Agile Software Quality Function Deployment

Sixten Schockert
University of Stuttgart, Chair of information systems II (Business Software)
Keplerstraße 17, D-70174 Stuttgart, Germany
+49 (0) 711 685 82387
schockert@wius.bwi.uni-stuttgart.de
and
QFD Institut Deutschland e.V., Germany
schockert@qfd-id.de

Prof. Dr. Georg Herzwurm
University of Stuttgart, Chair of information systems II (Business Software)
Keplerstraße 17, D-70174 Stuttgart, Germany
+49 (0) 711 685 82385
herzwurm@wius.bwi.uni-stuttgart.de
and
QFD Institut Deutschland e.V., Germany
herzwurm@qfd-id.de

ABSTRACT
This paper introduces Agile Software Quality Function Deployment. It represents an evolutionary development of dynamic Software QFD models, like Continuous QFD, with regard to the application within agile software development. The proposal for Agile Software QFD is characterized by the embedding in the agile iteration cycle and distinct methodological features such as the incrementally growing prioritization matrix and the priority map. It is based on and evaluated against 27 design requirements on agile Requirements Engineering which are derived from the principles and values of agile software development, the handling of requirements in agile development models and empirical sources of agile Requirements Engineering. Both agile software development and Software QFD benefit from Agile Software QFD: it is the expression of a truly business value oriented, agile requirements engineering embedding QFD in an iterative and incremental development process.

KEYWORDS
Software QFD, Agile software development, Agile Requirements Engineering, Agile Software QFD

1. INTRODUCTION
The way to use software and the consequences of using it cannot be predicted reliably. Together with the constant and rapid development of the applied technologies this has led to the fact that there is no longer a question of whether or not there will be changes in requirements. They always exist. Agile software development has emerged directly in response to this unstable and turbulent business world. Inherent component of agile software development in increments is the evaluation and prioritization of requirements as part of Requirements Engineering activities. This task of focusing development activities in the early phases of software development is the domain of Software QFD. Software QFD represents a variant of Quality Function Deployment (QFD) for the development of software products.

Both Software QFD and agile development methods define the customer's satisfaction as the top maxim for their doing. And although Software QFD has its strength in the transformation of customer needs into prioritized product requirements, its integration with agile development methods has not yet taken place. A current comprehensive analysis of Software QFD literature underpins this fact [14]. The question as to whether Software QFD will play an important role in agile surroundings or be of decreasing relevance is currently open.

This paper introduces Agile Software QFD as an evolutionary approach to Software QFD. In particular Agile Software QFD represents a further development of dynamic Software QFD models like the so-called Continuous QFD [11, 13] with regard to the application within agile contexts. The focus of this paper is on the methodical description of the artefacts of Agile Software QFD rather than their well-founded deduction and practical evaluation. The latter can both be found in detail in [24]. After an introduction to agile software development fundamentals and basic practices of agile Requirements Engineering (section 2) the techniques and methodological features of Agile Software QFD are presented (section 3). Section 4 gives an overview of the comparison of Agile Software QFD with common agile Requirements Engineering practice as well as existing Software QFD models based on certain evaluation criteria. The paper ends with some few concluding remarks regarding possible future work (section 5).

2. AGILE SOFTWARE DEVELOPMENT

Over the past few years, the agile development paradigm has become the standard way of work in many software development projects. In a current study, more than 60% of 368 IT managers consider Scrum, the most widespread agile development model (3 out of 4 agile applications are related to Scrum [31]), to be central to their daily work, a further 25% use Scrum among other methods [19]. Constituent to the agile paradigm in software development are the values and principles mentioned in the Agile Manifesto [5]. Only if methods base on these values and principles they are regarded as agile [2]. Thus, section 2.1 briefly describes these origins of agile development before in section 2.2 and 2.3 the basic agile development cycle and core concepts of agile Requirements Engineering are presented. Altogether these concepts form the framework for an Agile Software QFD to fit in.

2.1 Core ideas and origin of agile software development

In software engineering, agility is directly associated with the agile paradigm that has developed around the turn of the millennium. It emerged as a reaction to the constantly changing, increasingly unstable and turbulent business world. The times of deterministic, sequential development processes, in which the requirements are frozen at the beginning of the development and changes in requirements are only treated as exceptions, were things of the past. Requirements change in every case, and thus ways have to be found to deal efficiently with these changes. Agility stands here for the ability to handle these changes efficiently, aiming at a kind of balance between structuring and flexibility. As such there is no ONE agile software development model. The term is generally understood as an umbrella for methods and techniques, which are based on the values and principles of agility and the agile manifest [2].

These values and principles have remained unchanged to the present day. For methods as well as for humans they represent the main indicator of their "degree of agility". The Agile Manifesto has been launched by practitioners: "We are uncovering better ways of developing software by doing it and helping others do it." [2] In this sense, e.g. Scrum represents a framework that has emerged from successful work in practice and has gained resp. continues to gain knowledge from experience [25, 26, 27]. On the basis of their experience, the authors defined four guiding principles in which they juxtapose possible values of software development (Table 1).

Table 1: Guiding principles of the agile manifesto [2]

General principles of the agile manifesto		
Highly appreciated values	**are / is more important than**	**Less appreciated values**
Individuals and interactions		Processes and tools
Working software		Comprehensive documentation
Customer collaboration		Contract negotiation
Responding to change		Following a plan

The founders emphasize that the values on the right side possess significance. However, they consider the values on the left side for an agile software development as more worthwhile. Compulsory procedures and methods, extensive documentation, elaborated contracts and plans are not an end in themselves. They have to demonstrate their usefulness in real settings. When challenges change, problems need to be solved quickly, or other things alter the original situation, the team should react adequately [16]. In this sense, the methods and techniques described in the agile models do not reflect any strict rules to be observed, but they provide a flexible framework in which the abilities and creativity of individuals can unfold. The assumption is that simple, clear rules stimulate intelligent, innovative behavior, whereas complicated rules tend to lead to simple, less creative behavior (so-called "generative rules" [15, pp. 183-187]).

In order to strengthen these demands for more self-determination and freedom, the authors of the Agile Manifesto formulated 12 principles which concretize the stated values (Table 2).

Table 2: Principles of the Agile Manifesto [5]

12 Principles of the Agile Manifesto	
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4	Business people and developers must work together daily throughout the project.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity - the art of maximizing the amount of work not done - is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

A more precise definition of agile software development on the basis of the agile manifesto will be always incomplete, since not all principles and values can be packed into one compact phrase. Therefore many authors shy away from this task, the definition is made through the Manifesto. However, the International Standards Organization (ISO) cannot duck out of a definition:

“Agile development (is a) software development approach based on iterative development, frequent inspection and adaptation, and incremental deliveries, in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback.” (ISO 26515, [17])

2.2 Basic agile development cycle

According to the agile principles for continuous delivery of software (principles 1 and 3), an iterative approach in short cycles (often called “sprints” like in Scrum) is used where the product grows incrementally. However, just as there is no ONE agile development model, there doesn’t exist the one and only agile software development process. Such a prescription would completely contradict the agile values of self-determination and flexibility. Nevertheless some agile best practices can be identified which are applied in more than 70% of all agile projects [30, 31]:

- Use of so-called “user stories” (in the sense of product requirements) as elements of the so-called (prioritized) “product backlog”. These user stories are regularly analyzed and a distinct amount of them is selected for implementation within the next iteration (sprint backlog).
- Daily short (maximum 15 minutes) status meeting of the team to inform about each individuals work progress (so-called daily stand-up meeting, daily scrum).
- Review of the iteration results by demonstrating the implemented new or modified functionalities of the software to the team and/or stakeholders (review meeting, sprint review).
- Retrospective of the iteration to improve work and collaboration in the next iteration.

