Markup-basiertes Spezifikations- und Anforderungsmanagement in agilen Softwareprojekten

Roman Roelofsen[1], Stephan Wilczek[2]

[1]Rodalo GmbH
Kaiserstraße 108
53721 Siegburg
rr@rodalo.com

[2]Hochschule der Medien
Nobelstr. 10
70569 Stuttgart
wilczek@hdm-stuttgart.de

Abstract: Agile Softwareprojekte benötigen ein spezielles Management von Anforderungen und Spezifikation. Der Artikel versucht, die notwendige Sicht darauf zu beschreiben, daraus Anforderungen an eine Werkzeugunterstützung abzuleiten und stellt mit *agileSpecs* eine experimentelle Plattform vor, die mit Hilfe einer Markup-basierten Sprache versucht, einem agilen Anforderungs- und Spezifikationsmanagement gerecht zu werden.

1 Spezifikation als kritischer Erfolgsfaktor

Im Hinblick auf das Scheitern von IT-Projekten verweisen Untersuchungen insbesondere auf unvollständige oder sich im Projektablauf ändernde Anforderungen und Spezifikationen sowie unzureichendes Input und fehlende Kommunikation mit den verschiedenen Stakeholdern im Projekt [MM11]. Schwaber bringt es mit dem Aphorismus „garbage in, garbage out“ auf den Punkt und betrachtet kritisch die Definition von Anforderungen, den Change Management Prozess von Anforderungen und Werkzeuge sowie Techniken für das Management von Anforderungen [SL06].

Agile Methoden und “Continuous Delivery” haben die Entwicklung, den Test und die Auslieferung von Software verbessert. Dennoch wird gerade das Management von Anforderungen und Spezifikationen weiterhin kritisch gesehen [BM14]. Ansätze wie die Erfassung und Verwaltung von User Stories, Use Cases bzw. Softwaretests oder die Verlagerung in das UI Prototyping bzw. das Rapid Application Development (RAD) oder das Joint Application Development (JAD) bieten Alternativen in agilen Projekten.

2 Spezifikation in agilen Softwareprojekten

Agile Softwareentwicklung zeichnet sich unter anderem durch die Fokussierung auf die beteiligten Individuen, Zusammenarbeit mit dem Kunden und die Fähigkeit aus, schnell und flexibel auf Veränderungen zu reagieren. Der funktionierenden Software wird dabei ein höherer Stellenwert beigemessen als einer umfangreichen Dokumentation, die aber dennoch als wichtig[12] erachtet wird [Be01]. Die Erhebung und Verwaltung von Softwareanforderungen in agilen Projekten ist geprägt durch eine Vielzahl unterschiedlicher Techniken und Werkzeuge. Wird die initiale Definition der formalen und nicht-formalen Anforderungen an die Software zum Projektbeginn noch in einem Dokument spezifiziert und bspw. als Grundlage für Verhandlungen mit Dienstleistern genutzt, so werden die unterschiedlichen Anforderungen im Projektverlauf verfeinert, in unterschiedliche Werkzeuge übertragen und in der Regel an mehreren Stellen verwaltet.

2.1 Arbeit in Arbeitspaketen

Agile Vorgehensmodelle stellen den Entwickler in den Mittelpunkt. Dabei wird der Prozess konsequenterweise auf die üblichen technischen Tätigkeiten des Entwicklers ausgerichtet. So werden die Anforderungen mit Hilfe von Techniken wie bspw. User Stories, Use Cases oder UI Prototyping erhoben und anschließend in für die Entwickler geeignete Arbeitspakete zerlegt, die in iterativen Schritten bearbeitet werden. Die einzelnen Arbeitspakete lassen sich in der Regel einer bestimmten Kategorie zuordnen, wobei die Kategorien von der Umsetzung einer User Story über einen Change Request bis hin zu einem Softwaretest oder der Behebung eines Softwarefehlers reichen. Die einzelnen Schritte und Phasen des Vorgehensmodells unterscheiden allerdings nicht zwischen den unterschiedlichen Kategorien. Grund dafür ist, dass innerhalb einer Iteration die stetige Weiterentwicklung bzw. Verbesserung der Software im Vordergrund steht. Ob es sich dabei um die Behebung eines Fehlers (Bug Report) oder um die Implementierung einer neuen Anforderung handelt, spielt aus Sicht des Entwicklers nur eine untergeordnete Rolle – eine funktionierende Software gilt als das messbare Ergebnis jeglicher Veränderungen innerhalb einer Iteration. Der Soll-Zustand der Software ergibt sich daher implizit aus der iterativen Verwertung (Beurteilung und Umsetzung) der Arbeitspakete, die während des Entwicklungszeitraums umgesetzt werden. Dabei wirken sich die Arbeitspakete unterschiedlich stark auf den Soll-Zustand aus. So erweitern beispielsweise User Stories den Soll-Zustand, wohingegen Bug Reports den Ist-Zustand an den Soll-Zustand angleichen sollen. Im letzten Fall könnte sowohl eine falsch *formulierte* User Story als auch eine falsch *umgesetzte* User Story der Grund sein.

