Martin Engstler et. al. (Hrsg.): Projektmanagement und Vorgehensmodelle 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015 151

# Das Business setzt die Prioritäten?!

Sixten Schockert[1], Georg Herzwurm[2]

**Abstract:** Um zu entscheiden, welche Anforderungen im nächsten Entwicklungszyklus einer agilen Softwareentwicklung berücksichtigt werden, ist deren Priorisierung unabdingbar. Diese Priorisierung sollte gemäß dem agilen Paradigma aus Sicht des Business erfolgen. Doch allzu oft geschieht eben diese, für das weitere Vorgehen entscheidende Bewertung wenig systematisch, adhoc und vor allem ohne dezidierte Berücksichtigung der Nutzeffekte für den Kunden. Dieser Beitrag will einen Weg aufzeigen, dies zu ändern.

**Keywords:** Priorisierung von Anforderungen, Agile Softwareentwicklung, Requirements Engineering

## 1 Motivation

Innerhalb der agilen Softwareentwicklung wird in jedem Entwicklungszyklus eine abgestimmte Menge von Anforderungen in ein auslieferbares Produktinkrement umgewandelt. Um diese Anforderungen festzulegen, ist im Allgemeinen deren Auswahl aus einer größeren Menge an möglichen Anforderungen zwingend notwendig. Inhärenter Bestandteil der agilen Softwareentwicklung in Inkrementen ist demzufolge die Bewertung und Priorisierung von Anforderungen. Auf dieser Basis werden dann die wichtigsten dieser Anforderungen als konkrete, handlungsleitende Vorgaben für die Entwicklung ausgewählt. Um Unsicherheiten in Anforderungen zu reduzieren bzw. um keine unsicheren Anforderungen umzusetzen, müssen wir entscheiden, welche Anforderungen im nächsten Inkrement umgesetzt werden und welche nicht. Aber nicht nur das: von dieser Priorisierung von Anforderungen hängen essentiell die zukünftigen Entwicklungsaktivitäten ab. Fehlentscheidungen an dieser Stelle führen zu Fehlentwicklungen, die oft nur schwer zu korrigieren sind.

Dieser wichtige Schritt bei jeder agilen Entwicklung in Inkrementen – in Scrum z. B. die Priorisierung der Anforderungen aus dem Product Backlog für den nächsten Sprint [Pi08] – wird allerdings allzu oft wenig systematisch und wenig strukturiert durchgeführt. Sicherlich, durch die vermehrte Kommunikation und regelmäßige Diskussion von Anforderungen – z. B. in den Sprint-Planungs- und Reviewmeetings in Scrum – werden diese von allen Stakeholdern von Kunden- und Entwicklungsseite besser verstanden.

[1] Universität Stuttgart, BWI, Abt. VIII, LS für ABWL und WI II, Keplerstraße 17, 70174 Stuttgart und Fachhochschule für Ökonomie und Management (FOM), Rotebühlstraße 121, 70178 Stuttgart, schockert@wius.bwi.uni-stuttgart.de
[2] Universität Stuttgart, BWI, Abt. VIII, LS für ABWL und WI II, Keplerstraße 17, 70174 Stuttgart, herzwurm@wius.bwi.uni-stuttgart.de

Doch die Priorisierung ist schon aufgrund der unterschiedlichen Perspektiven der Beteiligten aus Technik und Business kein einfaches Unterfangen. Das agile Paradigma setzt dabei die höchste Priorität auf die Zufriedenheit der Kunden durch die frühe und kontinuierliche Auslieferung von für die Kunden „wertvoller" Software [Be01]. Doch gerade dieser „business value" d. h. die konkreten Nutzeffekte für den Kunden werden oft höchstens adhoc und wenig nachvollziehbar in die Priorisierung einbezogen. In vielen Fällen ist es dann doch die „große Erfahrung" einzelner Personen, die den Bewertungsprozess bestimmen. Dieser Beitrag will einen Weg aufzeigen, gerade die Bewertung aus Kundensicht systematisch und nachvollziehbar in den agilen Entwicklungsprozess einzubeziehen.