On the basis of these techniques, a simplified iterative agile process can be derived (Figure 1).

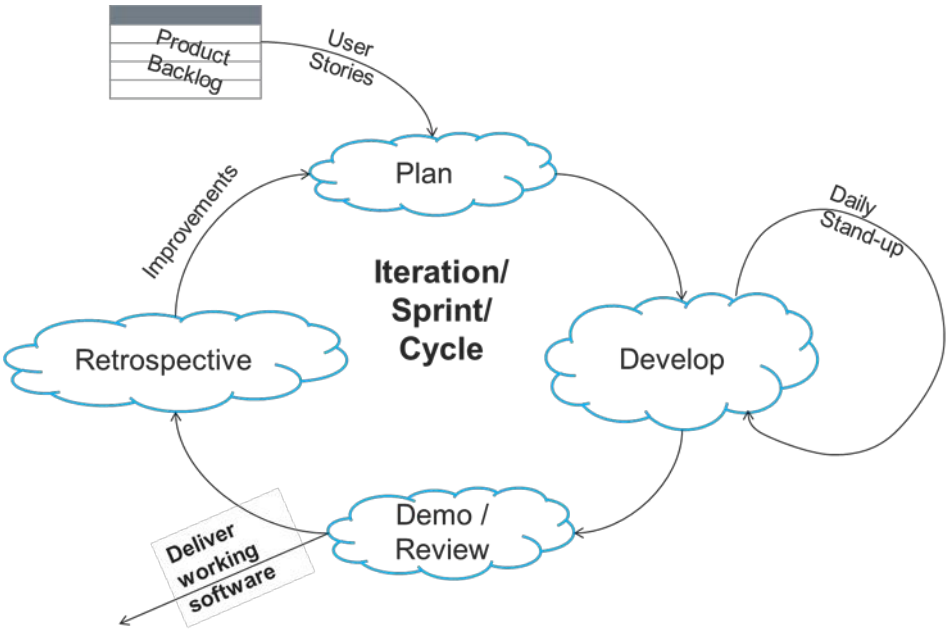

Figure 1: Simplified basic agile development cycle (according to [8])

Product requirements in terms of user stories are selected according to their prioritization in the backlog for implementation in the next increment (planning, sprint planning). During one cycle, the user stories remain stable resp. unchanged. They are designed, coded and tested (development). Every day, the team members report on their individual progress (daily stand-ups). The result of an iteration is demonstrated to the team (and beyond) and, if necessary, the working software is delivered to the customer (demo/review). Already during each iteration, the team should reflect on its work. Moreover there is a lessons-learned meeting at the end of each iteration to recognize improvements for the next run (retrospective). As one can see, this simplified agile process corresponds to the Plan-Do-Check-Act cycle (PDCA cycle) for continuous improvement according to Deming. [8].

To give a concrete example, the ISO has formulated its definition of Scrum according to this basic cycle: “Scrum (is an) iterative project management framework used in agile development, in which a team agrees on development items from a requirements backlog and produces them within a short duration of a few weeks.” (ISO 26515, [17]) The iteration, in Scrum called sprint, has a length of 2-4 weeks. In practice two-week sprints predominate in about 60% of the cases. The number of iterations per se is not limited, in practice the average is around seven iterations per project [27].

2.3 Core concepts of agile Requirements Engineering

Agile Requirements Engineering as self-contained task with dedicated activities doesn’t exist [24]. Thus, in a quite comprehensive list of agile practices of the “Agile Alliance” (a non-profit organization of agile supporters) there are only very few techniques directly related to Requirements Engineering. Most of the practices deal with issues of design, programming or testing of software, the focus lies clearly in implementation-related areas (Figure 2).

Practices somehow related to Requirements Engineering are “hidden” in the Product Management section like the use of Personas (kind of stakeholder/customer identification), user story related activities (like Story Mapping [22], Story splitting) and INVEST, an acronym for a list of agile quality criteria: independent, negotiable, valuable, estimable, small and testable (for short descriptions of these practices look e.g. [3]). Specific Requirements Engineering techniques, e.g. for the elicitation of requirements, are not included. This reflects to the self-understanding of the agile approaches. Demands and requests for new functionalities as potential items of the product backlog emerge at any time due to the intensive communication within the team and the feedback on the increments. Then these potential requirements have to be analyzed, but they do not represent a (larger) problem in their determination. And if there are, for example, demands for certain specification documents or formalized requirements models, then these tasks have to be accomplished. But such work takes place “outside” the agile development, from the agile point of view these are unnecessary activities.

To take Scrum as an example, the handling of requirements in Scrum is rather simple. All requirements, or statements, regardless of level of detail or type, are entered into the product backlog through the only input channel, the so-called product owner. He analyzes and ranks the entries according to their contribution to business value and at the start of each iteration the Scrum team (especially the developers) selects the items to be implemented next. In practice, these items mainly represent user stories which as such are the main form of communication and documenting requirements in agile development. The product owner has a clear vision for the future of the product and thus motivates all people involved. He is solely responsible for the value maximization of the product and the work of the development team. He is contact

person for all questions related to the requirements, he is the "requester" and as such (in almost two thirds of all cases [28]) THE (and then only) interface to the customer.

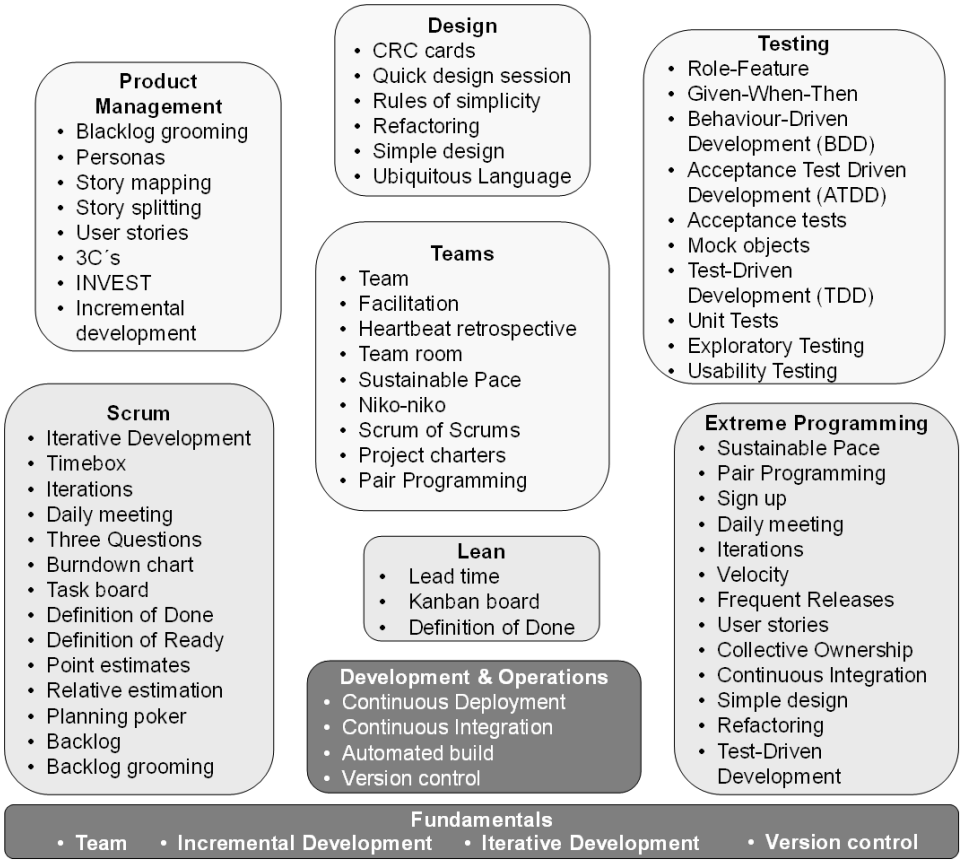

Figure 2: Overview of agile practices (according to [4])

3. AGILE SOFTWARE QFD

This section presents the core artefacts of Agile Software QFD.

