Managing Requirement Engineering Processes by Adapted Quality Gateways and critique-based RE-Tools

Eric Knauss and Thomas Flohr

Software Engineering Group, Leibniz Universität Hannover,
Welfengarten 1, 30167 Hannover, Germany
{eric.knauss, thomas.flohr}@inf.uni-hannover.de

Abstract. Choosing the right quality level for requirements is still a challenging but crucial task. While spending too little effort can result in a failed project, spending too much effort on requirements threatens the project schedule and the budget. Consequently, formal criteria are needed in order to determine from an objective point of view whether the quality of requirements is sufficient for a given project situation. We propose the use of an adapted Quality Gateway to obtain a comparable, repeatable, and objective check. Furthermore, the Quality Gateway concept can be strongly improved if it is combined with a Domain Oriented Design Environment (DODE) to construct adequate requirements documentation in the most critical areas beforehand. This way, requirements can be observed and fine-tuned for following activities within the software development process. This paper presents the DODE concept as a supplemental to adapted Quality Gateways.

1 Introduction

Requirements engineering should be tailored to a given project situation like other engineering tasks in the software development domain, too. Only this way it can be ensured that requirements engineering fits a project situation. On the one hand the right level of requirements engineering should avoid specifying unnecessary details and thereby wasting money. On the other hand requirements should have enough quality to be useful in later activities in a software development process as well as the project managing activities (e.g. cost estimation, progress monitoring).

In practice (university teaching and reports of local software companies), we have often observed the following situation: at some point in the project the requirements have to be promoted to the next activity in the developed process in order to form the foundation for design or implementation. We have regularly noticed that this point is defined by project schedules instead of quality aspects. Consequently, the requirements as the project's foundation are often far from perfect. In this dilemma we want to optimize requirements according to the following constraints:

– Whenever the time available for requirements engineering is limited, it shall be used as efficient as possible.

– Even if the requirements cannot be documented perfectly, we still want to avoid the most severe errors.
– Even if a complete requirements specification cannot be achieved, we still want to specify the most important areas.

Intuition can help to judge the adequacy of requirements for a given project situation. Nonetheless, decisions based on intuition require experience. Furthermore, such decisions are neither reliable nor repeatable because they depend on the project team's skills, on the availability of customers, on the criticality of the software system under construction, and in practice often on the tight project schedule.

This paper presents an approach for constructing and checking suitable requirements by combining the (constructive) DODE [1] and the (analytical) Quality Gateway [2] concepts in order to incorporate the strengths of both. Furthermore, we explain how experience management helps to fine tune both concepts and share results and experiences gained during evaluation of our approach.

1.1 Outline

This paper is structured as follows: section two explains the DODE and Quality Gateway concept. Furthermore, this section shows how both concepts have to be combined and how they can be mapped to a development process. Section three presents the integration of feedback cycles as a foundation for experience based improvements. Section four introduces a DODE for constructing use cases and NetQGate as a tool for supporting Quality Gateways. This is a possible setup of the concept that demonstrates the benefits of a common experience infrastructure. Our experiences and case studies are provided here to illustrate our concept. Within this section the metrics we have successfully used in our projects settings are shown. Finally, section five contains our conclusion and experiences we made with the concept.

1.2 Related Work

Modern software development process models suggest tailoring the demanded document types for the specific project context. However, this task is often neglected as shown in [3]. Project members are overstrained because they do not know the impact of tailoring decisions. Such knowledge has to be derived from experience which is gained through experimental studies or anecdotes. Knowledge-based concepts like the Experience Factory [4] promise to support this hard learning process. This is achieved by introducing a common knowledge infrastructure for the projects of an organization, thus creating a learning organization. Experiences gained in one project can directly affect other projects that face similar challenges. We will use such mechanisms in our approach to tune the quality assurance based on similar projects.

In our approach, the quality assurance is based on formal criteria to judge requirements from an objective perspective [5]. For example, a formal criterion can

be a measured value complying with a certain threshold. In general, we define that a formal criterion can be determined by a software tool or a human role in an objective and repeatable way.

