2014 IEEE 7th International Conference on Service-Oriented Computing and Applications

The Weighted Decision Matrix: Tracking Design Decisions in Service Compositions

Alexandra Mazak
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
Email: mazak@big.tuwien.ac.at

Bernhard Kratzwald
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
Email: kratzwald@big.tuwien.ac.at

***Abstract*—Often, there is a lack of efficient procedures in order to identify design errors when transforming customer requirements into a servic-oriented solution. A well-established approach supporting this kind of transformation in classical requirements engineering is the traceability matrix which allows tracing the coverage of customer requirements by system components in general, or services in particular. The matrix shows which services realize which customer requirements. However, the matrix does not show the relevance of a service in order to meet a certain quality aspect (e.g. security, usability, maintainability, etc.). Since this information is not explicitly covered, it is the system designer who has to keep this knowledge in mind when designing an appropriate set of services that build the overall system. Unfortunately, stakeholders not involved in this task, e.g. customers, project managers and developers, have a hard time to understand the significance of the coverage of customer requirements by services in order to meet desired quality aspects. This may cause misinterpretations especially in the early stages of the service system development life-cycle. In this paper, we present a heuristic-based approach to continually monitor and control design decisions and their effects on certain quality aspects. This approach extends the traceability matrix by a weighted decision-matrix.**

I. Introduction

When developing a service-oriented application, in a first step the customer requirements are recorded by a requirements analyst as in any other software development approach. In a subsequent step, a system designer takes these requirements and designs a system built by a number of appropriate services no matter whether existing services are reused or new services have to be implemented. Usually, the transformation from customer requirements to the specific set of services is based on the expert knowledge of the system designers. Without a proper documentation, it remains unclear why a certain service is part of the system and other alternatives are not. Thus, quality control in this early phase of the system design becomes more and more important. However, stakeholders rarely accept time-consuming quality-analyzes already starting in the design phase (e.g. when customer requirements are transfered in a set of appropriate services). Thus, a lightweighted approach as described in this paper is needed.

Especially in large projects it is difficult to spend as much effort on internal software quality as on keeping time-lines and budget [9]. Experience reports and interviews with practitioners [18], [19], [5] point to the lack of efficient procedures to identify design errors as early as possible. In most cases, problems result from the following issues:

- Architects and system designers need to know the design model very well in order to keep track of their design decisions made during the transformation of customer requirements to a set of appropriate services.
- In some cases project staff prematurely leaves a project and new project members mostly miss efficient documentation, e.g., to be able to gain an overview of the design decisions made.
- Often, requirements analysts model a design solution already in the analysis phase. Such hasty design decisions may cause solution-specific restrictions which are not appropriate.
- Stakeholders not involved in the design decisions, e.g. customers, project managers and developers, have a hard time to understand the significance of realization relationships between customer requirements and services. This often leads to misinterpretations among them.
- Ad-hoc changes to service implementations often neglect existing dependencies to other services.

In order to overcome these issues tracing requirements during the whole service system development life-cycle becomes more and more important [14]. For instance, traceability is a key concept in quality control of high-security systems, such as in medicine and the automotive industry. Traceability guarantees that all requirements are sufficiently considered in the service design in order to be correctly implemented. In this respect, traceability acts as an indicator of the system's maturity. At any point in the development process traceability may be used to justify the existence of a service. Furthermore, traceability ensures completeness that all customer requirements are considered by an appropriate set of services. Thereby, it helps to develop systems efficiently and less error-prone.

In addition to traceability, measurement-based quality management techniques known from classical software engineering can be used to control the internal quality in service-oriented solutions. In general, such techniques provide key figures to draw conclusions about the quality of the software.

Methods used in this context are *design* and *code reviews* as well as *inspections*. Such "ex post" methods should identify requirements that are not addressed by the services or extra services items that are not required. Großmann argues that these methods are indeed common in practice, but are usually applied in an unsystematic way or discarded in pressure situations [9].