3.1 User Stories from QFD perspective

An application of Software QFD in agile contexts has to handle user stories as main requirements representatives. In general, a user story refers to a product function that has a certain value for users or buyers of a software product. The focus is usually (but not always) on functionality, since software differentiates itself in most cases by its behavior. The value directly corresponds to a higher business value from the perspective of the development organization. There is a widely used and established standard template for the formulation of a user story:

"I as a (role) want (function) so that (business value)." [7, pp. 81]

Main advantages of the template are that both the relationship to the stakeholder as "requestor" of a function as well as the intended effect and benefit of the function are made explicit. Looking from a QFD perspective on this phrase lead to the central link of Software QFD with to the agile world. A user story can be restated as following:

"As a (stakeholder) I want (product function) to satisfy (customer need)."

Or even more abstractly: "As (WHO) I want (WHAT) to satisfy (WHY)."

Thus, the user story can be split explicitly into its three components, the "who", "what" and "why" parts:

- "Who": Who are the stakeholders? Who are the buyers and users?
- "What": What do stakeholders want? What do buyers and users want?
- "Why": Why do stakeholders want it? Why do buyers and users want it?

It is not only essential for an Agile Software QFD to be capable of handling user stories to be accepted in the agile world. Moreover with its central parts, especially the "why"-component and its link to the "what"-component, a user story represents a kind of strong or at least medium correlation in a QFD matrix.

But user stories only combine exactly one (!) need with exactly one (!) potential solution. Such 1:1-relationships between needs and solutions are anything but the normal case, because product features rarely have a singular effect on exactly one need, simply because functions can often be used in different contexts and by different users. Just as needs can have different solutions, functions or more general product requirements, can satisfy several needs (at least partially). The agile development abstracts from this fact. This corresponds to the situation in the matrix diagram on the left in Figure 3 with only 1:1-relationships in which each matrix cell represents a user story.

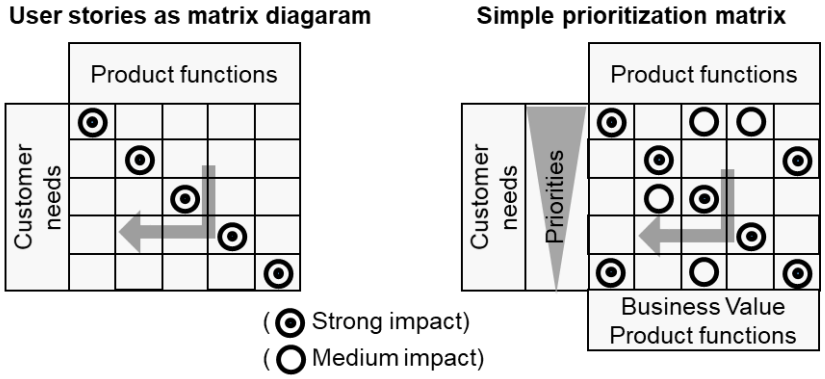

Figure 3: User stories vs. QFD prioritization matrix [24]

Even with only one further dependency on one other need, alternative solutions could be more promising and many-to-many prioritization matrices might be useful for further analysis (Figure 3, right side). Remarkable is, that simple symbolism and visualization are sufficient to achieve a prioritization of the functions in the backlog on the basis of the mostly only ordinal ranking scale used in agile contexts (indicated by the priorities triangle in Figure 3, right side). If the needs are placed in a simple ranking according to their business value, all the effects listed in the upper lines are to be judged higher than the lower ones. The focus is on the strong and medium impacts; weak effects, as also used in classical prioritization matrices, are omitted as they do not help in the differentiation of the solutions. With its multiple effects on several needs one could even say that each column of the prioritization matrix represents an "extended user story", not only focusing on one relation of a solution to one need but on many relations to many needs.

3.2 Incrementally growing prioritization matrices

The elaborations in the previous section 3.1 indicate a first potential benefit of using Software QFD within agile development due to discovering alternative and even better solutions for customer needs than simply relying on 1:1-relationships represented in user stories. But prioritization matrices in their standard form are often time-consuming and even boring to build due to the many possible correlations. This problem of the tedious and uninspiring nature of the matrix (and even of QFD as a whole) has to be tackled. One way is to let the matrix grow incrementally instead of trying to build it as a whole. Starting off, the matrices are already pre-filled on the diagonal by the elements of the known user stories (and possible further dependencies). Secondly, the matrices stay rather small because they aim "only" on deciding the functions to be implemented in the next iteration. Easy done ranking of the needs with respect to their business value is in most cases sufficient to determine these functions. By letting the matrix grow row-by-row from a base of the most important needs in the top left corner one comes up with potential new solutions to the right and even possibly new needs (important!) downwards. Figure 4 illustrates this growing schematically.

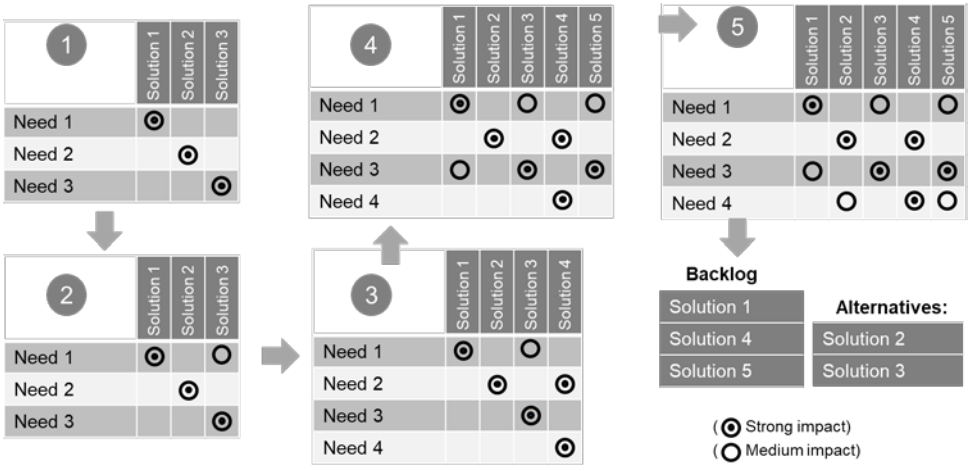

Figure 4: Incremental growing prioritization matrix on the basis of known user stories [24]

Starting with three stories (1) the analysis progresses row by row, i.e. need by need. The central question is how the need can be satisfied, or whether there is, among the existing solutions, one that contributes to its fulfillment, too. Thus, in (2), solution 3 becomes more important. Going on (3), the analysis of need 2 reveals a new solution 4, but also a "new" need 4, so that now solution 2 is dominated by the new solution 4. With the analysis of need 3 (4), a new solution 5 is added, which now dominates solution 3. In the last step (5), solution 2 appears as an alternative to solution 4 again (just as solution 3 remains an alternative to solution 5). If matrix (5) in Figure 4 would be the final result, the backlog order would be solution 1 before 4 before 5. And the analysis comes up with (additional) information that there are two alternative solutions, 2 and 3, which are either not realized at all, since they have less positive effects on the business value than 4 and 5. Or they can be used as alternatives, e.g. if there are greater risks or higher efforts accompanied by implementing solutions 4 and 5.