Robertson et al. [6] utilized formal criteria in their Quality Gateway concept to some extent. A Quality Gateway is a formal checkpoint at which formalized requirements are checked. Only requirements meeting the completeness, traceability, testing fit, and other criteria are allowed to pass the Quality Gateway. Then these requirements can be incorporated into the specification. Consequently, the Quality Gateway is a mechanism for judging requirements formally after their creation. Our approach uses a slightly changed and adapted version of Robertson's Quality Gateway in which all kinds of formal criteria can be checked.

But at the time of the Quality Gateway, effort was already spent on the creation and rework might be necessary. If we had formal checks on requirements much earlier in the process – at best during creation of a requirement – we could constructively enhance the quality and reduce expensive rework.

The DODE (Domain Oriented Design Environment) concept [1] fits into this gap and supports its users with their design tasks with a mixture of computer-based critics, experiences and examples. A DODE generally allows the construction of its design items (e.g. requirements documentation). It supports its user by critiques based upon rules and metrics, reflecting formal criteria. This helps the user to avoid well-known faults and sticking to quality goals. It is important that the user as a domain expert can add new design knowledge to the set of rules for supporting evolutionary growth of the experiences. Internally, the DODE needs a model of the desired result-type and a catalogue of examples – meta-information that is shared with the corresponding Quality Gateway in our concept. Finally, the simulation component of a DODE visualizes the effects of design decisions (e.g. showing how a given requirement fits into the global context).

Davis [7] describes how requirements should be discovered, pruned, and documented in a project that is subjected to a tight schedule. The key is to apply just enough process to minimize risk while still achieving desired outcomes. In [7] the focus lies on choosing the right requirements, whereas we concentrate on choosing the right quality of requirements. The main problem that remains is: How much exactly is "just enough"? This question has to be answered for each project individually to achieve best results although general advice, tricks, and tips are available. Our concepts help to approach this question an objective and repeatable way that is based on project-specific metrics.

2 Concepts

During each project the decision has to be made whether and when to start with design and implementation using the current set of requirements. This decision is not as difficult in incremental software engineering as in waterfall-like processes because it only affects the first iteration. However, this decision often is not based on the quality and completeness of the requirements specification but instead merely on the project schedule. If this is the case, there are three general options for the project:

1. Abort the project because of high risks.
2. Violate the schedule to give requirements engineers more time analyzing the most important open requirements.
3. Proceed with the requirements already documented and risk problems in the project's later phases.

Our approach strives to give this decision a more objective and repeatable foundation based on measurable criteria, experiences, and metrics.

Even if most projects chose option 3, a minimal quality standard should be defined at the start of the project. Additionally, priorities should be defined in order to ensure that the available time is used efficiently. This can only be done by experienced requirements engineers. Our goal is to support this task and the enforcement of priorities and quality standards with a complementary set of experience-based techniques, practices, and tools. This helps the organization to gather knowledge about what is most important and to improve decisions that were previously based on plain gut feelings.

Our approach is based on a Quality Gateway. Requirements have to pass this gateway before they are allowed to be used in design and implementation. The Quality Gateway defines the quality of the specification in a formal way. A requirement can only pass the Quality Gateway if it fulfills all formal criteria. Since it is not sufficient to check a requirement as an isolated entity, some criteria will judge a requirement in a larger context. Depending on the software development model, it might even be intended that only the specification as a whole can pass the Quality Gateway.

The check is conducted by a Gatekeeper in a Quality Gateway session. Suitable gatekeepers are requirements experts as well as all roles, which depend on the quality of the requirements. For example, a designer, who has to build a software-architecture, and a tester, who has to derive test cases, are suitable gatekeepers. We can understand the Quality Gateway as a continuous process as well as a procedure being conducted from time to time, e.g. whenever there is large set of requirements present or the requirements activities will be stopped (because of a lack of time).

The Quality Gateway is a good way to enhance the quality of the requirements analytically and has been evaluated and found helpful in many projects [8, 9]. Nevertheless, it is limited to detecting shortcomings instead of avoiding them. Therefore, we propose to combine it with a constructive measure. This measure should support its user to meet the quality standards defined by the Quality Gateway. Thus, we implemented a Domain Oriented Design Environments (DODE) [1] to support the design of good requirements in two ways:

1. Distributed experiences (i.e. best practices derived from other projects) are presented by computer-based critiques when appropriate.
2. These critiques help the requirements engineer focus on the most important requirements first, if they have been properly prioritized beforehand.