2.2 Werkzeugnutzung und Medienbrüche

User Stories müssen strukturiert und mit diversen Meta-Daten erfasst werden: Datum, Priorität, Zustand („Entwurf“, „Abgenommen“, ...). Dies widerspricht jedoch schnell dem Anspruch, die Spezifikation mit minimalen Aufwand pflegen zu können und sie pri-

[12] “That is, while there is value in the items on the right, we value the items on the left more.” [Be01] - wobei die „laufende Software“ links und eine „umfassende Dokumentation“ rechts stehen.

mär als textuelles Dokument zu behandeln, in dem User Stories und fachliches Wissen gesammelt werden. Ein Entwickler würde ab einem gewissen Zustand im Prozess eine User Story in ein Ticket-System überführen, um beispielsweise Diskussionen und Verantwortlichkeiten festzuhalten. Bug Reports werden von vornherein in einem Ticketsystem erfasst. Ein Bug Report kann, wie oben beschrieben, Auswirkungen nur auf die Software oder auf die Software und die Spezifikation haben.

Bei der Verwaltung von Softwareanforderungen und Spezifikationen finden sich in der Praxis so Text- oder Tabellenkalkulations-Dokumente, die mit verschiedenen Methoden versioniert werden. Gerade in kleineren Projekten ist es in der Praxis üblich, dass der Kunde User Stories und Meta-Daten in einer Tabellenkalkulation und den Text der User Stories sowie fachliches Wissen in Text-Dokumenten erfasst. Dazu kommen Architekturskizzen, die mit Grafik- oder Präsentationsprogrammen erstellt werden. Der Entwickler kopiert den Text der User Stories und arbeitet innerhalb des agilen Prozesses mit einem Ticketsystem. Jeder arbeitet in "seinem" System, was eine manuelle und fehleranfällige Synchronisation nötig macht.

Selbst wenn auf eine Spezifikation verzichtet wird, sind Medienbrüche und Systemgrenzen vorhanden: Ticket-Systeme sammeln User Stories und Bug-Reports, der Programmierer setzt diese in Source Code um, fachliches Wissen wird ergänzend vom Kunden in Textform geliefert und Change-Logs beinhalten Zusammenfassungen. Es ist daher besonders die Angst vor weiterer Redundanz (unterschiedliche Zustände, doppelte Arbeit) die das Erstellen einer Spezifikation erschwert. Die Arbeit mit verbreiteten Tabellenkalkulations- und Textverarbeitungsdokumenten lässt sich schwer mit den derzeit üblichen Techniken wie einem zentralen Ticket- und Source Code Management-System verbinden.

2.3 Agile Spezifikation

In einem agilen Projekt muss die Spezifikation nicht zwangsläufig jederzeit den Ziel-Zustand erfassen, sondern kann dem aktuellen Entwicklungsstand der Software entsprechen. Die Spezifikation kann in drei zeitliche Kategorien eingeordnet werden:

- Vorgelagert: Vor der Iteration werden die als nächstes zu implementierenden Anforderungen spezifiziert
- Begleitend: Die Spezifikation wird parallel zur Softwareentwicklung weiterentwickelt
- Nachgelagert: Nach der Iteration werden die durchgeführten Arbeiten konsolidiert und festgehalten

In der Praxis wird in allen drei Kategorien in unterschiedlichem Ausmaß gearbeitet. Besonders der erste Zustand hat bei der Kommunikation mit dem Kunden einen großen Wert. Ziel ist es, unnötige Entwicklungen zu vermeiden und fachliche Anforderungen zuerst schriftlich festzuhalten. Die dritte Kategorie dient zur Validierung und Abnahme der umgesetzten Anforderungen. Dies geschieht durchaus unter Anwendung der agilen Methoden, da sich auch eine Spezifikation iterativ und stetig entwickeln lässt.

Im Hinblick auf das Spezifikations- und Anforderungsmanagement in agilen Softwareprojekten liegt der Gedanke daher nahe, allen Beteiligten im Prozess die Möglichkeit zu geben, Anforderungen und Spezifikationen strukturiert aufnehmen zu können, sie jederzeit einzusehen und mit minimalem Aufwand auch zu editieren bzw. zu administrieren.