This row-by-row approach for incrementally growing the matrix can even be used to determine possible backlog items along the most important needs without direct reference to known user stories (Figure 5).

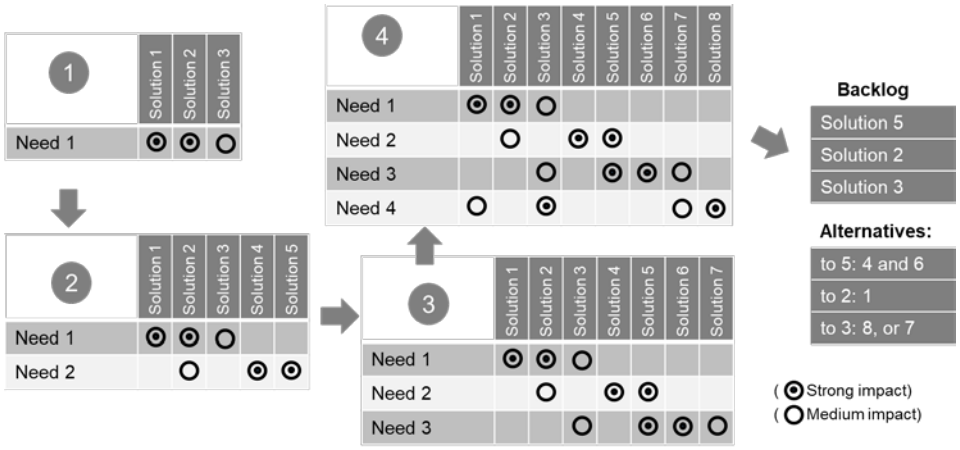

Figure 5: Diagonal incremental growing prioritization matrix [24]

Starting with the most important need 1 (1), the solutions with the strongest impacts on the fulfillment of need 1 are determined and, if appropriate, those with a medium effect. With step (2), solution 2 gains an advantage over solution 1 since it also acts on need 2. With step (3), solution 5 becomes most important according to business value because of the strong effect on two needs. Step (4) clarifies that solution 3 will be ranked third in the backlog as a result of its strong impact on need 4. There are, however, a number of alternatives for setting up a backlog order (at the bottom right in Figure 5).

The matrix in Figure 5 grows through the line-wise approach of finding solutions for needs only below the diagonal. One further benefit is that one can stop the matrix creation after each step since the most important needs always are considered adequately (i.e. there are strong effects). This fact is very important especially in agile contexts with often tight schedules and limited resources. Thus, the incrementally growing prioritization matrix introduced in this section reduces the "fear of the boring matrix tool".

3.3 Priority Map as product backlog

A prioritization matrix enhanced with further assessments of risks, dependencies and so on is called House of Quality (HoQ) in QFD [e.g. 6]. Of course, the House of Quality is only one part of QFD. But it is for sure the most famous one. And – as the critical analysis of Software QFD literature showed – there are even around 40% of the Software QFD applications that focus solely on the HoQ, moreover almost 90% out of them describe a case study [14]. Considering agile contexts, the focus on user stories and their ordering/ranking according to their contribution to business value as a basis for being selected for implementation during next sprint, it is even possible to visualize the product backlog "only" by a HoQ.

This "extended" HoQ acts like a compact Priority Map in guiding the agile development. The solution space consists of all possible kinds of requirements, esp. product functions and quality characteristics. Their positive as well as negative dependencies can be documented in the roof. As mentioned in section 3.1, a matrix column can be understood as an "extended user story" because it shows all the effects a product requirement has on the needs. A line shows all the solutions characteristics which have an effect on a particular need, especially possible alternative designs characterized by several strong impacts. The evaluations of the solutions can be noted at the bottom of the matrix and the product requirements to be implemented in the next increment can be selected on the basis of all evaluations including the dependencies. Similar to the so-called outcomes of the agile technique Story Mapping [22], an aggregated and motivating target can be derived from the strong effects of the selected product requirements. By focusing on the prioritization

aspects, the extended HoQ, very similar to the "story map as a product backlog" [21], can also be referred to as "Priority Map as the Product Backlog" (Figure 6, left side).

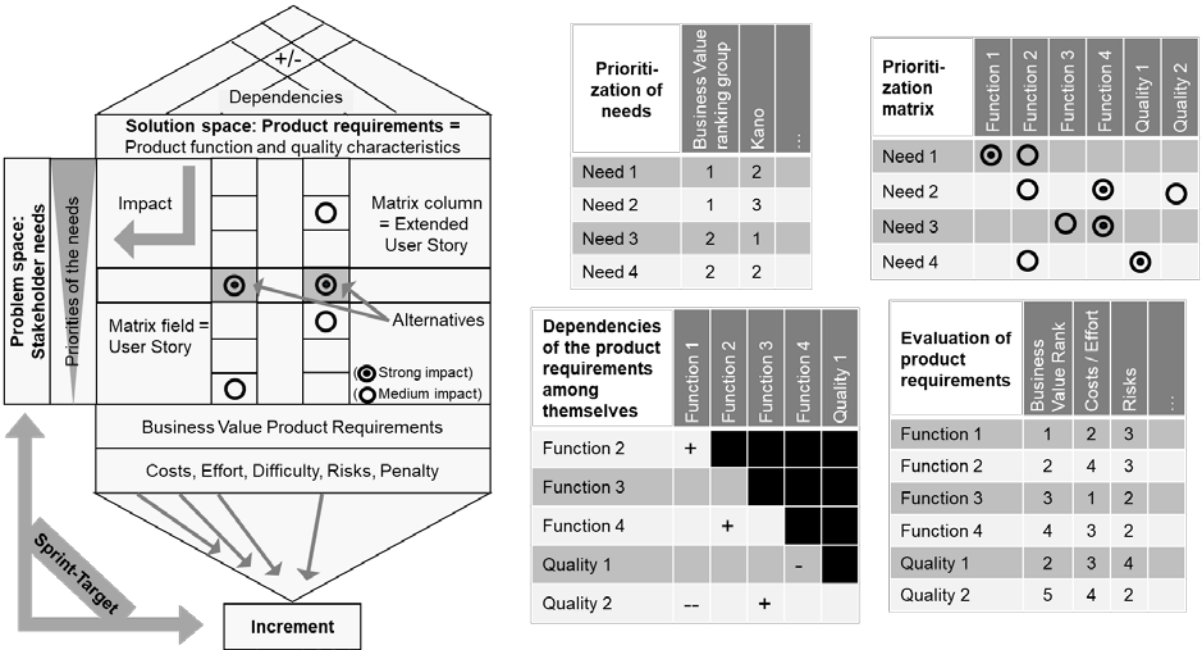

Figure 6: Priority Map – Extended House of Quality representing the Agile Product Backlog [24]

The Priority Map visualizes very compactly and tightly arranged all possible aspects to be taken into account when analyzing and refining the product backlog. In meetings, especially planning and so-called grooming meetings in an agile development cycle (see also next section 3.4), it could be of use to keep the matrix contents in the background to avoid misleading discussions. The Priority Map is then only consulted for deeper argumentation. In such cases it is advisable to divide the Priority Map into its four essential components to remove some complexity from its representation and visualization (Figure 6, right side).