Our approach uses Quality Gateways to formalize "good enough" in the context of a project. Some requirements might pass the Quality Gateway while others are

rejected. These requirements have to be revised and must be checked again later in the Quality Gateway.

DODEs support requirements engineers in preparing requirements. They do not provide aid for constructing all quality aspects of requirements, but the quality aspects are checked within a Quality Gateway anyway. Since both the Quality Gateway and the DODE concept are based on experiences, we create a common infrastructure for experiences. Fig. 1 summarizes the overall concept.

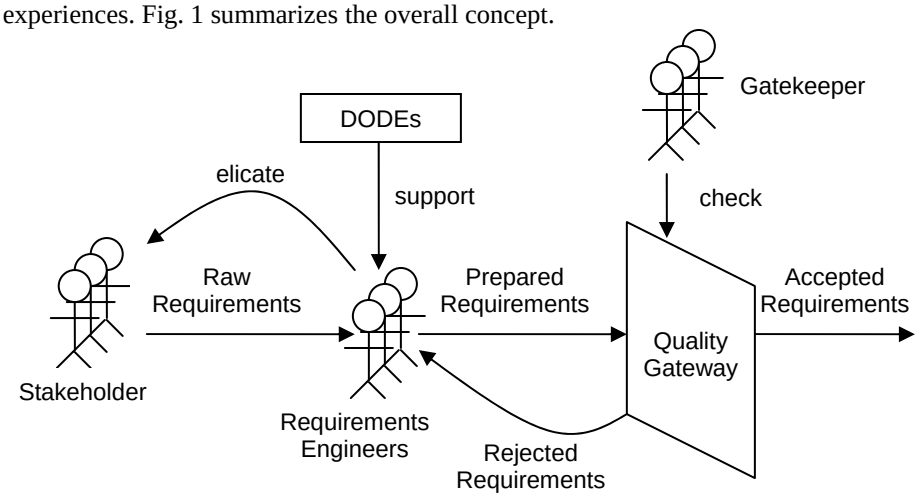

Fig. 1. Combination of DODE and Quality Gateway

Here, the critiques reflect quality goals helping to improve the requirements documentation while writing them down. The combination of this concept with a Quality Gateway promises some positive effects:

1. A DODE can enforce some of the guidelines of the Quality Gateway and reduce expensive rework.
2. A DODE complements the Quality Gateway by improving quality in the small and allowing gatekeepers to focus on more important checkpoints. It is important to mention that criticizing the details helps to detect major problems early.
3. A DODE can comprise recommendations for prioritizing and structuring requirements thereby helping to pursue the most important requirements first. This is an important prerequisite for “Just-enough requirements” decisions.
4. Criteria that can be automatically checked can be moved to the DODE freeing resources for manual and analytical tasks.

The DODE does not help the analyst to decide which Use Cases should be modeled, but helps to focus on the most important quality aspects. The positive effects depend on good guidelines and critiques. In our project we started with a reasonable set derived from literature and improved it based on experiences. Before we show how to use these concepts in a concrete project setting (see section 4) we therefore need to explain the underlying experience management concepts in the next section.

3 Improving Measurement

Experience is essential in order to determine adequate metrics and guidelines for the Quality Gateway as well, as supplemental critiques for the DODE. For two reasons it is important to elicitate and store this experience soon:

1. Experiences will fade over time.
2. Experiences can be reused very soon. Often they are valuable even for the project they originated from.

Additionally, each role's experience is valuable. Therefore, it is important to enable requirements engineers, gatekeepers and other roles relying on the quality of requirements to contribute their experiences.

According to Basili's Quality Improvement Paradigm [10] experiences have to be gathered, analyzed and packaged before they can be used in a project. Elicitation of experiences requires effective techniques such as LIDs [11] or questionnaires. Irrespective of the chosen technique, a special experience engineer has to conduct or at least assist the elicitation procedure. After the elicitation has taken place the experience engineer has to analyze the experiences and package them. This also requires converting subjective experiences into formal criteria whenever possible. However, analyzing and packaging are difficult tasks. Some feedback cycles might be necessary to extract good experiences for some quality aspects. Besides, contradicting or redundant experiences have to be resolved.