2.4 Pragmatische Betrachtung der Spezifikation

Eine Spezifikation in einem agilen Softwareprojekt sollte nicht als "umfassende Dokumentation" verstanden werden, sondern als Zustandsbeschreibung der Software, Sammlung der User Stories und Dokumentation von fachlichem Wissen. In der Praxis zeigt sich jedoch, dass die Softwareentwicklung oft nur unzureichend mit der Erstellung der Spezifikation kombiniert wird. Das mag technische (siehe der Hinweis auf die Medienbrüche oben), aber auch soziale Gründe haben. Insbesondere sehen viele Entwickler keinen direkt Mehrwert in einer ausführlichen Dokumentation und empfinden ihre Arbeit nach der erfolgreichen (softwaretechnischen) Implementierung als erledigt.

Das Problem lässt sich dadurch mindern, dass die Spezifikation eben nicht als "umfassende Dokumentation" betrachtet wird, sondern, wie oben beschrieben, primär als Sammlung der User Stories und als nützliches Hilfsmittel bei der Kommunikation mit dem Auftraggeber und anderen Stakeholdern. Zusätzlich sollte die Spezifikation ein integraler Bestandteil im Arbeitsprozess des Entwicklers sein. Ein „copy & paste“ zwischen den Systemen sollte auf jeden Fall verhindert werden und die Spezifikation selbst zum aktiv genutzten Werkzeug bzw. Material des Entwicklers werden.

2.5 Ableitung eines aktuellen „Status Quo“

Unabhängig vom Entwicklungsprozess benötigen Softwareprojekte eine geeignete Spezifikation des Zielsystems bzw. eine Möglichkeit, den „Status Quo“ abzufragen. Mag sich die Spezifikation über die Laufzeit des agilen Projekts zwar stetig weiterentwickeln und vervollständigen, benötigen die Beteiligten in den meisten Fällen dennoch ein aktuelles Gesamtbild der bereits implementierten Software und des angestrebten Zielsystems. Mit steigender Anzahl der Iterationen werden mehr und mehr Arbeitspakete umgesetzt. Aber auch Modifikationen, ein Hinzufügen oder ein ersatzloses Streichen von Arbeitspaketen sind im agilen Prozess jederzeit möglich. Die aktuelle Gesamtspezifikation des Zielsystems ergibt sich aus einer Analyse aller offenen und geschlossenen Arbeitspakete in ihrer aktuellen, seit dem Projektstart aber möglicherweise mehrfach modifizierten Form. Je nach Prozess muss hier auf eine Reihe unterschiedlicher Quellen zugegriffen werden. Da die Arbeitspakete, wie oben geschildert, in unterschiedlichen Systemen (Ticketsystem, Bug-Tracking-System, Test-Werkzeug etc.) verwaltet werden können, ist die Herleitung einer aktuellen sowie strukturierten Gesamtspezifikation eine komplexe und schwierige Aufgabe.

3 Anforderungen an eine Spezifikationsplattform

Eine Plattform zur Erstellung der Spezifikation muss sich in eine Umgebung bestehend aus Ticket- und Source Code-Verwaltungssystem einfügen lassen und muss als sinnvolle Ergänzung wahrgenommen werden, die Redundanz vermeidet und die Kommunikation mit den beteiligten Stakeholdern unterstützt. Die (unorganisierte) Verwendung von Textverarbeitungs- und Tabellenkalkulationsprogrammen kann diesen Anspruch nicht erfüllen.

Mit dem Fokus, die Spezifikation als Kooperationsmaterial aller beteiligten Stakeholder zu sehen, folgt man dem ersten Grundsatz des agilen Manifests. Entsprechend sollte die Plattform (a) einen technischen Fokus haben, um als dezidierte Lösung wahrgenommen werden zu können, und (b) eine rein semantische Erstellung der Spezifikation ermöglichen.

Vergleichbare Ansprüche finden sich bei Ticket- und Wiki-Systemen, wo Markup-basierte Texteingaben mittlerweile zum Standard gehören. Eine Plattform zur Spezifikationserstellung sollte diesem Ansatz folgen, um den Programmierer eine einfache und zeitgemäße Spezifikationspflege zu ermöglichen. Die Plattform muss die Markup-basierte Eingabe jedoch so aufbereiten, dass die Spezifikation professionell zur Kommunikation mit dem Kunden genutzt werden kann. Für die Typisierung und Verwaltung einzelner Anforderungen in einem größeren Dokument bieten sich natürlich XML-basierte Formate wie DocBook [Oa14] oder DITA [DP05] an. Hier wird es allerdings selbst mit geeigneter Softwareunterstützung für die Mehrzahl der Beteiligten im Projekt schwierig, ein auf dieser Basis erstelltes Dokument zu bearbeiten. Ein hoher Schulungsaufwand steht hier zunächst der Agilität im Weg.