3.4 Iteration cycle of Agile Software QFD

Using QFD techniques in agile contexts is – when making some adjustments – possible and worthwhile as the previous sections showed. But decisive for their application is their integration within the agile iteration cycle. The QFD's main activities of elicitation, agreeing upon and implementing requirements [18, 24] are well reflected in it (Figure 7).

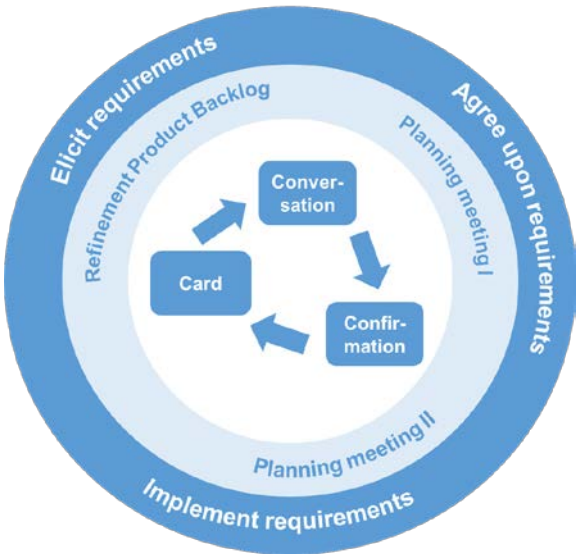

Figure 7: Schematic embedding of the QFD main activities into the iteration cycle of agile development [24]

Thus, in each iteration there is a refinement of the backlog (grooming meetings) in the sense of requirements elicitation. Based on the results of these meetings the team members have to agree upon the requirements to be implemented next in a first planning meeting. Finally, the implementation of the selected requirements start with a second planning meeting

[27]. The so-called "3Cs" of the User Stories help associating QFD main activities with these agile process steps: "Card" represents the elicitation and description of user stories; "Conversation" stands for the discussion and cooperation between the stakeholders to clarify the user stories; and "Confirmation" is about detailing the users stories and transforming them into defined development tasks (sprint tasks) [7, 22]. Figure 7 shows these relationships schematically. Figure 8 concretizes this general connection between QFD and agile development activities in a certain iteration resp. sprint of Scrum.

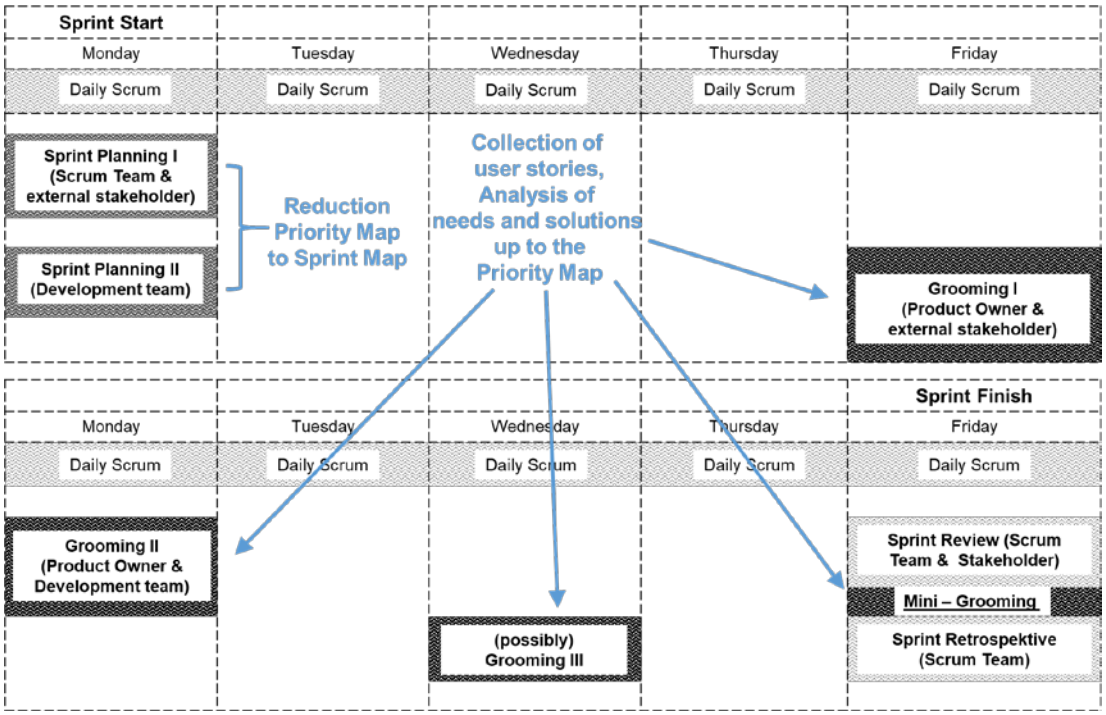

Figure 8: Typical 14-day iteration/sprint with Agile Software QFD [24]

In Agile Software QFD the first planning meeting should take place with external stakeholders, especially customers to clarify details and to achieve an agreement and commitment on the next iterations doings among all participants. This stands in sharp contrast to the usual practice within e.g. Scrum where the stakeholders are only "informed" at the end of day 1 of the sprint (this behavior is in line with the product owner being solely responsible for the product backlog, cf. section 2.3). At the end of planning meeting I, a sprint target (cf. section 3.3) has to be determined serving as joint commitment of the development team to the work of the next sprint.

The second planning session on the same day is executed very similar as in "normal" agile development. It results in a selection of product requirements to be implemented. This means that the Priority Map is reduced to a so-called "Sprint Map" consisting only of the needs and solutions (or more precise the "extended user stories") to be tackled in the next sprint. As a consequence, the priority map to look at in further iterations is reduced with each sprint: the needs which have been fully-satisfied are deleted as well as the solutions that have been already implemented in the product (see also step (8) in Figure 9 below).

The so-called first grooming meeting is about the further refinement of the backlog with the external stakeholders (mainly customers and users). In the second and third grooming meetings, estimates and assessments are updated and, above all, new solutions are sought to satisfy new stakeholder needs identified in Grooming I. In order to consider the feedback on the working products performance directly in the next iteration a mini-grooming is introduced (in contrast to usual agile practice) immediately after the review meeting and before the start of the next cycle. The daily status meetings as well as the retrospective meeting on the experiences with the past iteration remain unchanged as in usual Scrum practice.

The cyclic and interdependent sequence of Agile Software QFD can thus be summarized in a graphical way as shown in Figure 9. It starts of (1) with the collection and elicitation of user stories together with the stakeholders. After (2) splitting the user stories analytically into customer needs and product requirements the assessment of the needs (3) by the stakeholders according to the business value takes place. Next (4) alternative solutions and possible synergies to meet the needs are identified. The (5) connection of the needs and requirements is taking place in an incrementally growing prioritization matrix. (6) Further evaluation of the requirements and their representation as extended user stories is done within the Priority Map. To reach the aligned Sprint target extended user stories are selected (7) in the Sprint Map for realization in the next increment. The Priority Map is finally reduced (8) on the basis of the realized and fully fulfilled needs.