After the packaging procedure of an experience has been finished, the prepared experience can be added to the DODE and the Quality Gateway.

Fig. 2 summarizes the overall procedure.

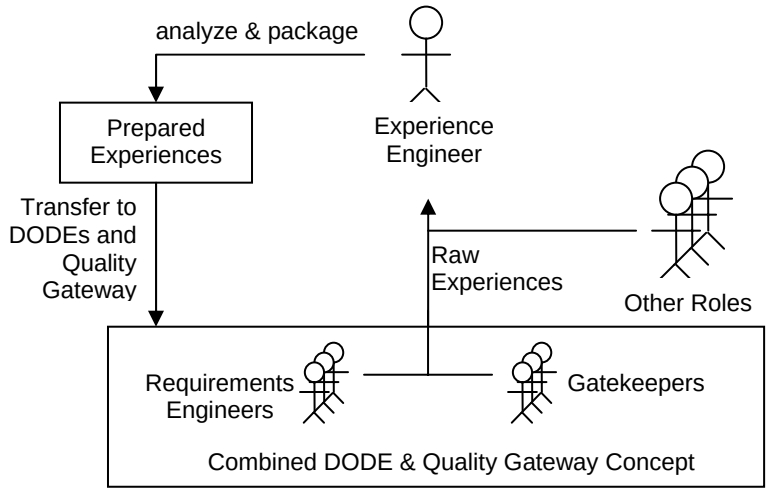

Fig. 2. Integration of Experience Feedback in the Concept

4 Proof of Concept

The concept was incorporated in our students' software projects. All projects were based on waterfall-like models. Consequently, we had only one Quality Gateway after the requirements phase for the whole specification. If the specification failed to pass the Quality Gateway it had to be improved and the Gataway had to be repeated.. We used a DODE for constructing use cases in our proof of concept.

4.1 Installed Quality Gateways

The Quality Gateway procedure was supported by the Tool NetQGate [2]. NetQGate can provide assistance in all steps within the Quality Gateway procedure. It also allows giving feedback for formal criteria. The feedback can be evaluated, packaged and contributed to the experience facility integrated in NetQGate. Table 1 shows one of the criteria contained in our Quality Gateway's checklist:

Table 1. A formal criterion within the NetQGate system

Formal Criterion	Rationale
Are acceptance tests defined in the specification?	Acceptance tests are a contract between customer and development team.

If one of the criteria was not met, the requirements specification was rejected as a whole and had to pass a second instance of the Quality Gateway.

4.2 Use Case Design Environment

The Use Case Design Environment (UCDE) [12] is an example for a tool that influences quality constructively. It helps creating a comprehensive set of use cases based on heuristic rules. Examples for such rules are warnings of possible ambiguities if sentences are formulated using passive forms or certain keywords are used (e.g. "sometimes" and "if" are not allowed in scenario steps, because an extension should be used for alternatives). Table 2 shows an example for the formalization of these rules and Fig. 3 the rules application in the UCDE (this critique is not implemented based on complicated Natural Language Processing but on a heuristic searching for certain keywords).

Table 2. Formalization of the recommendation of active voice

Formal Criterion	Rationale
Is every requirement formulated in active and is always clear, who acts?	It should always be clear, who is responsible for a given task.

The UCDE is also able to check for violations of certain patterns like the *ever-unfolding-story* pattern [13], urging the user to start with high level use cases that are extended by goal-level or even sub-function use cases. Also, the minimum and maximum use case length can be specified. These examples show how writing style and certain properties of use cases can be adjusted while they are written down, influencing the project in a subtle yet effective way. The uniformity of requirements, which is achieved this way, is especially important when comparing requirements, estimating cost, or measuring progress [14].