However, an early detection of design pitfalls avoids architectural "dead-ends" and "design patchworks" which do not meet the expectations of stakeholders. Often, customers are not willing to pay extra costs for revisions since the extra effort is due to the faults of the solution provider. Moreover, services which have been designed incorrectly or incompletely are implemented as is by developers, since they rely on the previous work of architects and system designers. However, design errors have a negative effect on development costs and time and can lead to the project's termination or a redevelopment of the service-oriented system [16].

All these common pitfalls motivated us to work on a heuristic-based controlling-approach in order to provide appropriate quality indicators. These indicators enable an ongoing assessment of the design quality for all stakeholders involved in the project (e.g. project managers, architects, system designers, developers, customers). For this reason, we introduce a procedure in order to record the system designers' perception of internal software quality during the service design. Through this procedure the "design-knowledge" is explicitly documented, and thus, verifiable and traceable. We implement a relevance-weighting method by which the fulfillment of certain quality aspects (e.g., usability, security, maintainability) can be assessed. For this purpose we extend the relationship or traceability matrix (Section II-A) into a *Weighted Design Decision Matrix (WDDM)*.

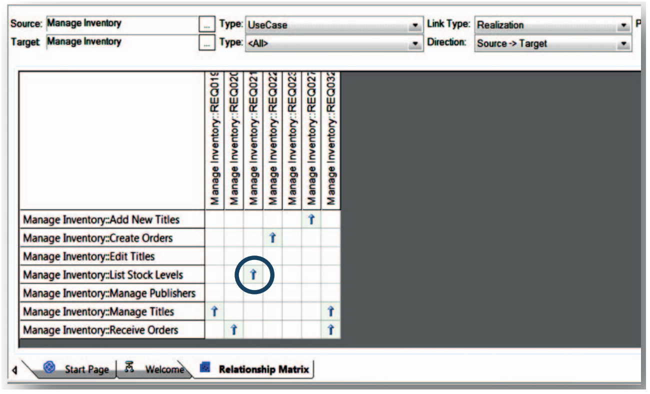
Unlike conventional methods as reviews or inspections, the WDDM helps to prevent design errors right from the beginning. By applying the WDDM, project members can continuously monitor and control service design decisions and possible alternatives at any time during the system development life-cycle. For instance, system designers can track how the quality of design is affected when stakeholders change their prioritization or delete and add new requirements. Our aim is to align the service design on the needs of stakeholders and according to standard-based quality aspects as defined in [4]. The higher the quality of the service design is the lower the difference between customer needs and the actual suitability of the service design.

The remainder of this paper is organized as follows: Section 2 presents background knowledge and related work. In Section 3 we describe the theoretical basis, the relevance-weighting procedure and the Weighted Design Decision Matrix. Both are already realized as add-in in the tool *Enterprise Architect* of *SparxSystems* [18]. Finally in Section 4, we present our conclusions and future work.

II. Related Work

A. Requirements Traceability Matrix

In 1986, McMillan and Vosburgh introduced the *traceability matrix*. This matrix is a useful and well-established



Source: Manage Inventory
Type: UseCase
Link Type: Realization
Target Manage Inventory
Type: <All>
Direction: Source -> Target

	Manage Inventory::REQ019	Manage Inventory::REQ020	Manage Inventory::REQ021	Manage Inventory::REQ022	Manage Inventory::REQ023	Manage Inventory::REQ027	Manage Inventory::REQ032
Manage Inventory::Add New Titles						↑	
Manage Inventory::Create Orders				↑			
Manage Inventory::Edit Titles							
Manage Inventory::List Stock Levels			↑				
Manage Inventory::Manage Publishers							
Manage Inventory::Manage Titles	↑						↑
Manage Inventory::Receive Orders		↑					↑

Fig. 1. Enterprise Architect Relationship Matrix

aid to relate customer requirements to design items (such as services in our case). The authors propagate that generally "validation pain" (e.g. of the architecture design) can be reduced through the traceability matrix [13]. The traceability matrix traces relationships among requirements at different points. The matrix displays the complexity of dependencies to users. Consequently, requirements traceability must have a backwards and forwards direction to follow the life of a requirement.