Eine weitere Anforderung an die Plattform besteht darin, aufbauend auf der rein textuellen und unformatierten (Markup) Eingabe der Spezifikation die Verwaltung der Meta-Daten zu ermöglichen:

- Anforderungen müssen verfolgbar sein
- Anforderungen müssen organisierbar sein
- Änderungen am Inhalt müssen nachvollziehbar sein
- Verantwortlichkeiten müssen definierbar sein

In Softwareprojekten wird, unabhängig davon, ob eine Spezifikation gepflegt wird, ein Ticket-System zur Erfassung der Anforderungen verwendet. Dies kann auf zwei Arten geschehen:

(1) Explizit: Eine Anforderung wird, bevor sie implementiert wird, als Ticket/User Story im Ticket-System erfasst. Falls eine Spezifikation oder Auflistung der User Stories existiert, werden die Beschreibungen per "copy & paste" übernommen.

(2) Implizit: Falls der Ist-Zustand der Software nicht dem Soll-Zustand entspricht, wird ein Ticket erstellt, das die nötigen Änderungen, die Gründe dafür und das notwendige Wissen erfasst.

Das Problem mit diesem Ansatz ist, dass Tickets in der Regel nur einen transitorischen Charakter haben. Fachliches Wissen wird z.B. oft nur als Kommentar zu einem Ticket ergänzt. Es ist nicht praktikabel, aus der Summe aller Tickets den zu einem bestimmten Zeitpunkt definierten Soll-Zustand einer Software abzuleiten. Dafür wären Text-Dokumente notwendig, was jedoch eine manuelle und damit als redundant empfundene Pflege voraussetzen würde.

Tickets werden primär auf den Software-Stand bezogen und nicht auf die Spezifikation. Dabei sollte sich jedes Ticket primär auf die Spezifikation beziehen:

1. Eine neue User-Story sollte in der Spezifikation, sortiert nach Kapiteln und aus Kundensicht organisiert, angelegt werden.
2. Bug-Tickets bedeuten letztlich eine falsch dokumentierte oder falsch implementierte User-Story, die wiederrum in der Spezifikation stehen sollte.
3. Tickets zur Verbesserung der Software sollten im ersten Schritt als fachliche Ergänzung zu einer User-Story verstanden werden, die zuerst in der Spezifikation erfasst werden sollte.

Es darf kein Ticket erfasst werden, welches sich nicht auf eine fachliche Anforderung zurückführen lässt. Dies bedeutet keinesfalls, dass die Spezifikation *a priori* erstellt werden muss. Neue Anforderungen können bei Bedarf stichpunktartig erfasst werden, sobald ein fehlendes oder falsches Verhalten der Software dies notwendig macht. Die Spezifikation entsteht so automatisch als nebenläufiges Artefakt.

Die Lösung muss also darin bestehen, fachliches Wissen (sei es eine neue Anforderung oder eine Korrektur) ausnahmslos in einem Text-Dokument, nämlich der Spezifikation, festzuhalten. Gleichzeitig muss aber die Pflege der Meta-Daten, wie in einem Ticket-System, möglich sein. Basierend auf den gemeinsam gepflegten Daten und Meta-Daten sollte es dann möglich sein, für die verschiedenen Stakeholder geeignete Berichte und Analysen zu generieren. Insbesondere eine jederzeit aktuelle Spezifikation für den Auftraggeber (bspw. in Form eines formatierten PDF-Dokuments) oder eine detaillierte Historie für den Projektleiter (bspw. in Form einer Liste) wären wünschenswert.

4 Systeme für Spezifikations- und Anforderungsmanagement in agilen Projekten

Die Softwarelandschaft für das Spezifikations- und Anforderungsmanagement in agilen Projekten lässt sich in Bezug auf eine integrierte Gesamtsicht auf den aktuellen Systemstand grob in zwei Arten unterteilen:

Dokumentengeneratoren: Mit Hilfe spezifischer Regeln und durch die Auswertung von Meta-Informationen in den Artefakten des Projekts (Projekt-Wikis, Code-

Kommentare, etc.) wird eine aktuelle Dokumentation generiert. Diese kann fachliches Wissen, Implementierungshinweise und auch Anforderungen enthalten. Eine Überarbeitung der generierten Gesamtdokumentation (bspw. in Bezug auf die Änderung von Anforderungen etc.) und eine automatisierte, strukturelle Rückübertragung der Änderungen in die Quellsysteme sind in der Regel nicht möglich. Auf Quelltextebene stehen hier Systeme wie *JavaDoc*, *Doxygen* oder *Natural Docs* zur Verfügung, die mit Modellierungswerkzeugen und Wikis kombiniert werden können.