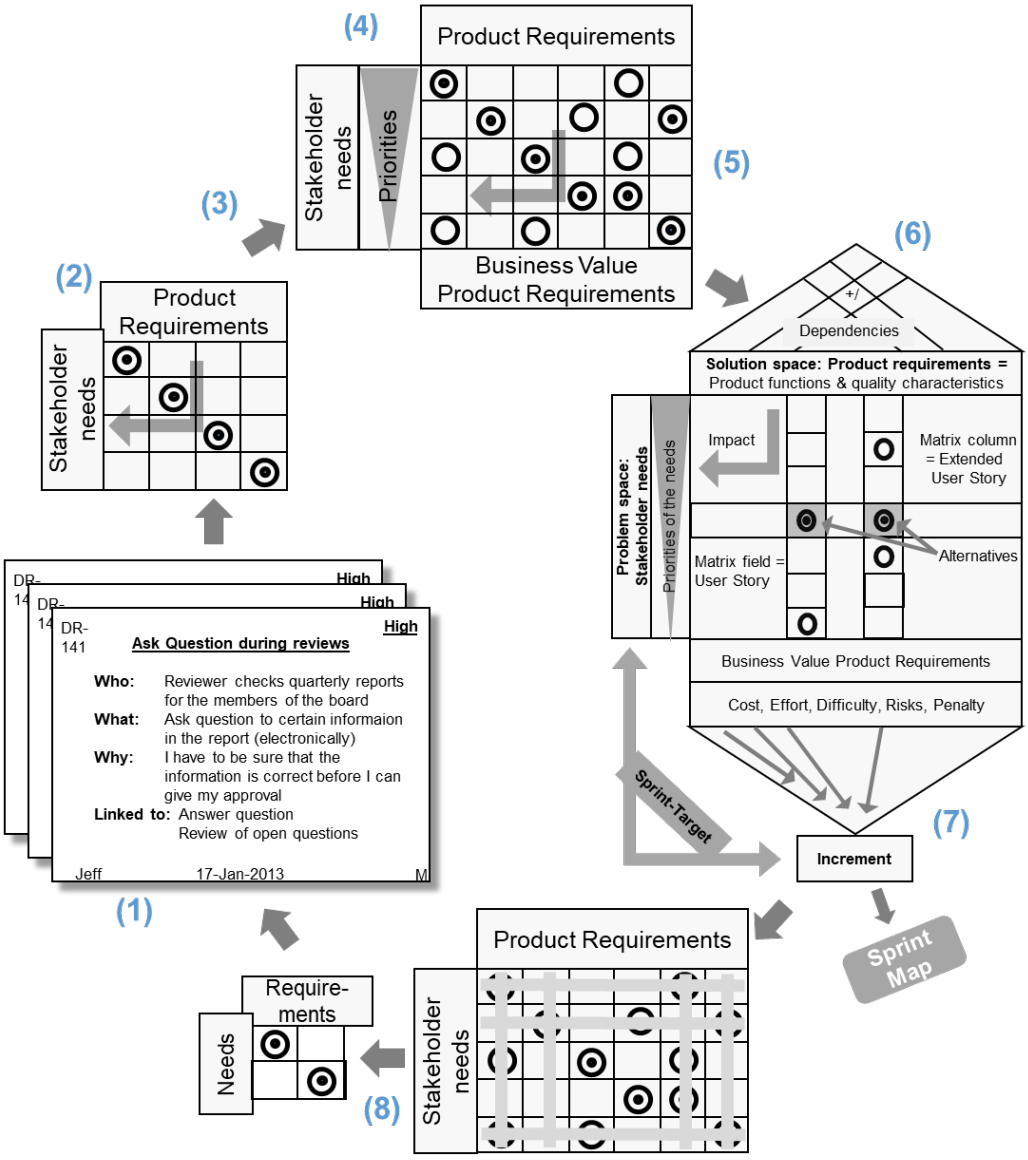

Figure 9: Iteration cycle of Agile Software QFD [24]

To summarize, the essential techniques of Agile Software QFD are the following (for more detail, especially on the techniques not mentioned in this paper see [24]):

- Use of (extended) user Stories from their collection in Grooming Meeting I to their selection in the planning meeting I.
- Splitting the user stories into stakeholder needs and product requirements (in particular by means of the questions about “why” and “what for” a solution is required).
- (Simple) prioritization of the needs by the stakeholder by means of rank group formation or ranking.
- Linking stakeholder needs and product requirements by…
    - …incrementally growing prioritization matrices.
    - …relation diagrams in the sense of the maximum value table (see also [20]).
- Assessment of product requirements regarding possible dependencies, costs, effort, difficulty, risks of implementation, and other criteria like the penalty of not fulfilling a requirement
- Priority Map for the compact presentation of the priorities of the stakeholder needs, the prioritization matrix itself as well as the evaluations of the product requirements and their dependencies amongst each other
- Sprint Map with the visualization of product requirements selected for the next increment and stakeholder needs to be met (balanced with the Sprint target).

4. EVALUATION OF AGILE SOFTWARE QFD

In this section Agile Software QFD is compared against common agile Requirements Engineering as well as existing Software QFD practice. Overall, the values and principles of the Agile Manifesto remain the basis of any agile development practice. "All agile teams choose among software development practices, but, if they want to be agile, they should choose practices that are in line with the principles." [34, pp. 73] Therefore, the design of Agile Software QFD should not be in contradiction with the agile principles, or even better, Agile Software QFD should proactive fulfill them. For this reason the agile principles form the basis for the evaluation of the proposed Agile Software QFD techniques and methodology features. Moreover, Agile Software QFD must be able to take on tasks arising from the handling of requirements in agile development practice so that it fits smoothly within agile frameworks like Scrum. Therefore, the widespread techniques in dealing with requirements in agile development practice are sources of requirements for the design of the Agile Software QFD as well as sources for potential evaluation criteria. For a detailed description and analysis of these criteria as well as the evaluation itself see [24]. In the following only the main differences and main findings are highlighted.

4.1 Comparison with Software QFD practice

Table 3 gives an overview regarding the main advantages and disadvantages comparing Agile Software QFD with other Software QFD models. To distinguish the approaches we used the classification of Software QFD models from [13]. This means, we compared Agile Software QFD to traditional and (more or less) comprehensive Software QFD models merely adapting the classical way of doing QFD in manufacturing (e.g. [1]) to software development (like e.g. [29, 35]); to more specific Software QFD models focusing on the Requirements Engineering and product planning phase of software development aiming at an agreement on product requirements to be implemented (like e.g. PriFo-QFD [11] or Blitz/Modern QFD [20]); and to Software QFD models which try to apply QFD in an iterative way to consider explicitly the dynamics resulting from changing needs and/or solutions (like e.g. Watanabe's twin peaks approach [32, 33]) and Continuous QFD (CQFD, 10, 11, 12]).