Gotel and Finkelstein analyze in detail the multifaceted nature of the "traceability problem" based on an empirical study [8]. They differentiate between *pre-requirements specification traceability* and *post-requirements specification traceability*. The pre-requirements specification traceability refers to those aspects of a requirement's life cycle prior to inclusion in the specific task (e.g., when stakeholders prioritize requirements); whereas the latter refers to those aspects resulting from inclusion in the requirements specification [8].

In the tool *Enterprise Architect (EA)* [18] the traceability matrix is implemented as a relationship matrix (cf. Figure 1). This *relationship matrix* consists of one row for each requirement and one column for each design item in general or service in specific. In the cells an arrow symbolizes that a customer requirement is covered by a service. A cell is marked by an arrow based on an existing `realization` relationship in UML [6] or a `satisfy` relationship in SysML [7].

The matrix provides a traceability view on the service design for stakeholders. Figure 1 shows a snapshot of an example matrix. The matrix has the form of a spreadsheet displaying the relationships between requirements and design items in general or services in specific. On the top there is a filter to select a source package of customer requirements and a target package of services. In addition, users can select the direction for displaying the source and target packages and the relationship type (e.g. a realization). Based on the filter selection the corresponding grid square is generated and arrows indicate the requirements traceability.

The traceability matrix is a convenient method for quickly visualizing realization relationships. The matrix guides users when creating, modifying, or deleting relations among elements with a minimum effort by a simple mouse-click. It is an aid for tracing which customer requirements are to be implemented by which service. Thereby, system designers can ensure that all requirements are taken into account. However, the matrix does not show the relevance of the service in order to meet certain quality aspects (e.g. security, usability, maintainability). It is the system designer, who has to keep this knowledge in mind during the service design. This means that this design knowledge is not explicitly available or transparent to other stakeholders not involved in design decisions.

B. Requirements Prioritization

In the analysis phase, stakeholders prioritize requirements (e.g. must-be, expected, nice-to-have) depending on their expectations, e.g., based on a certain business goal. Various methods (e.g. Quality Function Deployment [1]) and supporting tools exist for guiding the process of customer requirements prioritization. Mostly, these methods use ranking or classification techniques for prioritization. For instance, stakeholders use the top-ten technique [14] to select a number of requirements by a certain criterion (e.g. costs of implementation) and prioritize them according to their needs. In the analysis phase, stakeholders have the real usage of the final product in mind. Thereby, quality aspects (e.g. usability, security, maintainability, etc.) play an important role in the prioritization process.

We argue that after this phase, the view on the quality of the final product is often neglected. Especially, during the design phase—i.e., when system designers cover the prioritized requirements by an appropriate set of services—a "quality-view" on the service design would help to overcome misinterpretations among stakeholders. Such misinterpretations are mainly caused by a gap in the quality-based expectations between customers and architects / system designers.

Usually, a certain customer requirement is realized by multiple services. We point to the fact that the importance of those realizations differ from quality aspect (e.g. security) to quality aspect (e.g. usability). Therefore, we suggest that in the same way like stakeholders prioritize their requirements based on certain business goals in the analysis phase, also, system designers should assess their decisions made in the design phase regarding to the expected quality of the service design.

C. Review, Walkthrough and Inspection

Manually performed inspection techniques enable the verification of requirements. These techniques are known under the generic term *reviews* [14]. Pohl and Rupp describe three forms of reviews [15]: (i) report, (ii) inspection, and (iii) walkthrough.