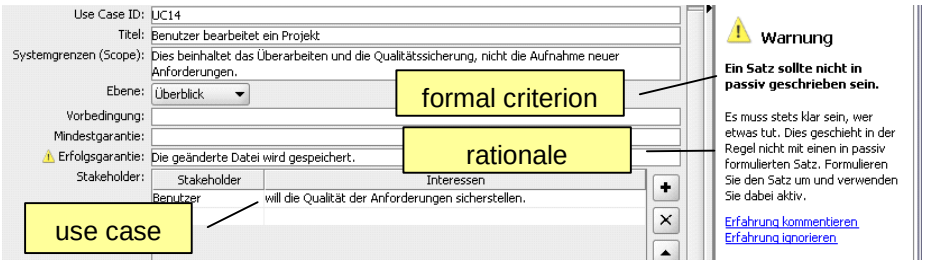

Fig. 3. Warning of the use of passive voice in the UCDE

In our example, the DODE's critiques are matched to checklist items of the Quality Gateway. For example, the UCDE warns the user if no priority is specified for a use case and suggests concentrating on high priority use cases first. The corresponding Quality Gateway checks if all requirements have priorities attached and if the high priority requirements are sufficiently detailed. In this case the quality goal enforced in the Quality Gateway is addressed by the critiques of the DODE during documentation of the requirements.

This example supports one of Davis' central suggestions [7]: always remind the engineer what the goal is. Therefore, it is very important to communicate the project vision and enable the engineer to decide the importance of each requirement. This way, only very important requirements with a given style are submitted to the analytical process of the Quality Gateway.

4.3 Case studies

This section shows the results of our evaluation efforts. The value of our Quality Gateway has been shown in [8, 9]. Therefore, we start with the evaluation of the UCDE in section 4.3.1. In section 4.3.2 we give an example of a project that exceptionally benefited from our approach. Afterwards we present our general experiences in section 4.3.3.

The case studies were obtained from a series of similar software projects conducted at our university as part of the curriculum. The project teams consisted of 5 to 6 computer science students. All students attended the basic classes and some even advanced courses. The setting of our case study has the advantage that projects' environments (e.g. support by teaching staff, time to elicit requirements, etc.) were similar in all projects. We also assigned students to project teams to get teams of

comparable strength, based on information provided voluntarily by the students . Post-Mortem analysis indicates that this was successful.

However, the project setting had some disadvantages related to empirical studies: The software under construction differed between the projects. We had a classic administrative application, an algorithm-centric project, three embedded software projects and three projects based on SOA technology.

The case studies in this section serve to illustrate our approach. Our experiences indicate how the combined use of quality gateway and RE-specific DODE can be beneficial. In addition, the metrics used in our case studies may be useful.

4.3.1 Case Study: UCDE evaluation

We used the UCDE in three out of eight software projects in order to evaluate it. Overall the evaluation resulted in mixed feedback. One project reported that the critiques were rather disturbing and stopped using the design environment. This indicates that a lot of fine tuning is needed. As this project was concerned with embedded software, we assume that the UCDE's generic experiences had the wrong focus for this special setting. In retrospect we find the use cases from all projects dealing with embedded software to differ significantly from the others. Therefore, we decided not to use data obtained from these projects because it is insignificant for the evaluation in the setting provided.

Another group (referred to as project 2) started also with the UCDE and stopped using it before the end of the requirements phase. This was because the project team decided to use other requirements management tools (DOORS from Telelogic in this case) after writing all use cases. The group had problems synchronizing the requirements between both tools and consequentially switched tools at some point in time. We had not expected usage of such tools in the projects and did not provide sufficient support for interoperability.

The third group (project 1) gave very positive feedback. The UCDE was used in this group as intended. The use case model was constructed, exported to a word processor and integrated into the requirements specification. After that, all changes of the use cases where introduced through the UCDE and afterwards exported to the requirements specification, overwriting the last export. Unfortunately the group was closely associated with two other project teams (projects 4 and 5): Some of the requirements were relevant for the other teams, too. In addition, these three teams worked closely together. As a result, the comparison groups benefited from the UCDE, too.