Ticket-Systeme: Mit Tickets werden Anforderungen, Change Requests und Bug-Reports abgebildet. Eine strukturierte, textuelle Gesamtdarstellung (Spezifikation) ist nicht vorgesehen. Wie oben beschrieben, haben Tickets einen transienten Charakter. Bezüge zu fachlichen Anforderungen gehen leicht verloren. Bekannte Ticket-Trackingsysteme wie *JIRA* oder *Redmine* unterstützen insbesondere mit agilen Erweiterungen die Verwaltung von Aufgaben in Sprint Task Boards oder Sprint Burndown Charts – die Rückführung oder Verankerung in eine Gesamtspezifikation im Kontext fachlicher Anforderungen wird nicht unterstützt.

Der Grundidee hier stellt, anders als bei Dokumentengeneratoren oder Ticket-Systemen, eine agile *Gesamtspezifikation* als zentrales Artefakt in den Mittelpunkt und betrachtet die Möglichkeit, alle Anforderungen und die daraus abgeleiteten Tickets im Gesamtkontext zu verfolgen.

5 Experimentelle Plattform *agileSpecs*

Basierend auf dem hier beschriebenen Lösungsansatz, agile Softwareprojekte und Spezifikationen zu verbinden, wurde die Plattform *agileSpecs*[13] entwickelt.

Die oben genannten Anforderungen an eine Spezifikationsplattform lassen sich wie folgt zusammenfassen:

- Zentrale Plattform für alle Beteiligten im Prozess
- Leicht zu erlernende Syntax
- Trennung von Text und Anforderungen/Spezifikationen
- Bereitstellung der Meta-Daten (Versionierung, Bearbeitungshistorie etc.)
- Verwaltung von Zuständen der Anforderungen, um z.B. neue Anforderungen zu markieren oder Fehler beim Ist-Zustand zu melden
- Verfolgung der Anforderungstexte und unterstützende Benachrichtigungen und Warnungen bei Änderungen
- Erzeugung spezifischer Berichte für das Gesamtprojekt
- Erzeugung einer grafisch ansprechenden Form der Spezifikation

agileSpecs bietet eine **zentrale Plattform**, die als SaaS-Lösung im Internet genutzt oder vom Projekt im Intranet betrieben werden kann.

[13] http://www.agilespecs.com

Der Projekt-Administrator vergibt den Beteiligten passwortgeschützte Accounts, die er bei Bedarf mit anderen Single-Sign-On-Lösungen verbinden bzw. synchronisieren kann. Die Lösung benötigt lediglich einen aktuellen Browser, die Installation lokaler Software ist nicht notwendig.

Als Auszeichnungssprache wird Markdown[14] verwendet (vgl. Abb. 1). Die **einfache Syntax** kann schnell erlernt werden. Markdown ist bei Wiki-Systemen sehr verbreitet und Plattformen wie beispielsweise GitHub (github.com) setzen Markdown für Tickets, Wiki und sogar Commit-Messages ein.

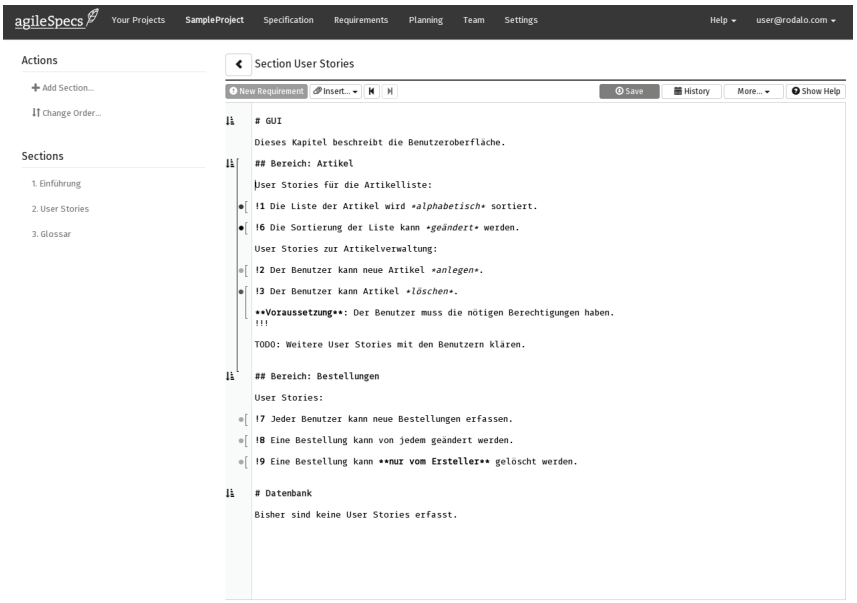

Abbildung 1: Markdown im System *agileSpecs*

Markdown verfügt über keine eigene Unterstützung für User Stories/Anforderungen. Im Rahmen des Projekts *agileSpecs* wurde daher untersucht, wie sich die Syntax von Markup-Sprachen um die Fähigkeit erweitern lässt, Anforderungen zu definieren. Dafür wurden die folgenden Anforderungen abgeleitet:

- Eine Anforderung muss als Strukturelement (ähnlich wie ein Kapitel, ein Aufzählungspunkt, etc.) erkannt werden können.
- Eine Anforderung muss über den gesamten Zeitverlauf hinweg eindeutig identifizierbar sein, damit spätere Änderungen festgestellt oder Bezüge hergestellt werden können.

Da Markdown als Markup-Sprache gewählt wurde, ließen sich die Anforderungen wie folgt abbilden:

[14] Für eine Übersicht dazu siehe: http://de.wikipedia.org/wiki/Markdown

Markdown verwendet zum Markieren von Strukturelementen einfache Symbole wie beispielsweise #, - oder *. Für Anforderungen wurde das Ausrufezeichen gewählt, da es (a) noch keine andere strukturelle Bedeutung hat und (b) passend für die Bedeutung von Anforderungen scheint.

Damit kein Konflikt mit dem normalen Ausrufezeichen am Satzende entsteht, gilt die Einschränkung, dass es am Zeilenanfang stehen muss. Das Ausrufezeichen markiert den Beginn des Anforderungstextes. Das Ende der Anforderung markiert automatisch das Absatzende, oder eine Zeile mit "!!!" als einzigem Inhalt. Weiterhin muss auf das Ausrufezeichen direkt, ohne Leerzeichen, eine Zahl folgen. Dies dient insbesondere der zweiten obigen Anforderung, der zu beschreibenden Anforderung eine eindeutige Kennung zuzuordnen. Die Zahl dient als stabile und eindeutige ID und unterliegt sonst keiner weiteren Semantik. Insbesondere darf aus der Stetigkeit der Zahlen keine Reihenfolge der Anforderungen abgeleitet werden. Niedrige Zahlenwerte weisen lediglich auf eine ältere Anforderung hin. Da es zu aufwendig wäre, die Vergabe der nächsten Nummer von Hand zu pflegen, vergibt *agileSpecs* automatisch fortlaufende neue Zahlen (default-Einstellung) und fügt diese während der Erstellung ein. Ein Beispiel mit zwei Anforderungen zeigt Abbildung 2.

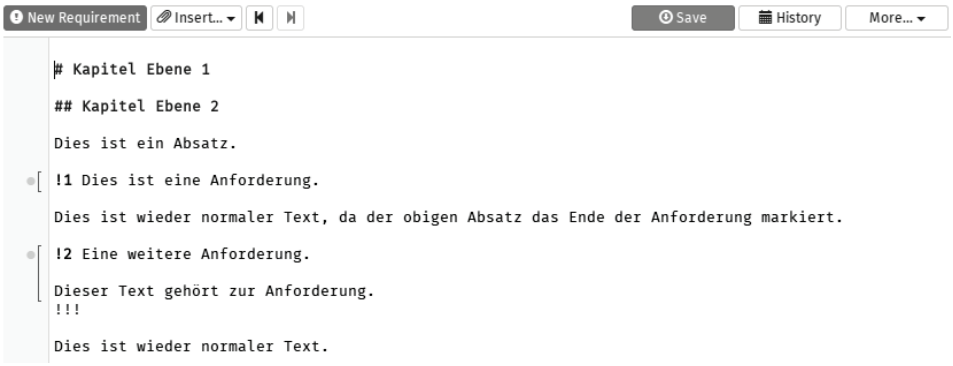

Abbildung 2: Markdown für die Beschreibung von zwei Anforderungen

Es hat sich gezeigt, dass diese einfachen Regeln ausreichen, um Anforderungen leicht in einer Markup-Sprache einzubetten. Werden Dokumente mit Anforderungen im System gespeichert, wird der folgende Prozess durchlaufen:

- Unter Anwendung der obigen Regeln extrahiert das System alle Anforderungen im Dokument. Die ID, die Beschreibung und die Zeilennummer können damit ausgelesen werden.
- Wurde die ID bisher nicht verwendet, handelt es sich um eine neue Anforderung.
- Ist die ID bekannt, wird die Beschreibung mit dem letzten bekannten Stand abgeglichen.
- Hat sich nur die Zeilennummer verändert, wird der neue Wert übernommen.