By applying the test technique *report*, the architect or system designer hands the requirements over to a third person (e.g. colleague, expert). The aim is to obtain an expertise with regard to the quality of development artifacts, such as the service design. This procedure is applied ex post. This means that the report starts when the customer requirements are already transformed into services. The expert checks the requirements and identifies quality defects based on predefined quality criteria.

The *walkthrough* is a light-weighted version of the review. The walkthrough runs less stringent than the inspection and the roles involved are less differentiated. Generally, there are three assigned roles: author, reviewer, and supervisor. Together, they identify possible quality defects in the development artifact. The goal of this method is to produce a consolidated understanding among the involved roles.

Inspections have the goal to systematically search development artifacts for design errors based on a strict process scheme [12]. Usually, an inspection is applied for collecting and evaluating the findings. In this procedure, different roles (e.g. organizer, moderator, author, reader, supervisors, and reporter) are assigned for certain tasks.

In addition, there exists the *prototype method*. This method enables to check the conversion of requirements by a prototype. The prototype method is also an ex post method for detecting design errors.

Last but not least, agile approaches (e.g. Scrum [10]) have been introduced in recent years in order to avoid expensive trouble shooting in the final phases of a project. Agile approaches recommend an early stakeholder feedback, often based on early prototyping. However, working with agile methods does not mean developing without a design model. Steinpichler and Kargl [18] report—based on their industrial experience—that a distributed traceability of requirements by external tools or methods that are not integrated in the design process itself, is often error-prone. Furthermore, they argue that such tools produce less compelling documentation.

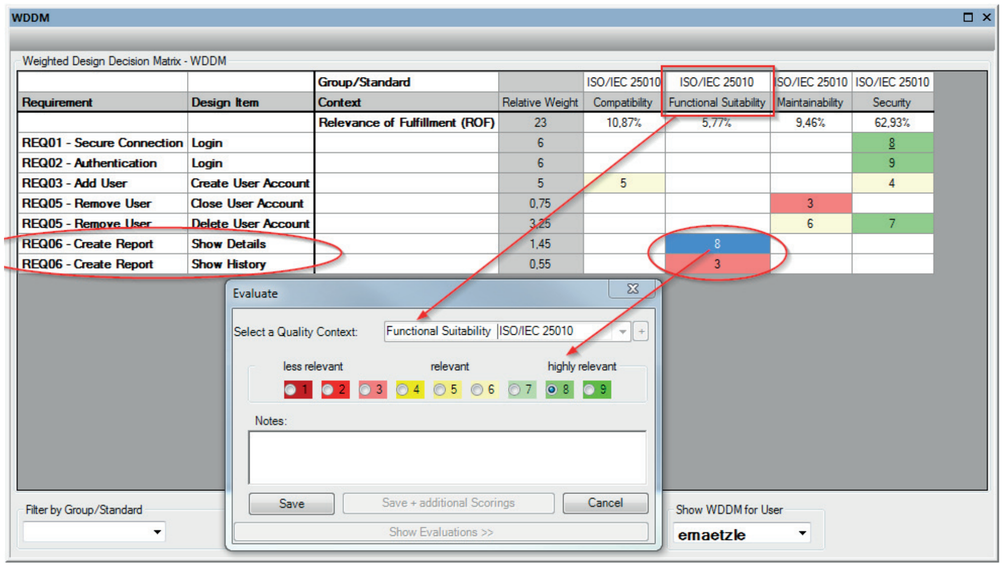
III. Theoretical Approach

A. TEMoR-cycle

Architects or system designers decide about how to transform customer requirements into a set of appropriate services. There are several alternatives to specify these realization relationships. Typically, there does not exist a unique optimal solution for a service design. We propose to document the arguments that lead to design decisions by the system designers. By explicitly capturing these decisions, we aim at utilizing the cognitive abilities of system designers.

