software Requ. to Architectures Workshop STRAW, 2001.

[ISO02] International Standards Organization ISO: Risk management – Vocabulary – Guidelines for use in standards. ISO Guide 73, International Standards Organization, 2002.

[ISO13] International Standards Organization ISO: Norm DIN EN ISO 13407, Benutzer-orientierte Gestaltung interaktiver Systeme.

[ISO92] International Standards Organization ISO: Norm DIN EN ISO 9241, Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten.

[KAK01] Kazman, R.; Asundi, J.; Klein, M.: Quantifying the Cost and Benefits of Architectural Decisions. In: Proc. Int. Conf. Software Eng., 2001; pp.297-306.

[KAB96] Kazman, R.; Abowd, G.; Bass, L.; Clements, P.: Scenario-based analysis of software architecture. In: IEEE Software, 13(6), 1996; pp. 47-55.

[Ka96] Karlsson, J.: Software requirements prioritisation. In: Proc. 2nd Int. Conf. Requirements Engineering, 1996; pp.110-116.

[KN92] Kaplan, R.S.; Norton, D.P.: The Balanced Scorecard: Measures That Drive Performance. In: Harvard Business Review, 70(1), 1992; pp. 71-79.

[KL06a] Korherr, B.; List, B.: Aligning Business Processes and Software Connecting the UML 2 Profile for Event Driven Process Chains with Use Cases and Components. In: Proc. 18th Int. Conf. on Advanced Inf. Systems Eng. CAiSE´06, Luxembourg, 2006: pp. 19-22.

[KL06b] Korherr, B.; List, B.: A UML 2 Profile for Event Driven Process Chains. A UML 2 Profile for Business Process Modelling. In: Proc. 1st Int. Workshop on Best Practices of UML at the 24th Int. Conf. on Conceptual Modeling ER 2005, Klagenfurt 2005.

[KWR98] Karlsson, J.; Wohlin, C.; Regnell, B.: An evaluation of methods for prioritizing software requirements. In: Information and Software Technology 39, 1998; pp. 939-947.

[La01] van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proc. of 5th Int. Symposium on Requirements Eng., 2001; pp. 249-263.

[La02] Lauesen, S.: Software Requirements - Styles and Techniques. Addison-Wesley, 2002.

[LRV99] Lassing, N.; Rijsenbrij, D.; van Vliet, H.: On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn´t Everything. In: Proc. Second Nordic Software Architecture Workshop NOSA 99, 1999; pp. 1103-1581.

[Lu00] Luftman, J.N.: Assessing business/IT alignment maturity. In: Comm. of AIS, 4(14), 2000.

[LW00] Leffingwell, D.; Widrig, D.: Managing Software Requirements - A Unified Approach, Addison-Wesley, Reading, Massachusetts, USA, 2000.

[No04] Noran, O.S.: Business Modelling: UML vs. IDEF. Griffith University, School of Computing and Information Technology, 2004. http://www.cit.gu.edu.au/~noran/Docs/UMLvsIDEF.pdf (last visit: 12 nov 2007)

[PK03] Paech, B.; Kohler, K.: Task-driven Requirements in object-oriented Development. In (Leite, J.; Doorn, J., eds.): Perspectives on Requirements Engineering, Kluwer Academic Publishers, 2003.

[PSR04] Papadacci, E.; Salinesi, C.; Rolland, C.: Payoff Analysis in Goal-Oriented Requirements Engineering. In: Proc. 10th Int. Workshop on Requirements Eng.: Foundation of Software Quality – REFSQ04, 2004.

[REP03] Ruhe, G.; Eberlein, A.; Pfahl, D.: Trade-Off Analysis For Requirements Selection. In: Int. J. of Software Eng. and Knowledge Eng. 13 (4), 2003; pp. 345-366.

[RHN01] Regnell, B.; Höst, M.; Natt och Dag, J.; Beremark, P.; Hjelm, T.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. In: Requirements Eng. 6, 2001; pp. 51–62.

[RS05] Rolland, C.; Salinesi, C.: Modeling Goals and Reasoning with Them. In (Aurum, A.; Wohlin, C., Eds.): Engineering and Managing Software Requirements, Springer, Berlin, Heidelberg, 2005.

[Sa80] Saaty, T.L.: The Analytic Hierarchy Process, McGraw-Hill, New York, 1980.

[Sc01] Scheer, A.-W.: ARIS – Modellierungsmethoden, Metamodelle, Anwendungen. 4th edition, Springer, Berlin, 2001.

[SH00] Scheer, A.; Habermann, F.: Making ERP a success. In: Communications of the ACM, 43(3), 2000; pp. 57-61.

[SKK97] Stylianou, A.C.; Kumar, R.L.; Khouja, M.J.: A total quality management-based systems development process. In: ACM SIGMIS Database, 28(3), June 1997, pp. 59-71.

[SO00] Sindre, G.; Opdahl, A.L.: Eliciting Security Requirements by Misuse Cases. In: TOOLS Pacific, 2000; pp. 120-131.

[SO01] Sindre, G.; Opdahl, A.L.: Templates for Misuse Case Description. In: Proc. 7th Int. Workshop on Requirements Eng.: Foundation of Software Quality – REFSQ 01, Essener Informatik Beiträge Band 6, 2001; pp. 125-136.

[So01] Sommerville, I.: Software Engineering, Pearson Education Deutschland, 6th edition 2001.

[Sy07] Sysiphus http://sysiphus.in.tum.de/, 2007 (last visit: 12 nov 2007)

[Wi99] Wiegers, K.E.: First things first: prioritizing requirements. In: Software Development 7(9), September 1999.

[Wi02] Wiegers, K.E.: Success Criteria Breed Success. In: The Rational Edge, 2(2), 2002.

[WKK07] Weiß, D.; Kaack, J.; Kirn, S.; Gilliot, M.; Lowis, L.; Müller, G.; Herrmann, A.; Binnig, C.; Illes, T.; Paech, B.; Kossmann, D.: Die SIKOSA-Methodik – Unterstützung der industriellen Softwareproduktion durch methodisch integrierte Softwareentwicklungsprozesse. In: Wirtschaftsinformatik 49(3), 2007; pp. 188-198.

[XMC04] Xie, N.; Mead, N.R.; Chen, P.; Dean, M.; Lopez, L.; Ojoko-Adams, D.; Osman, H.: SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies. Technical Note CMU/SEI-2004-TN-045, Software Engineering Institute, Carnegie Mellon University, 2004.

[YM98] Yu, E.; Mylopoulos, J.: Why Goal-Oriented Requirements Engineering? In: Proc. 4th Int. Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality – REFSQ 1998, Pisa, Presses Universitaires de Namur, 1998, pp. 15-22