## 2 Priorisierung bei agiler Entwicklung

Das agile Paradigma fokussiert auf die Auslieferung der wichtigsten Geschäftsanforderungen innerhalb kurzer Zeiträume von zwei Wochen bis zu einem Monat. Schnell und in regelmäßigen Abschnitten soll tatsächlich lauffähige und für die Kunden produktiv einsetzbare Software verfügbar sein. Aus diesen Gründen gibt es auch die ideale Zweiteilung im Sinne der Konzentration auf Kernkompetenzen:

- „Das Business setzt die Prioritäten" [Co15]: Auftraggeber, Fachabteilung, Produktmanager o.ä. orientieren sich am maximalen business value für die Kunden und haben damit die finale Entscheidung über die im nächsten Entwicklungszyklus zu berücksichtigenden Anforderungen.
- Die Entwicklung entscheidet über die Art und Weise der Umsetzung der Anforderungen: Selbstorganisierende Entwicklungsteams legen das beste Vorgehen zur Auslieferung der höchstpriorisierten Anforderungen fest [Co15].

Diese Zweiteilung ist nicht gleichbedeutend damit, dass diese Aktivitäten strikt getrennt werden sollten. Es werden für jeden Zyklus konkrete Entwicklungsvorgaben benötigt und diese sollten in gemeinsamen Sitzungen der Business- und Entwicklungsseite abgeleitet und abgestimmt werden. In der Terminologie des Requirements Engineering werden diese Entwicklungsvorgaben auch als Produktanforderungen (engl. product requirements) bezeichnet [Eb14]. Produktanforderungen repräsentieren konkrete Eigenschaften und Fähigkeiten der Softwarelösung, die z. B. für den Mitarbeiter auf der Fachseite oder den Endbenutzer deren Probleme löst bzw. einfach Nutzen stiftet [ISO08].

**Doch wie kann das Business z. B. der Projektleiter auf der Fachseite die Priorisierung von solchen Lösungseigenschaften und –fähigkeiten leisten?** Oder konkret für Scrum formuliert: wie kann der Product Owner als Verantwortlicher für den Projekterfolg [Co15, Pi08] die Priorisierung von Lösungsfeatures aus Marktsicht leisten? Die Produktanforderungen entstammen ja gerade nicht dem Problemraum des Business, sondern bereits dem Lösungsraum der Entwickler. Müssen sie ja auch, sonst würden sie nicht als Entwicklungsvorgaben taugen.

## 3 Dreiteilung aus Problemen, Lösungen und Anforderungen

Der Projektleiter auf der Fachseite oder auch der Product Owner bei Scrum müssen dabei unterstützt werden, die Auswahl der Entwicklungsvorgaben unter Berücksichtigung des business value, d. h. auf Basis der Nutzeffekte aus Kundensicht, nachvollziehbar für alle Stakeholder zu leisten. Wenn man sich in dieser Situation nicht auf den Erfahrungsschatz einzelner Personen verlassen will, ist es zuerst nötig sich bewusst zu machen, dass es eine **Dreiteilung aus Problemen, Lösungen und Anforderungen** gibt. Lauenroth hat dies folgendermaßen für das Requirements Engineering zusammengefasst:

„Das Requirements Engineering als gestaltende Disziplin ist dafür verantwortlich, dass mit allen relevanten Stakeholdern die Problemstellungen, die eine Software lösen soll, definiert und abgestimmt werden, Lösungen für die identifizierten Problemstellungen definiert und abgestimmt werden (sowie) Anforderungen an die Lösungen definiert und abgestimmt werden." [La10]

Nötig für die Priorisierung von konkreten Anforderungen aus Business-Sicht sind demzufolge vor allem das Verständnis von Problemen und möglichen Lösungen zu den Problemen. Oder anders formuliert: Produktanforderungen können nur dann aus Business-Sicht zu Entwicklungsvorgaben priorisiert werden, wenn zuvor Probleme bzw. Kundenbedürfnisse und Lösungen definiert wurden.

Aber dazu muss zuerst einmal die explizite Trennung von Kundenbedürfnissen, Lösungen und konkreten Produktanforderungen erfolgen, die beschriebene Dreiteilung muss nachvollzogen werden. Das ist sowohl im Requirements Engineering (RE) als auch bei agiler Softwareentwicklung nicht selbstverständlich. Beide Domänen lassen sich als Teilgebiete des umfassenderen Software Engineering auffassen in dem traditionell der Schwerpunkt auf den Produktanforderungen im Sinne konkreter Vorgaben für die weitere Entwicklung liegt. Die Trennung von „was" wird gefordert und „wie" wird das „was" umgesetzt ist zwar legendär [Da90] und viele Ansätze bemühen sich um eine Trennung wie z. B. Ebert mit der Unterscheidung von Markt-, Produkt- und Komponentenanforderungen [Eb14].