The monitor and control aspects in our approach are based on the *Plan-Do-Check-Act (PDCA) cycle* [11]. The PDCA-cycle is a popular iterative four-step management method frequently used in the domain of controlling. The cycle constitutes a continuous improvement of processes. Several activities are involved in each step. In short, users identify and analyze a problem in the *Plan-step*. In the *Do-step*, users develop and test potential solutions. In the *Check-step*, they measure the effectiveness of each test solution and analyze whether the solution could be improved or not. In the *Act-step*, users implement the improved solution.

Based on this well-established controlling method, we define the *TEMoR-cycle* for our approach. The core functions are transact, evaluate, monitor and regulate. Following the idea of the the PDCA-cycle, the TEMoR-cycle represents a continuous improvement process. This demand corresponds to the basic principles of quality management defined in the ISO 9001 standard [3]. In the first step called *transact*, system designers make design decisions (i.e., by the time they create realization relationships). In the *evaluate-step*, they evaluate these design decisions by considering qualitative aspects (e.g.



		Group/Standard		ISO/IEC 25010	ISO/IEC 25010	ISO/IEC 25010	ISO/IEC 25010
Requirement	Design Item	Context	Relative Weight	Compatibility	Functional Suitability	Maintainability	Security
		Relevance of Fulfillment (ROF)	23	10,87%	5,77%	9,46%	62,93%
REQ01 - Secure Connection	Login		6				8
REQ02 - Authentication	Login		6				9
REQ03 - Add User	Create User Account		5	5			4
REQ05 - Remove User	Close User Account		0,75			3	
REQ05 - Remove User	Delete User Account		3,25			6	7
REQ06 - Create Report	Show Details		1,45		8		
REQ06 - Create Report	Show History		0,55		3		

Fig. 2. GUI of the relevance-weighting procedure of realization relationships implemented in the tool *Enterprise Architect*.

security, usability, maintainability). This step is tool-guided, since the system designer is automatically prompted by the system to assess his design decisions by relevance-weightings. The *monitor-step* enables system designers and other stakeholders (depending on the granted rights) to review the service composition according to certain quality aspects at any time in the service design phase. This step helps to identify potential quality risks, such as important quality criteria that were disregarded or not designed as expected by stakeholders. In the monitor-step, the system designer continuously reconsiders his design decisions by focusing on certain quality aspects. Recognized issues are addressed in the *regulation-step* in order to avert quality risks. We have prototypically implemented the TEMoR-cycle as a controlling instrument as *Add-in* in the *Enterprise Architect* tool.

B. Relevance-Weighting Procedure

In our daily lives, we usually weight multiple criteria implicitly and are then comfortable with the consequences of such decisions based on our intuition. This can be seen as a cognitive process. During the relevance-weighting procedure, system designers assess each realization relationship between customer requirements and services with a "relevance value" which refers to a purely subjective (intuitive) value. These relevance-weightings are based on the system designer's perception when making design decisions.

Relevance is a decision-theoretical construct in the pragmatic-based *relevance theory* introduced by Sperber and Wilson [17]. Relevance describes the rating of the importance of objects and information for a person in certain contexts. The works of Sperber and Wilson [17], [20] mainly rely on the *speech act theory* which describes the relation between a communicator and his uttered proposition to an auditor. The authors define that *"an assumption is relevant in a context if and only if it has some contextual effect"*, [17]. The aim of their theory is to assess the *contextual effects* of explicitly expressed assumptions. In their approach relevance is a comparative, quantitative, and context-dependent concept which makes relevance assessable and measureable [20].