- Wenn sich die Beschreibung oder der Autor (der Benutzer, der gerade das Dokument speichert) verändert hat, wird der alte Eintrag der Anforderung historisiert und archiviert und die neuen Werte übernommen (vgl. Abb. 3).

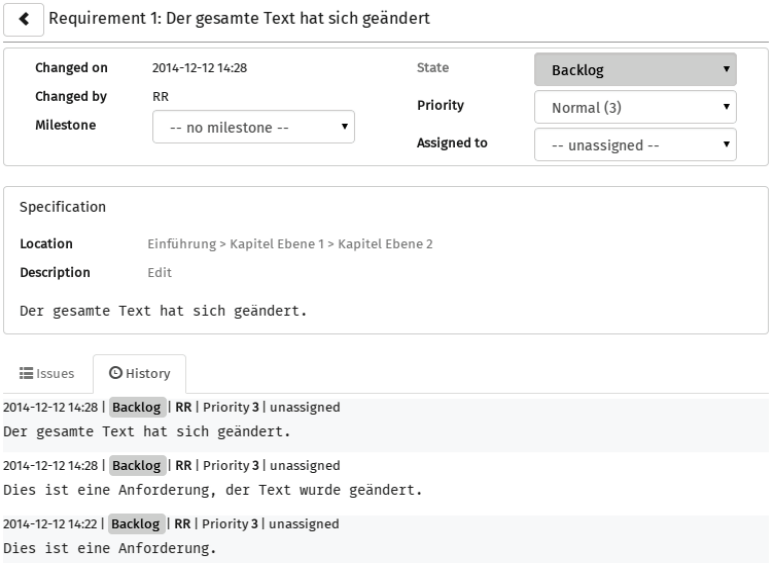

Abbildung 3: Historie der Änderungen einer Anforderung

Mit der ID der Anforderung können nun weitere Funktionen implementiert werden. So lassen sich beispielsweise Verantwortlichkeiten festhalten, Prioritäten setzen oder die Anforderungen in Meilensteine organisieren.

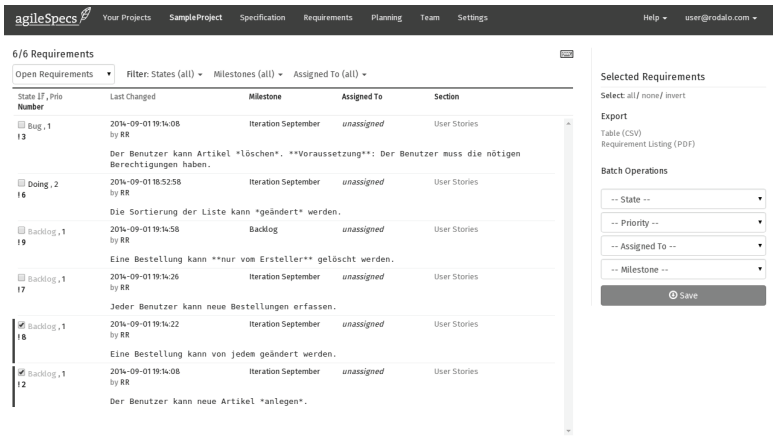

Abbildung 4: Filterung von Items aus dem Text

Der einzig notwendige Schritt, um mit *agileSpecs* eine Spezifikation zu erstellen, ist die Pflege des Textes mit Markdown. Dies kann begleitend zur Entwicklung zuerst stichpunktartig und bei Bedarf dann detailliert erfolgen.

Basierend auf der Kapitelstruktur von Markdown und den Markierungen für Anforderungen werden automatisch alle Anforderungen erkannt und deren Textbeschreibung, Autor und Zustand historisiert. Weiterhin können, wie in Ticket-Systemen üblich, die Anforderungen priorisiert und Benutzern zugeordnet werden. Anforderungen können unabhängig vom Text der Spezifikation kommentiert werden. Für jedes Projekt lassen sich beliebige Zustände für Anforderungen (z.B. *Neu*, *in Arbeit*, *Fehler*, *Implementiert*) definieren. Aufbauend auf der textuellen Ansicht der Spezifikation und den darin markierten Anforderungen können alle Anforderungen aufgelistet und gefiltert werden (vgl. Abb. 4). Das Schreiben der Spezifikation kann komplett mit der Pflege der Anforderungen verwoben werden, wodurch die zusätzliche Verwendung eines Ticket-Systems obsolet wird.

Obwohl die Spezifikation mit Markdown bereits formatiert und grafisch aufbereitet wird, sind zusätzliche Ansichten sinnvoll, die z.B. vom Auftraggeber genutzt werden bzw. den Fokus auf das Lesen und Verstehen der Spezifikation legen. *agileSpecs* stellt hier verschiedene Ansichten in Tabellenform sowie nach Bedarf formatierte PDF-Dokumente (vgl. Abb. 5) zur Verfügung.