Doch die (sprich-)wörtliche Trennung reicht alleine nicht aus für die Priorisierung von Anforderungen aus Business-Sicht. Sie erlaubt zwar den Blick auf die Anforderungen aus unterschiedlichen Perspektiven wie der Entwickler- und der Marktsicht. Sie führt aber auch dazu, dass Entwickler und Kunden sich noch verstärkt mit ihren Beschreibungsmitteln für Anforderungen voneinander entfernen: Entwickler bevorzugen präzise, möglichst formale Modellbeschreibungen von Anforderungen als exakte Entwicklungsvorgaben, Kunden bevorzugen die natürlich-sprachliche Beschreibung von Anforderungen. Nicht umsonst sind textbasierte Anforderungsbeschreibungen in natürlicher Sprache mit und ohne Vorlagen weiterhin weit verbreitet [AWK13]. Entscheidend ist deswegen nicht die Trennung von Kundenbedürfnissen, Lösungen und konkreten Produktanforderungen, sie ist nur die notwendige Bedingung. Vor allem müssen Bedürfnisse und Lösungen für die Priorisierung von Produktanforderungen als Vorgabe für die Entwicklung wieder begründet zusammengeführt werden.

## 4 Zusammenführung von Bedürfnissen und Lösungen

Neben textbasierter Anforderungen haben sich insbesondere bei agiler Entwicklung sog. user stories als Beschreibungen für Eigenschaften und Fähigkeiten zukünftiger Softwarelösungen durchgesetzt [AWK13, Co10]. So werden user stories z. B. bei Scrum zur Formulierung der Einträge im Product Backlog eingesetzt, die der Product Owner dann für die Auswahl für den nächsten Sprint priorisiert. User stories sind dabei explizit aus der Perspektive der Benutzer bzw. Stakeholder formuliert. Und bei konsequenter Handhabung der Vorlage „als <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen/Vorteil> zu erreichen“ [Co10] lässt sich mit ihnen auch eine Beziehung zwischen Bedürfnissen (= Nutzen/Vorteil) und Lösungen (= Ziel/Wunsch) abbilden. Solche „begründeten“ user stories berücksichtigen explizit den Grund für einen Lösungswunsch in der Beschreibung einer Anforderung. Sie repräsentieren damit einen Schritt in die Richtung der Zusammenführung von Bedürfnissen und Lösungen.

Aber user stories besitzen gleichzeitig auch einige Schwächen. So sind sie oft zu unbestimmt um als Entwicklungsvorgabe im Sinne einer konkreten Produktanforderung zu dienen. Zudem bilden sie nur einfache 1:1-Beziehungen zwischen Bedürfnissen und Lösungen ab, ggf. vorhandene 1:n bzw. n:m-Beziehungen sind nur schwer bzw. kompliziert darstellbar. So werden sowohl Lösungen, die mehrere Kundenbedürfnisse befriedigen als auch mögliche alternative Lösungen für dasselbe Kundenbedürfnis nicht systematisch berücksichtigt.

Darüber hinaus geben user stories nur wenig Handlungsunterstützung bei der expliziten Priorisierung vieler Anforderungen untereinander, denn dazu müssen hier gleichzeitig viele Bedürfnisse und Lösungen miteinander verglichen werden. Die vorhandene Abgrenzung von Bedürfnissen und Lösungen wird eben gerade nicht (!) zu einer getrennten Priorisierung von Bedürfnissen und Lösungen genutzt. Eine getrennte Bewertung würde den Fokus in Richtung des Business verschieben, denn dann können die Lösungen auf Basis der wichtigsten Nutzeffekte für den Kunden ausgewählt werden. Nur dann ist die für die Formulierung von konkreten Produktanforderungen als Entwicklungsvorgaben entscheidende Zusammenführung von priorisierten Kundenbedürfnissen mit den besten Lösungen zur Erfüllung dieser Bedürfnisse nachvollziehbar leistbar.