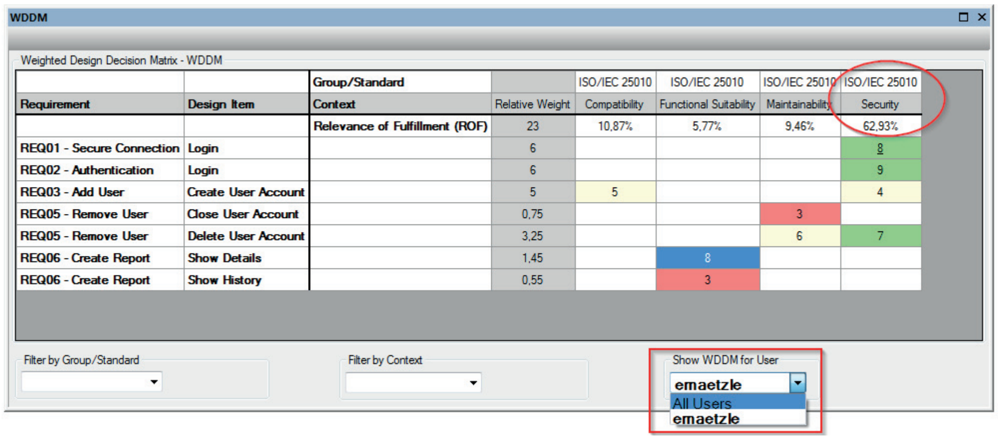
In our approach, we apply parts of the pragmatic-based relevance theory to the evaluation step in the TEMoR-cycle which the relevance-weighting procedure starts with. In this procedure, we use the term "quality context" synonymously to the introduced term "quality aspect", since relevance is context-dependent. In the current version, we implement such contexts regarding to the ISO/IEC 25010 quality standard [4]. However, it is also possible to create user- or application-specific contexts (e.g. certain safety conditions).

Realization relationships visualize the coverage of customer requirements by design iterms in general or services in specific (cf. Figure 1). The relevance-weightings in our approach visualize the consideration of certain quality aspects (e.g. security, usability, functionality) when creating those relationships. The service designers perform these relevance-weightings in the evaluate-step of the TEMoR-cycle (cf. Section III-A).

The weightings are stored as meta-information in the service design. The meta-information itself is a data structure consisting of ID, source, quality context (e.g. standard ISO/IEC 25010), context criterion (e.g. usability, functional suitability, maintainability, etc.), object of evaluation (the realization relationship), relevance-weighting and a description field (optional).

The relevance of realizing a certain customer requirement by a certain service may differ from quality context to quality context. We consider this fact by providing a 9-point rating scale to the system designer. Once a realization relationship has been created, the relevance-weighting procedure is automatically initiated. The system prompts the system designer to select a predefined quality context (e.g. ISO/IEC 25010), and a context criterion (e.g. usability), or to create a new one and to assign a relevance value from the rating scale.

The pop-up window in Figure 2 presents the three classes



		Group/Standard		ISO/IEC 25010	ISO/IEC 25010	ISO/IEC 25010	ISO/IEC 25010
Requirement	Design Item	Context	Relative Weight	Compatibility	Functional Suitability	Maintainability	Security
		Relevance of Fulfillment (ROF)	23	10,87%	5,77%	9,46%	62,93%
REQ01 - Secure Connection	Login		6				8
REQ02 - Authentication	Login		6				9
REQ03 - Add User	Create User Account		5	5			4
REQ05 - Remove User	Close User Account		0,75			3	
REQ05 - Remove User	Delete User Account		3,25			6	7
REQ06 - Create Report	Show Details		1,45		8		
REQ06 - Create Report	Show History		0,55		3		

Fig. 3. GUI of the Weighted Design Decision Matrix implemented as add-in in the tool *Enterprise Architect*.

for the relevance rating in form of a scoring, (i) "less relevant" in the range of [1, 3] highlighted red, (ii) "relevant" in the range of [4, 6] highlighted yellow, and (iii) "highly relevant" in the range of [7, 9] highlighted green.

Definition 1. $(1, 9)$ is a scale of relevance (1 = lowest relevant, 9 = highest relevant), then assessing the realizations `X` with `i` and `Y` with `j` implies that `Y` has more contextual effect than `X`; $\forall i, j \in \{1, \ldots, i, j, \ldots 9\}$, $j > i$.