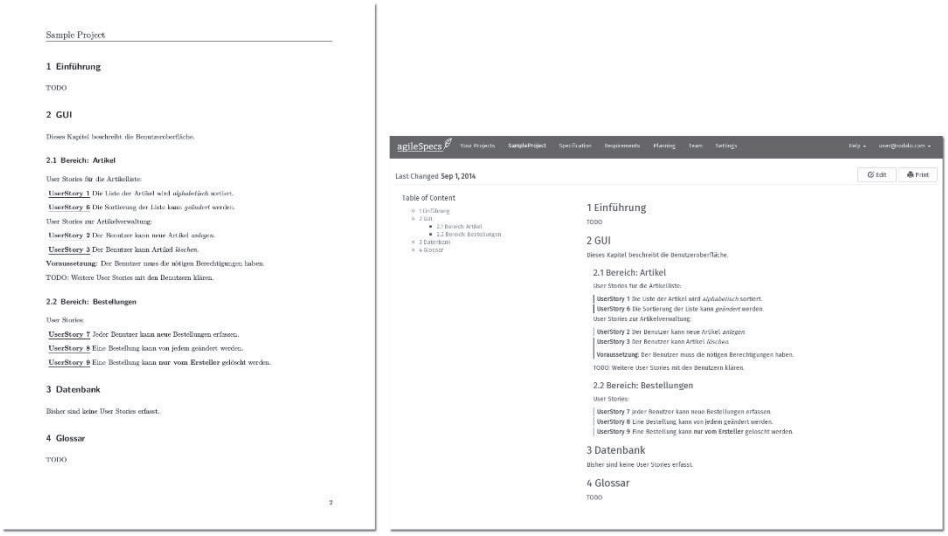

Abbildung 5: Ausgabe als PDF-Dokument (links) und als elektronischer Report (rechts)

Bei der Kommunikation mit dem Kunden sind Berichte bzw. Zusammenfassungen sinnvoll, mit denen beispielsweise Iterationen besprochen werden können. Ausgewählte Anforderungen können dafür als Excel-Datei exportiert oder zusammengefasst in einem PDF aufgelistet werden.

6 Erste Erfahrungen mit dem Einsatz von *agileSpecs*

agileSpecs wird zurzeit in kleineren Softwareprojekten evaluiert. Die ersten Berichte deuten auf eine für Projektmanager sehr schnell erlernbare Syntax und einfache Bedienung hin. Insbesondere die Generierung eines repräsentativen PDF-Dokuments der aktuellen Gesamtspezifikation scheint für die Kommunikation mit dem Kunden und anderen Projektpartnern einen Mehrwert darzustellen. Über Erfahrungen der Verwaltung einer Gesamtspezifikation über die Zeit kann in der jetzigen Pilotphase noch nicht berichtet werden - lediglich die erfolgreiche Nutzung als einfach zu bedienendes Werkzeug für die strukturierte Verwaltung und Verfolgung von Aufgaben wurde herausgestellt. In ersten Kommentaren der Nutzer werden die Wünsche geäußert, die Spezifikation auch offline bearbeiten zu können und die Nummerierung der Anforderungen individuell konfigurierbar zu machen.

Literaturverzeichnis

[Be01] Beck, K. et. al.: Manifesto for Agile Software Development (2001). http://www.agilemanifesto.org.

[BM14] Bittner, K.; Mines, C.; Whittaker, D.: Brief: Software Requirements Practices Are Ripe For Disruption. Forrester, Forrester Research, 25.04.2014. http://www.forrester.com/Brief+Software+Requirements+Practices+Are+Ripe+For+Disruption/fulltext/-/E-RES115852.

[DP05] Day, D.; Priestley, M.; Schell, D.: Introduction to the Darwin Information Typing Architecture. http://www.ibm.com/developerworks/xml/library/x-dita1/?ca=dgr-wikiaDita1. IBM Developerworks (2005) - abgerufen am 20.08.2014.

[MM11] McLeod, L.; MacDonell, S.G.: Factors that affect Software Systems Development Project Outcomes: A Survey of Research. ACM Computing Surveys, 43 (2011), 24–55.

[Oa14] OASIS - Organization for the Advancement of Structured Information Standards: DocBook Technical Committee Document Repository. http://www.oasis-open.org/docbook/ - abgerufen am 20.08.2014.

[SL06] Schwaber, C.; Leganza, G.; Daniels, M.: The Root of the Problem: Poor Requirements. Forrester Research – Trends (2006).