Ansätze basierend auf der Produktplanungsmethode Quality Function Deployment (QFD) [HSM00] können hier einen Weg weisen, denn QFD ist im Kern eine Priorisierungsmethode von Lösungen auf Basis von Kundenbedürfnissen und passt demzufolge exakt auf die beschriebene Problemstellung. QFD trennt konsequent zwischen Bedürfnissen und Lösungen, priorisiert die Bedürfnisse separiert, beurteilt die Lösungen im Hinblick auf ihre Nutzeffekte für die Kunden, ermittelt auf diese Weise die wichtigsten Lösungen aus Kundensicht, die dann weiter bewertet werden und zu konkreten Entwicklungsvorgaben verdichtet werden. Das alles geschieht mit dem einfachen Mittel einer Matrixdarstellung (Abb. 1), die im Gegensatz zu User Stories n:m-Beziehungen übersichtlich darstellt und eben auch die für die Priorisierung aus Business-Sicht notwendige Zusammenführung von Bedürfnissen und Lösungen zu Produktanforderungen leistet.

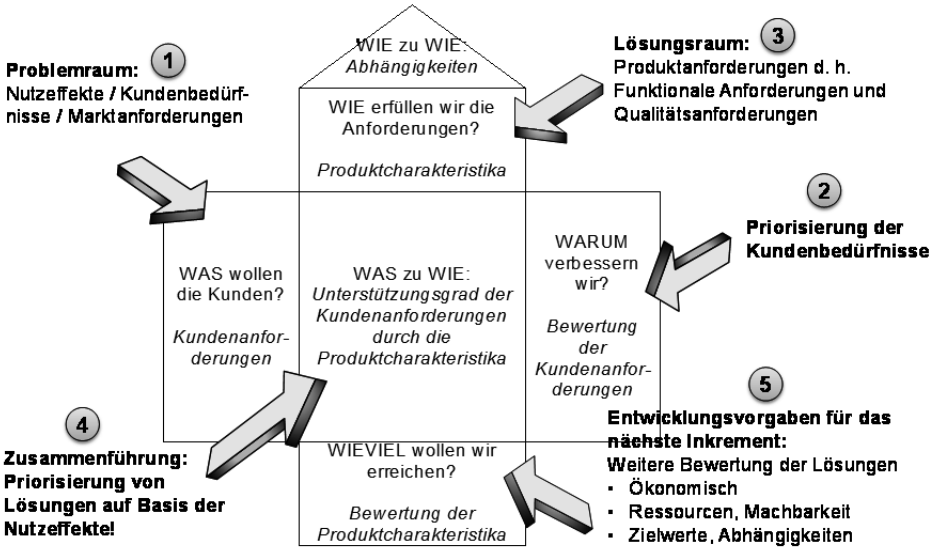


Abb. 1: Schematische Matrixdarstellung in QFD

Allerdings ist QFD im Sinne der Kundenorientierung zwar als effektiv aber vor allem auch als sehr komplex bekannt [z. B. CE94, HSM00]. Gerade die Zeit und der Aufwand zur Erstellung der Priorisierungsmatrix können bei vielen Bedürfnissen und Lösungen unverhältnismäßig im Verhältnis zum Nutzen erscheinen. Neuere methodische Ansätze in der Durchführung von QFD sind allerdings vielversprechend und fordern eine Integration mit dem Vorgehen der agilen Softwareentwicklung.

So beruht das sog. Continuous QFD (CQFD) auf inkrementellen Planungs- und Implementierungszyklen, die durch den Einsatz von ergänzenden Methoden wie dem design thinking [z. B. PML11] und Informationstechnologie zum Prototyping die möglichen Produkteigenschaften visualisieren und den Kunden damit helfen, mögliche Umsetzungen ihrer Anforderungen zu ersten Lösungen besser zu verstehen [HS14]. In CQFD werden die Priorisierungsmatrizen inkrementell bei jeder Iteration d. h. für jeden Sprint weiterentwickelt. Dabei werden in die Planungs- und Reviewmeetings die QFD-Aktivitäten der Ideensammlung/Brainstorming, des Verstehens und Sortieren/Klassifizieren von Artefakten sowie des Bewertens/Prüfens und schließlich des Entscheidens integriert. Die Priorisierung erfolgt dabei zweigeteilt: auf der einen Seite die Bedürfnisse z. B. mittels des Analytic Hierarchy Process [Sa99] aber auch einfacherer Priorisierungstechniken [z. B. HSM00, Po10]; auf der anderen Seite die Lösungen durch die Beurteilung ihrer Wirkung auf die Bedürfniserfüllung in der Priorisierungsmatrix.