Despite these difficulties we tried to prove or disprove our hypothesis that the UCDE improves Use Case quality based on the GQM (Goal Question Metric [15]) method. We consider some of the achieved results relevant:

For judging the impact concerning the quality of use cases we studied two attributes of the use cases. Through reviews we identified the number of ambiguities in the use of language, and the number of stylistic problems as given in [16]. These questions should help (among others) to evaluate clearness, readability, and homogeneity. We chose to measure stylistic findings per use case. We considered findings of the following categories to be stylistic:

– Wrong usage of the use case template fields (e.g. variations in technology, scenario steps, or goal-level).
– Not defining responsibilities for all actions.
– Not specifying relationships between use cases.
– Incomprehensive parts due to bad grammar or missing information

The UCDE increases the quality of Use Cases as shown in figure 4. Projects 1 and 2 benefited from our experience-based design environment. The use cases of the embedded software projects were not evaluated, because these use cases were of poor quality and the groups did not used the UCDE. The diagram indicates a positive effect of the UCDE in terms of finding per use case.

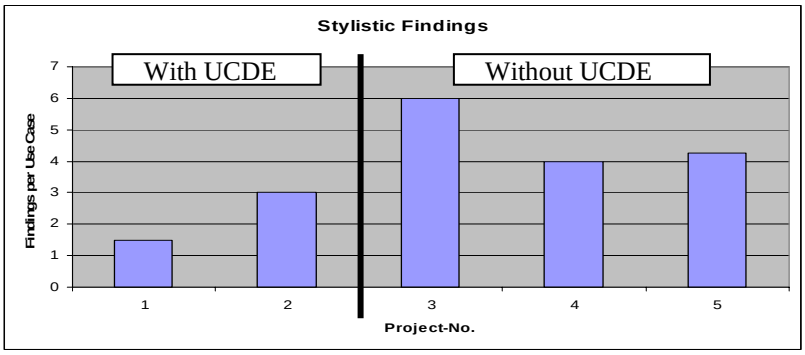

Fig. 4. Stylistic findings per Use Case

All in all, the results of this case study show that the concept of the UCDE is valuable for requirement engineering in projects. The remaining findings show that some of the UCDE's heuristics have been ignored or were even tricked. We conclude that a DODE cannot substitute analytical quality assurance.

4.3.2 Example: Combination of UCDE and Adapted Quality Gateway

After our investigations concerning the UCDE, we were interested in evaluating the effect of its combination with the adapted Quality Gateway. A detailed case study has yet to be conducted. Therefore, we only can provide anecdotal evidence here. Section 4.3.3 presents our general observations in a series of software projects in university teaching. This section presents a project that especially benefited from our approach.

In this project an Experience Management System should be developed. The goal was to integrate the experiences of the Quality Gateway and the UCDE. In this project the UCDE was used with an extension for effort estimation based on Use Case Points [17]. When the initial set of use cases reached the Quality Gateway, it was rejected. Here is a list of the findings:

– Actor/Initiator of step or use case is not clear.
– Extensions were missing. Indicators: Conditions in steps of the main success scenario or a step that could possibly fail.
– Not all steps or use cases were on an appropriate level of description.

These findings indicated that the requirements were incomplete. Furthermore, it was hard to estimate the effort of the project. After some rework the Quality Gateway was passed. However, the UCDE indicated that the effort exceeded the project scope by three times. The estimation was double-checked and found realistic. Therefore, the project leader selected a minimal set of requirements to start with and raised the risk of missed deadlines in his portfolio.

From the observations so far we can conclude, that formal criteria can indicate incomplete requirements. Fast feedback like the DODE's effort estimation extension is a valuable hint for PM. The Effort Estimation is more accurate because of the DODE criticizing writing style and the quality gateway checking for appropriate abstraction levels.

4.3.3 General Experiences

Within our setting (see section 4.3) almost all requirements specifications passed the first instance of the Quality Gateway. Some specifications failed the first instance of the Quality Gateway, but only because of minor defects.

Table 3. Aspects and Functions distributed over UCDE and Adapted Quality Gateway.