Table 3: Comparison of Agile Software QFD with Software QFD models (according to [24])

Agile Software QFD	Agile Software QFD vs. Software QFD models	Dynamic (like CQFD)	Focused (like PriFo-QFD)	Traditional compre-hensive
+ / ++	A1: Focus on the business value of the requirements	+ / ++	++	+
+	A2: Support procedure in short, regular iterations	O / +	-	--
+	A3: Artefacts can be detailed and can grow incrementally	O / +	O	--
+	A4: Get Feedback to RE-Artefacts	+	O	-
+	A5: Could make changes to RE-Artefacts fast and easy	O / +	O	--
++	A6: Support cooperation of customers and developers	++	++	+
O / +	A7: Provide sustained motivating procedure of work	O	O	--
+ / ++	A8: Deliver authentic and credible procedure and results	+	++	+
++	A9: Support of direct personal communication	++	+	+
+ / ++	A10: Consider quality requirements and design constraints	+ / ++	+	++
++	A11: Focus on the essential doings	++	++	O
O / +	A12: Facilitate self-organization in teams	O	-/O	-
++	A13: Support the commitment to common goals	+ / ++	+/++	O

Compared to typical dynamic Software QFD models like CQFD, Agile Software QFD stands on the same level regarding six criteria and dominates with respect to the remaining seven aspects. This is not surprising because Agile Software QFD represents obviously an evolutionary further development of these dynamic models, they both share the same motivation of corresponding to the ever faster changing and innovative business world. Agile Software QFD outperforms former dynamic models at criteria A2, A3 and A5 regarding an iterative, incremental procedure with the possibility to incorporate changes and feedback fast and easy. This results from the now improved embedding into an iterative development process (section 3.4) and the concretization of the incrementally growing prioritization matrix (section 3.2). In conjunction with the lesser need for moderation in Agile Software QFD, the ability to build the Priority Map in components separately (section 3.3), the incremental approach of Agile Software QFD also leads to progress with regard

to A7 and A12, supporting a sustained motivation and self-organization. Commitment (A13) benefits from the first planning meeting with possibly all stakeholders and the well-founded derivation of the Sprint target on the basis of the stakeholder needs addressed in an increment. Similarly, the assessment of the higher authenticity and credibility (A8) of the procedure and the outcomes are based on the firmly established embedding of both customer and developer grooming meetings (section 3.4) into the iteration cycle.

The difference regarding the fulfillment of the criteria is overall and on average higher when comparing Agile Software QFD to focused or even classical approaches of Software QFD [cf. Schockert for more details]. Overall, the comparison shows that Agile Software QFD doesn't deny or has lost its QFD identity. Agile software QFD is a contribution to the dynamic software QFD models. It represents an evolutionary further development with focus on the smooth and better integration in the short agile development cycles taking into account the agile principles and current practices.

4.2 Comparison with agile Requirements Engineering practice

Table 4 gives an overview regarding the main advantages and disadvantages comparing Agile Software QFD with common agile Requirements Engineering practice. As there is no ONE well-defined "Agile RE", the comparison is based on widespread agile practices with greater relevance for the handling of requirements (cf. [24] in detail and section 2.3 for an overview).

Table 4: Comparison of Agile Software QFD with Agile RE practice (according to [24])

Agile Software QFD	Agile Software QFD vs. Agile RE practice	Agile RE practice
+ / ++	A1: Focus on the business value of the requirements	O
++	A6: Support cooperation of customers and developers	O / +
O / +	A7: Provide sustained motivating procedure of work	+
+ / ++	A8: Deliver authentic and credible procedure and results	O / +
+ / ++	A10: Consider quality requirements and design constraints	O / +
O / +	A12: Facilitate self-organization in teams	+ / ++
++	A14: Well-defined and clear ranking of the backlog according to business value	O
+	A17: Show dependencies of backlog items	O
++	A20: Consider problem space (needs) as well as solution space (requirements)	O
++	A21: Link problem space (needs) and solution space (requirements) explicitly	O
O	A26: Transforming the requirements of the backlog into sprint tasks	O / +
O	A27: Monitoring the implementation of sprint tasks	+

Obviously, the consequent and ongoing focus on the product requirements with highest business value to be implemented based on the satisfaction of the most important stakeholder needs (A1, A14) is an outstanding advantage of Agile Software QFD. Agile RE practice claims the same but doesn't act consequently in this matter due to only looking at the 1:1 relationships of needs and solutions in user stories. The systematic search for alternative and better solutions in Agile Software QFD by first splitting the user stories and then linking needs and solutions again support strongly the consideration of problem and solution space (A20, A21; agile practice improves in A20 when applying Story Mapping [22]). Instead of (almost) "encapsulating" the developers from the customers by a sole contact person like the product owner in Scrum, Agile Software QFD emphasizes close cooperation of all stakeholders, especially customers and users (A6). Altogether the transparent and comprehensible procedure of Agile Software QFD leads to more authentic and credible results (A8). But also potential drawbacks have to be mentioned. They are caused by some more (possibly demotivating) rules in Agile Software QFD which lead to limitations in the support of self-unfolding and self-organization (A7, A12). Besides, the derivation of concrete sprint tasks and their monitoring during an iteration are better supported by the more implementation-related agile practices (A26, A27).

Summing up, Agile Software QFD fills a gap in the agile world. It is the expression of a truly business value oriented, agile requirements engineering embedding QFD in an iterative and incremental development process. It is not about merely administration and documentation of requirements. Nor is it about identifying and implementing some user stories. It is about true design-engineering of requirements and systematic discovering of new and alternative solutions to problems in the business world.

5. CONCLUSIONS

This paper introduced Agile Software QFD as the adaptation of Software QFD with regard to its application within agile software development. The comprehensive and detailed analysis, derivation, description and evaluation of Agile Software QFD can be found in [24]. This paper focused on an overview especially of the methodological features of Agile Software QFD.

The proposal for Agile Software QFD is based on 27 design requirements on agile Requirements Engineering. They are derived from the principles and values of agile software development, the handling of requirements in agile development models and empirical sources of agile Requirements Engineering. These design requirements also form the basis for the evaluation of the proposed techniques of Agile Software QFD. The proposal for Agile Software QFD is characterized by its embedding in the agile iteration cycle and distinct methodological features such as the incrementally growing prioritization matrix and the priority map.

Evaluated against the derived criteria and compared to the most widespread techniques of agile Requirements Engineering, Agile Software QFD can provide additional value to agile development: focus on the most important stakeholder needs; search for alternative and better solutions; and close cooperation with customers/users. Agile Software QFD thus embodies the design-engineering part of Requirements Engineering in an agile software development. It is the expression of a truly business value oriented, agile requirements engineering. Similarly, Software QFD benefits from the methodological innovations as well as from the embedding in an iterative development process, which itself becomes more and more the standard procedure for developing products, not only software.

In future, further empirical validation of Agile Software QFD is needed. The proposal regards itself not as the end of an evolution. It is rather a new start which in future should lead to further enhancements by e.g. innovation methods like Design Thinking [23] (e.g. to incorporate feedback in Grooming Meeting I) or more mathematical foundation of the prioritization and correlation values together with more extensive control of the development activities on the basis of precise calculations [9]. Agile work is often combined with strong visualization activities in meetings and daily work but even more comprehensive IT support could be a future option, especially in distributed environments. One objective should be to identify simple, manageable instructions as “generative rules” to guide agile teams. These rules should not have the character of a “cookbook” or step-by-step instructions, but should rather provide a framework with enough freedom for flexibility and creativity in their application. Just in the sense of self-responsible, self-organizing, and agile acting and also the maxim of the founder of QFD, Yoji Akao, “copy the spirit, not the form.”

REFERENCES

[1] Akao, Y. (1990). Quality Function Deployment: Integrating Customer Requirements into Product Design. Translated by Glenn H. Mazur and Japan Business Consultants, Cambridge, Massachusetts, USA, 1990.