Die Matrix d. h. die Zusammenführung aus Bedürfnissen und Lösungen ist dabei zu Anfang eher klein: ausgehend von den wichtigsten Bedürfnissen aus Kundensicht werden zuerst nur die Lösungen identifiziert und zu Entwicklungsvorgaben konkretisiert, die diese am besten abdecken. Mit jedem Entwicklungszyklus vergrößert sich die Matrix, es

kommen sowohl neue Bedürfnisse als auch neue Lösungen hinzu, die in die bestehende Matrixdarstellung integriert werden. Die Priorisierung der Bedürfnisse und Lösungen zu Entwicklungsvorgaben erfolgt transparent und dynamisch mit der inkrementell wachsenden Priorisierungsmatrix. Diese kann dabei auch als Kontrollinstrument in den Reviewmeetings eingesetzt werden um z. B. festzustellen, ob eine umgesetzte Lösung den gewünschten Effekt auf den Kundennutzen hatte oder nicht. Die Projektverantwortlichen wie z. B. der Product Owner bei Scrum erhalten auf diese Weise ein einfaches, für alle nachvollziehbares Planungs- und Steuerungsinstrument aus Business-Sicht in die Hand.

In CQFD aber auch in anderen QFD-Ansätzen wie in Watanabes Twin Peaks Model [WYS13] fließen überdies auch Architekturüberlegungen und mögliche Technologiekonzepte (d. h. Betriebssysteme, Datenbanken, Anwendungsserver, Softwareplattformen, wiederverwendbare Webkomponenten etc.) in die Priorisierung ein. Damit ist auch sichergestellt, dass die Bewertung nicht nur die Kundensicht sondern eben auch Realisierbarkeit, Risiko und grundsätzliche Wirtschaftlichkeitsüberlegungen berücksichtigt. QFD und insb. die Priorisierungsmatrix leisten dabei die rückverfolgbare Verbindung zwischen Problem- und Lösungsraum. Neben der iterativen Erstellung einer immer weiter wachsenden Matrix gibt es zudem die Möglichkeit gerade zu Entwicklungsbeginn oder nur für bestimmte Nutzeffekte eine Variante eines Ursache-Wirkungsdiagramms, die sog. Maximum Value Table (Abb. 2), einzusetzen [Ma12]. Dabei können allerdings n:m-Beziehungen zwischen Bedürfnissen und Lösungen nur eingeschränkt berücksichtigt werden, denn die Übersichtlichkeit würde bei zu vielen Pfeilen leiden. Trotzdem kann die Maximum Value Table als eine Art Mittelweg zwischen der einfachen user story und der umfassenderen Matrix fungieren.

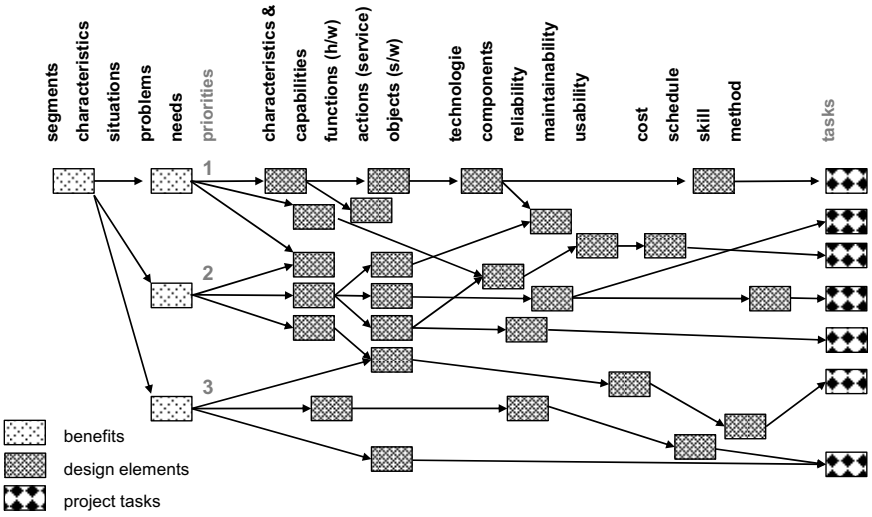


Abb. 2: Konzept der Maximum Value Table [Ma12, GDM14]