Aspect	*Description*	*UC-DODE*	Adapted QG
Completeness	Are all parts of the requirement present?	●	○
Understandability	Can the requirement be understood by any stakeholder?	○	●
Traceability	Can the requirement be traced to its origin?	⦿	⦿
Consistent Terminology	Does the specification use naming conventions and a defined vocabulary?	⦿	⦿
Relevance	Is the requirement relevant to the purpose of the product?	○	●
Ambiguity	Is it possible to determine, whether a requirement is met or not from an objective point of view?	⦿	⦿
Viable within Constraints	Is it possible to meet the requirement when considering the project's constraints?	○	●
Solution-bound	Is the requirement free of a solution?	⦿	⦿
Customer Value	Does the requirement contain a rating indicating the value for the customer?	⦿	⦿
Gold-Plating	Are there any requirements which do not contribute to the product's goal?	○	●
Function	*Description*	UC-DODE	Adapted QG
Requirements Creeping	Are there any mechanisms that systematically manage the change and adding of requirements after the requirement activities are supposed to be finished?	○	●
Requirements Leakage	Are there any mechanisms that systematically prevent the adding of requirements without a traceable origin after the requirement activities are supposed to be finished?	○	●

● Check/Function is performed here
⦿ Check/Function is partly performed here
○ Check/Function is performed here

Moreover, the experience feedback component included in NetQGate was frequently used. Nevertheless, not all experiences were meaningful, because some students misunderstood the feedback system. Consequently, we would prefer a better experience feedback mechanism in the next run of the project. Experiences should be elicited, analyzed and package by a specialized experience engineer.

Concerning the UCDE we observed that its users had better confidence in the quality of their requirements compared to past projects. The project members did not experience any slowdown. Some even reported a speedup because the DODE saved effort for searching relevant information on how to write use cases. The Experience feedback mechanism of the UCDE was poorly used, partly because of its clumsiness and partly because of the lack of experience with use cases. However, the concept succeeded because better use cases were delivered as in comparable projects. In contrast to past projects these use cases were helpful in the following phases, i.e. design, project management, and testing. Thus, we achieved in this context to gain more value in comparable time.

Table 4. Examples of aspects covered by UCDE's heuristics.

Aspect	*Heuristic*
Completeness	Warning, if parts of a template are left empty (e.g. Use Case title), or if a description is too short
Understandability	Warning, if a sentence is too long, an enumeration has too many items, or certain keywords are found (e.g. approx., often).
Traceability	Warning, if a Use Case is a) not on highest abstraction level and b) not linked to a Use Case on a higher abstraction level.
Consistent Terminology	Warnings based on regular expressions (requires naming conventions) and fuzzy searching.
Relevance	See Traceability.
Ambiguity	Warning, if certain keywords are used (e.g. approx., often, always, every, for all).
Viable within Constraints	Warning, if computed Use Case Points exceed a certain level.
Solution-bound	Warning, if certain keywords are used (e.g. button, linux, WLAN, Bluetooth).
Customer Value	Warning, if priority is not set, warning, if priority is lower than priority of dependent requirements.
Gold-Plating	See Traceability.
Function	*Description*
Requirements Creeping	Display development of computed Use Case Points.
Requirements Leakage	Not implemented yet.

4.4 Assessment of the Approach

In this section we assess our approach by comparing it with the Quality Gateway described in [6]. Table 3 shows the aspects and functions normally covered by Robertsons' Quality Gateway. Our goal is to increase the number of quality aspects the UCDE can provide constructive support. The third column indicates how much of the aspect is covered by the UCDE. Because the critiques are heuristic they can only

indicate a possible finding. The absence of a critique has no significance to whether an aspect is violated or not. The author is free to ignore or avoid the critiques. Therefore, it can support the checks only partly.

Table 4 shows examples, how each aspect is addressed in the UCDE. Although we give examples of heuristics for every aspect, we considered the support of some aspects as insufficient (see Table 3).

The UCDE currently provides 57 critiques, with 24 critiques being based on metrics (most often keywords, followed by counting of steps, use cases, etc.). Of course they are not sufficient to completely cover even a single aspect. Nevertheless, these weak indications of possible problems are helpful as shown above. In addition, rules can easily be created and adjusted to fit a project's special situation.

5 Conclusion

In this paper we presented an approach to formalize the decision on when to consider the quality of requirements good enough to proceed. We argued that this decision is normally based on the requirements engineers' personal experience, intuition and plain gut feeling. This results in unreliable and unrepeatable decisions.