[2] Agile Alliance (2016a). Agile 101. URL: https://www.agilealliance.org/agile101/, access on 8.11.16.

[3] Agile Alliance (2016b). Agile Glossary. URL: https://www.agilealliance.org/agile101/ agile-glossary/, access on 8.11.16.

[4] Agile Alliance (2016c). Subway Map to Agile Practices. URL: https://www.agilealliance.org/agile101/subway-map-to-agile-practices/, access on am 8.11.16.

[5] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). Manifesto for Agile Software Development. URL: http://agilemanifesto.org and http://agilemanifesto.org/principles.html, access: 15.07.16.

[6] Cohen, L. (1995). Quality Function Deployment. How to make QFD work for you. Addison-Wesley. Reading, MA et.al., 1995.

[7] Cohn, M. (2004). User Stories Applied for agile Software Development. Boston u. a. 2004.

[8] Davies, R., Sedley, L. (2009). Agile Coaching. O'Reilly UK Ltd. 2009.

[9] Fehlmann, T. M. (2017). Managing Complexity: Uncover the Mysteries with Six Sigma Transfer Functions. Berlin 2017.

[10] Herzwurm, G., Schockert, S., Dowie, U. (2000). Continuous QFD - Employing QFD in case of fuzzy development tasks. In: Proceedings of 6th International Symposium on QFD & 12th North American Symposium on QFD, Novi, Michigan, USA, 2000, pp. 84-103.

[11] Herzwurm, G., Schockert, S., Mellis, W. (2000). Joint Requirements Engineering – QFD for Rapid Customer-Focused Software and Internet-Development. Braunschweig and Wiesbaden 2000.

[12] Herzwurm, G., Schockert, S. (2014). QFD for Cloud Computing, in: Proceedings of the 26th North American Symposium on Quality Function Deployment. Charleston, USA, pp. 54-64.

[13] Herzwurm, G., Schockert, S., Tauterat, T. (2015). Quality function deployment in software development - state-of-the-art. In: Proceedings of the 21st International Symposium on QFD, Hangzhou, China, 2015.

[14] Herzwurm, G., Schockert, S., Tauterat, T. (2016). A Critical Analysis of Software QFD Publications. In: Proceedings of the 22st International Symposium on QFD, Boise, USA, 2016, pp. 171-200.

[15] Highsmith, J. (2002). Agile Software Development Ecosystems. Boston 2002.

[16] Highsmith, J., Cockburn, A. (2001). Agile software development: The business of innovation. In: Computer, 34, 2001, 9, pp. 120-122.

[17] ISO/IEC/IEEE 26515 (2012). Systems and software engineering - Developing user documentation in an agile environment, ISO/IEC/IEEE 26515:2012. Genf, New York, 2012.

[18] ISO 16355-1 (2015). Application of statistical and related methods to new technology and product development process – Part 1: General Principles and Perspectives of Quality Function Deployment (QFD). ISO 16355-1:2015, Genf 2015.

[19] Komus, A., Kuberg, M. (2015). Status Quo Agile Studie zu Verbreitung und Nutzen agiler Methoden. URL: https://www.gpm-ipma.de/fileadmin/user_upload/Know-How/studien/Studie_Agiles-PM_web.pdf, access on 10.09.16.

[20] Mazur, G. (2012). Blitz QFD – The Lean Approach to Product Development. In: Proceedings of ASQ'S World Conference on Quality and Improvement. Anaheim, CA, USA, 2012, pp. 1-16.

[21] Patton, J. (2008). The New User Story Backlog is a Map. URL: http://jpattonassociates.com/the-new-backlog/, access on 20.11.16.

[22] Patton, J. (2014). User Story Mapping. Beijing u. a. 2014.

[23] Plattner, H., Meinel, C., Leifer, L. (2011). Design Thinking – Understand, Improve, Apply. Heidelberg 2011.

[24] Schockert, S. (2017). Agiles Software Quality Function Deployment. Lohmar – Köln, Eul Verlag 2017. (to be published)

[25] Schwaber, K. (1996). Controlled Chaos – Living on the Edge. In: American Programmer, 10, 1996, 4, pp. 11-16.

[26] Schwaber, K., Beedle, M. (2002). Agile Software Development with Scrum. Upper Saddle River 2002.

[27] Schwaber, K., Sutherland, J. (2016). The Definitive Guide to Scrum: The Rules of the Game. URL: http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf, access on 4.11.16.

[28] ScrumAlliance (2015). The 2015 State of Scrum Report. URL: http://www.leroyward.com/wp-content/uploads/2014/08/State-of-Scrum-2015.pdf, access on 7.11.16.

[29] Shindo, H. (1999). Application of QFD to Software and QFD Software Tools. In: Pre-Conference Workshops of the Fifth International Symposium on Quality Function Deployment and the First Brazilian Conference on Management of Product Development. Belo Horizonte, Brazil, 1999.

[30] VersionOne (2013). 7th Annual State of Agile Development Survey. URL: https://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf, access on 6.11.16.

[31] VersionOne (2016). 10th annual State of Agile Report. URL: http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf, access on 6.11.16.

[32] Watanabe, Y., Kawakami, Y., Iizawa, N. (2012). Software requirements analysis method using QFD. In: Proceedings of the 18th international symposium of quality function deployment. Tokyo, Japan, 2012.

[33] Watanabe, Y., Yoshikawa, M., Shindo, H. (2013). Software development method based on twin peaks model with QFD. In: Proceedings of International Symposium on QFD & North American Symposium on QFD. Santa Fe, USA, 2013, pp. 117-126.

[34] Williams, L. (2012). What Agile Teams Think of Agile Principles. In: Communications of the ACM, 55, 2012, 4, pp. 71-76.

[35] Zultner, R. E. (1994). Software quality function deployment - the North American experience. In: SAQ, EOQ-SC (Eds.), Software Quality Concern for People. Proceedings of the Fourth European Conference on Software Quality. Basel, Switzerland, Zürich 1994 pp. 143-158.

BIOGRAPHIES OF THE AUTHORS

Dipl.-Wirt.-Inf. Sixten Schockert

Dipl. Wirt.-Inf. Sixten Schockert holds a Master's degree in Information Systems from the University of Cologne, Germany. Since 2003 researcher and lecturer at the department of Information Systems, esp. Business Software of Prof. Dr. Georg Herzwurm at the University of Stuttgart; founding member and since 2006 member of the board of the QFD Institute Deutschland (QFD-ID); 2013 he received the international Akao-Prize for outstanding contributions to the further development and support of QFD. Certified QFD-Architect conferred by the QFD-Institute Germany. He passed his doctoral examination in the end of 2016, his PhD thesis deals with the development of Agile Software Quality Function Deployment and will be published in October 2017.

Prof. Dr. Georg Herzwurm

Prof. Dr. Georg Herzwurm is a Distinguished Full Professor and holds the Chair for Business Administration and Information Systems, esp. Business Software at the University of Stuttgart, Germany. Since 2013 he is board member at the Graduate School of Excellence advanced Manufacturing Engineering (GSaME) and director of the cluster B "Management of Global Manufacturing Networks". He is founder and speaker of the board of the German QFD Institute (QFD-ID). Since 2009 he is director of the International Council for QFD (ICQFD). 2000 he received the international Akao-Prize for outstanding contributions to the further development and support of QFD. Certified QFD-Architect conferred by the QFD-Institute Germany.