## 5 Fazit

Ziel muss es sein die Dreiteilung aus Problemen, Lösungen und Anforderungen einfach und nachvollziehbar bei allen Stakeholdern zu verankern. Dazu gehört die Trennung dieser Artefakte, aber noch viel mehr ist ihre begründete Zusammenführung für die wirkliche Bewertung aus Business-Sicht entscheidend. Nur dadurch kann die Priorisierung von Lösungen nachvollziehbar auf Basis der Bedürfnisse/Nutzeffekte erfolgen, um die wichtigsten Lösungen zu Produktanforderungen im Sinne von Entwicklungsvorgaben für das nächste Inkrement zu konkretisieren. Überdies muss dies systematisch und transparent geschehen, denn die korrekte Bewertung darf nicht von einzelnen Personen und deren Erfahrung abhängen. Die verbreitet in agiler Softwareentwicklung genutzten user stories gehen hier allerdings nicht weit genug. In Quality Function Deployment hingegen ist sowohl die Trennung als auch die Zusammenführung von Bedürfnissen und Lösungen inhärent vorhanden. Insbesondere mit Continuous QFD, der inkrementellen Matrixerstellung und der Maximum Value Table kann QFD als Priorisierungsmethode einen Weg zur Bewertung von Anforderungen aus Business-Sicht und ihrer Auswahl für den nächsten Entwicklungszyklus ebnen.

## Literaturverzeichnis

[AWK13] Adam, S.; Wünsch, C.; Koch, M.: Ergebnisbericht RE-Kompass 2013. Fraunhofer-Institut IESE. www.iese.fraunhofer.de/content/dam/iese/de/dokumente/oeffentliche_studien/ Ergebnisbericht_RE-Kompass_2013.pdf, Stand: 8.9.2015.

[Be01] Beck, K. et.al.: Prinzipien hinter dem Agilen Manifest. www.agilemanifesto.org/iso/de/principles.html, Stand: 7.9.2015.

[CE94] Curtius, B.; Ertürk, Ü.: QFD-Einsatz in Deutschland. In: QZ – Qualität und Zuverlässigkeit, Nr. 4, 1994, S. 394-402.

[Co10] Cohn, M.: User Stories: für die agile Software-Entwicklung mit Scrum, XP u.a. mitp, Bonn 2010.

[Co15] Cohn, M.: Einführung in Scrum. www.mountaingoatsoftware.com/agile/scrum/a-reusable-scrum-presentation, Stand: 7.9.2015.

[Da90] Davis, A.M.: Software Requirements. Analysis & Specification. Englewood Cliffs, New Jersey 1990.

[Eb14] Ebert, C.: Systematisches Requirements Engineering. Anforderungen ermitteln, dokumentieren, analysieren und verwalten. dpunkt, Heidelberg 2014.

[GDM14] Grimm, J.; Denavs, D.; Mazur, G.: Using QFD to Design a Multi-Disciplinary Clinic. In: Proceedings of North American Symposium on QFD, San Diego, USA, 2011.

[HS14] Herzwurm, G.; Schockert, S.: QFD for Cloud Computing. In: Transactions of the 26th North American Symposium on Quality Function Deployment. Charleston, 2014, S. 54-64.

[HSM00] Herzwurm, G.; Schockert, S.; Mellis, W.: Joint Requirements Engineering. QFD for Rapid Customer-Focused Software and Internet Development, Vieweg – Gabler, Braunschweig – Wiesbaden 2000.

[ISO08] ISO/IEC 12207:2008: Systems and software engineering -- Software life cycle processes. 2008.

[La15] Lauenroth, K.: Die Legende von der lösungsneutralen Anforderung oder warum das Requirements Engineering eine Gestaltungsdisziplin ist! In: Object Spectrum – Online Themenspecial Requirements Engineering 2015.

[Ma12] Mazur, G.: Blitz QFD – The Lean Approach to Product Development. In: Proceedings of the World Conference on Quality and Improvement. Milwaukee WI, ASQ, Mai 2012.

[Pi08] Pichler, R.: Scrum: Agiles Projektmanagement erfolgreich einsetzen. dpunkt, Heidelberg 2008.

[Po10] Pohl, K.: Requirements Engineering. Springer, Berlin 2010.

[PML11] Plattner, H.; Meinel, C.; Leifer, L.: Design Thinking – Understand, Improve, Apply, Springer, Heidelberg 2011.

[Sa99] Saaty, T. L.: Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World. 3. Aufl., Pittsburgh 1999.

[WYS13] Watanabe, Y.; Yoshikawa, M.; Shindo, H.: Software development method based on twin peaks model with QFD. In: Proceedings of the 19th International Symposium on QFD & 22nd North American Symposium on QFD 2013, Santa Fe, USA, S. 117-126.