Figure 2 shows that the customer requirement *REQ06-Create Report* is modeled by two services: *Show Details* and *Show History* (cf. red circle on the left in the snapshot). The system designer decides that the *Show Details* service has a greater contextual effect (relevance-weighting = 8, cf. red circle on the right in the snapshot) regarding the quality criterion "Functional Suitability" than *Show History* (relevance-weighting = 3, cf. red circle on the right in the snapshot).

The semantics of the relevance-weighting procedure means: the more relevant a realization of a customer requirement by a certain service is with respect to a given quality context the greater is the effet of this realization in terms of the qulity of the service design.

C. Weighted Design Decision Matrix

As a result of the relevance-weighting procedure system designers receive a *Weighted Design Decision Matrix (WDDM)* which reflects their design choices. The WDDM is a concise representation on how the objects of evaluation (i.e. realization relationships between requirements and services) are weighted in a selected quality context.

Figure 3 shows the object of evaluation in each row of the matrix, which is a key value pair consisting of a customer requirement related to a service. The relative weight (cf. Figure 3, fourth column highlighted gray) is a ratio value calculated from the predetermined stakeholder value of the customer requirement (cf. Section II-B) and the previously conducted relevance-weightings. By this relative weight, we merge the integration view of system designers in the design phase—expanded by a "relevance-view"—with the perspective of stakeholders in the analysis phase.

If multiple system designers complete the WDDM, a cognitive map of different intuitive quality-views can be provided to stakeholders. So, stakeholders can see and understand the consequences of different service design alternatives and different understandings of quality aspects. This helps in creating a common understanding among customers and project managers on the one side and architects and system designers on the other side.

The *relevance of fulfillment (RoF)* calculated in percentage is an indicator for estimating which quality aspects (e.g. compatibility, functional suitability, maintainability, security) the system designers have preferred (e.g. the quality aspect security with the value 62,93% shown in Figure 3).

The manually assigned relevance-weightings as well as the results of the subsequent computations are continuously updated when a relevance-weighting of a realization relationship has been performed.

Similar to the traceability matrix, the Weighted Design Decision Matrix is a list of information and values grouped in rows and columns. The WDDM enables users to identify, evaluate, monitor and control the relationships between requirements, services, and various quality aspects. The WDDM helps all stakeholders to validate if the service design is presented in a satisfactory quality corresponding to the customer requirements prioritization and to standard-based quality aspects (ISO/IEC 25010—system and software quality models [4]), or to self-defined contexts.

IV. Conclusion and Future Work

Our approach starts from the customer requirements prioritization recorded in the analysis phase (cf. Section II-B). We use the traceability matrix (cf. Figure 1) for identifying realization relationships between customer requirements and services. We extend the traceability matrix into a Weighted Design Decision Matrix (cf. Figure 3). The WDDM facilitates the comparison among service design results of different system designers.

We make the individual view of each system designer on quality aspects transparent and traceable to other stakeholders. For this purpose, we introduce the relevance-weighting procedure (cf. Section III-B) where system designers weight the relevance of realization relationships between customer requirements and services. In this manually conducted procedure, system designers quantify the quality-based contextual effect a realization has in their expert opinion (cf. Figure 2). We implement these relevance-weightings as tagged values. These values are summarized and evaluated as indicators (Relevance of Fulfillment, cf. Section III-C). These indicators enable a continuous quality monitoring and controlling already during the service design phase. The WDDM enables stakeholders to reflect on possible quality risks at any time. The introduced approach can be classified as post-requirements specification traceability method according to [8].

The WDDM can be expanded to describe a *multi-criteria decision analysis (MCDA)* problem. The MCDA-problem describes M alternative options each need to be assessed on N criteria. For this purpose, we work on a second controlling instrument that should help to identify the critical, highly relevant services in the architecture—the so called "architectural drivers". For instance, this instrument may be used to estimate the effort of developers in the implementation of services.