Our approach addresses this problem by combining two experience based concepts: On the one hand we have an adapted Quality Gateway process that includes experience based tailored formal criteria.

On the other hand we leverage a subset of the Quality Gateways' formal criteria in a constructive way to lessen rework because of failed quality goals. The DODE concept helps to construct better requirements documentation from the beginning and ensures that experiences are applied in the process of the documentations design.

With the combination of these concepts we are able to establish, monitor, and constructively use a system of metrics. The system of metrics could be reused in similar projects.

We evaluated our approach by applying it to software projects in university teaching. We found that our approach helps to leverage experience from past projects without significantly increasing effort. As a result we got better quality requirements, based on experiences with similar projects, in the same time. Especially quality assurance agents were more confident that the critical quality goals were met without wasting time and effort with unimportant details: Just-Enough requirements through the combination of Quality Gateways and a DODE.

However, we suggest establishing the role of a dedicated experience engineer because of difficulties with the experience feedback. Additionally, further integration of the underlying experience bases could help in this area.

We are aware that our evaluation does not pose any statistical significance. Especially the effects of computer-based feedback in requirements engineering need further investigation. Therefore, we currently evaluate the concept of combining constructive and analytical measures – as presented in this paper – for being applicable in the automotive software industry.

References

1. Fischer, G., *Domain-Oriented Design Environments.* Automated Software Engineering, 1994. **1**(2): p. 177-203.
2. Flohr, T. *NetQGate - Tool Support for Quality Gate Processes*. in *9th International Conference on Quality Engineering in Software Technology*. 2006. Berlin: dpunkt.verlag.
3. Knauss, E., D. Lübke, and T. Flohr, *Learning to Tailor Documentation of Software Requirements.* Journal of Universal Knowledge Management, 2006. **1**(2): p. 103--111.
4. Basili, V., G. Caldiera, and D.H. Rombach, *The Experience Factory*. Encyclopedia of Software Engineering. 1994: John Wiley and Sons.
5. Filho, W.P.P. *Quality gates in use-case driven development*. in *International Conference on Software Engineering*. 2006. Shanghai, China: ACM Press.
6. Robertson, S. and J. Robertson, *Mastering the Requirements Process*. 1999: ACM Press/Addison-Wesley Publishing Co.
7. Davis, D.M., *Just Enough Requirements Management*. 2005, New York: Dorset House Publishing.
8. Lübke, D., T. Flohr, and K. Schneider. *Serious Insights through Fun Software-Projects*. in *EuroSPI 2004: European Software Process Improvement Conference*. 2004. Trondheim, Norway: Springer.
9. Lübke, D. and T. Flohr. *Experiences from the Conducation of a simulated Software Project driven by Quality Gates*. in *TESI 2005*. 2005. Maastricht, Netherlands.
10. Basili, V.R. *The Maturing of the Quality Improvement Paradigm in the SEL. Presentation*. in *Nokia Research Centre: Software Engineering Workshops*. 1994. Helsinki, Finland.
11. Schneider, K. *LIDs: A Light-Weight Approach to Experience Elicitation and Reuse*. in *Product Focused Software Process Improvement (PROFES 2000).* 2000. Oulo, Finland: Springer.
12. Knauss, E., *Einsatz computergestützter Kritiken für Anforderungen.* GI Softwaretechnik-Trends, 2007. **27**(1).
13. Adolph, S. and P. Bramble, *Patterns for Effective Use Cases*. The Agile Software Development Series, ed. Cockburn and Highsmith. 2003, Boston: Addison-Wesley.
14. Gorschek, T. and C. Wohlin, *Requirements Abstraction Model.* Requirements Eng, 2005. **11**(1): p. 79-101.
15. Basili, V., G. Caldiera, and H. Rombach, *Goal question metric paradigm*, in *Encyclopedia of Software Engineering*, J.J. Marciniak, Editor. 1994, John Wiley & Sons: New York. p. 528-532.
16. Chockburn, A., *Writing Effective Use Cases*. 2001: Addison -Wessley.
17. Schneider, G. and J.P. Winters, *Applying Use Cases: A Practical Guide*. 1998: Addison-Wesley.