The WDDM in combination with the new instrument should guide project members by identifying which customer requirements were transferred not at all, not sufficiently, or even too detailed in the service design. This should guarantee—according to the "*Grundsätze ordnungsmäßiger Modellierung*" (in English: Principles of Good Modeling) [2]—the adequacy of the service design. Thereby, stakeholders can identify services which result in a low benefit and are, therefore, not adequate for implementation. As a result, a possible over-engineering risk can be minimized even before the implementation task starts. Moreover, developers can identify significant services in order to give them priority, e.g., as a starting point for agile methods.

In addition, we have started an evaluation survey. Currently, trainers of SparxSystems conduct this survey during on-the-job-trainings at their customers (approximately 80-100 respondents). Although the survey is in a preliminary stage, the first results are rather promising and underpin the usefulness of the WDDM.

Acknowledgment

This work was supported as part of the BRIDGE program of the Austrian Research Promotion Agency (FFG) under grant number 836529—a joint researcher effort of Research Studios Austria (in cooperation with the Vienna University of Technology) and SparxSystems Software GmbH.

References

[1] Y. Akao. *Quality Function Deployment QFD: Integrating Customer Requirements into Product Design*. Productivity Press, 2004.

[2] J. Becker, M. Rosemann, and R. Schütte. Grundsätze ordnungsmäßiger Modellierung GoM. *Wirtschaftsinformatik*, 37(5):435–445, 1995.

[3] J.-P. Brauer. *DIN EN ISO 9000:2000 ff. umsetzen: Gestaltungshilfen zum Aufbau Ihres Qualitätsmanagementsystems*. Hanser Verlag München, 2009.

[4] B. BSI. ISO/IEC 25010: 2011 Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models, 2011.

[5] CompTIA. State of the IT Skills Gap. CompTIA Properties LLC, 2012. http://www.comptia.org.

[6] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3th edition, 2003.

[7] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Elsevier, 2nd edition, 2012.

[8] O. C. Z. Gotel and A. C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101, Apr 1994.

[9] M. Großmann. Messbasierte Qualitätssicherungstechniken für die Steuerung von innerer Software-Qualität in der Praxis. In *Workshop Software-Reengineering und Design for Future*, Bad Honnef, 2010.

[10] H. Kniberg. Scrum and XP from the Trenches. *Lulu. com*, 2007.

[11] S. Kostka and C. Kostka. *Der Kontinuierliche Verbesserungsprozess*. Carl Hanser Verlag München, 6 edition, 2013.

[12] O. Laitenberger. A Survey of Software Inspection Technologies. In *Handbook of Software Engineering and Knowledge Engineering*, pages 517–555. World Scientific Publishing, 2002.

[13] J. MacMillan and J. R. Vosburgh. Software Quality Indicators. Technical Report ADA181505, DTIC Document, 1986.

[14] K. Pohl. *Requirements Engineering: Grundlagen, Prinzipien Techniken*. dpunkt.verlag, 2007.

[15] K. Pohl and C. Rupp. *Basiswissen Requirements Engineering*. dpunkt.verlag, 3 edition, 2011.

[16] A. Schatten, S. Biffl, M. Demolsky, E. Gostischa-Franta, T. Östreicher, and D. Winkler. *Best Practice Software-Engineering*. Springer DE, 2010.

[17] D. Sperber and D. Wilson. *Relevance: communication and cognition*. Blackwell Verlag, second edition, 1995.

[18] D. Steinpichler and H. Kargl. *Enterprise Architect: Project Management with UML and EA*. SparxSystems Software GmbH, 9.3 edition, January 2011.

[19] S. Wagner, M. Broy, F. Deißenböck, M. Kläs, P. Liggesmeyer, J. Münch, and J. Streit. Eine forschungsagenda für softwarequalität. *TUM*, page 47, 2008.

[20] D. Wilson and D. Sperber. *Meaning and Relevance*. Cambridge University Press, 2012.