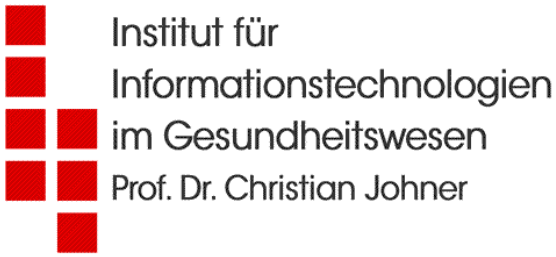

Einsatz agiler Vorgehensmodelle bei der Entwicklung medizinischer Software

Master Thesis

zur Erlangung des akademischen Grades

„Master of Science“

Donau Universität Krems

Dr.-Karl-Dorrek-Straße 30

A-3500 Krems

und

Institut für Informationstechnologien im Gesundheitswesen

Prof. Dr. Christian Johner

Kaiser-Joseph-Str. 274

D-79098 Freiburg

Verfasser: Bernhard Fischer

Matrikelnummer: 0664130

Abgabedatum: 12. Mai 2008

Gutachter: Dipl.-Inf. Christian Denger

Eidesstattliche Erklärung

Ich, Bernhard Fischer,

geboren am 02.12.1955 in Letmathe, jetzt Iserlohn, versichere, dass

- ich die vorliegende Masterarbeit selbständig verfasst, keine als die angegebenen Quellen und Hilfsmittel benutzt und mich auch sonst keiner unerlaubten Hilfen bedient habe,

- ich diese Arbeit bisher, weder im In- noch im Ausland einer/m Gutachter/in zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt habe

- diese Arbeit mit der von dem Gutachter beurteilten Arbeit übereinstimmt.

....

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abstract

Die Erstellung medizinischer Software unterliegt einer Vielzahl regulatorischer Vorgaben. Daher wird häufig als Vorgehensmodell ein schwergewichtiges Vorgehen gewählt, das an dem Wasserfallmodell angelehnt ist. Das ist auch verständlich, da aus den regulatorischen Vorgaben doch ein recht formales und dokumentenzentriertes Vorgehen ableitet werden kann.

Für die Softwareentwicklung zeitgenössischer Produkte werden aber Verfahren, die flexibler auf Änderungen eingehen, immer häufiger als zweckmäßigeres Vorgehen angesehen. Ziel solcher agiler Verfahren ist es, den Softwareentwicklungsprozess zu flexibilisieren, auf die zu erreichenden Ziele zu fokussieren und mehr auf die sozialen Probleme bei der Softwareentwicklung einzugehen, als dies klassische Vorgehensmodelle tun.

In dieser Arbeit wird untersucht, inwieweit solche agilen Verfahren für die Entwicklung medizinischer Software geeignet sind. Dazu werden zunächst die Eigenschaften agiler Vorgehensmodelle beschrieben und sodann an Hand von drei Beispielen (XP, Scrum und FDD) exemplifiziert. Anschließend werden die wesentlichen regulatorischen Anforderungen kurz charakterisiert.

Im Anschluss wird gezeigt, wie ein agiles Vorgehensmodell aussehen kann, das an den Eigenschaften der vorgestellten agilen Vorgehensmodellen angelehnt ist, aber auch den regulatorischen Vorgaben genügt. Dies Vorgehensmodell wird detailliert an Hand von Rollen, Aktivitäten und Artefakten beschrieben. Abschließend wird nachgewiesen, dass dieses Vorgehensmodell tatsächlich den regulatorischen Vorgaben genügt.

.

1 Einleitung

Von welchen Faktoren hängt die erfolgreiche Erstellung von Software ab? In den Anfangstagen der Softwareentwicklung schienen die Probleme der Softwareentwicklung überschaubar. Die Anzahl der vorhandenen Computer war sehr begrenzt, die Anwender und die Programmierer waren häufig dieselben Personen. Das Erlernen und korrekte Anwenden einer Programmiersprache stand im Vordergrund.

Dies änderte sich Mitte der 1960er Jahre. Man stellte erstmalig fest, dass weder die eingesetzten Methoden und Verfahren, noch die vorhandenen Werkzeuge für die Softwareentwicklung für die geänderten Anforderungen angemessen waren[1]. Die später so genannte Softwarekrise war geboren. Edsger Dijkstra hat dies 1972 so formuliert: „Als es noch keine Rechner gab, war auch das Programmieren noch kein Problem, als es dann ein paar leistungsschwache Rechner gab, war das Programmieren ein kleines Problem und nun, wo wir gigantische Rechner haben, ist auch das Programmieren zu einem gigantischen Problem geworden.“[2]

Auf der mittlerweile legendären NATO-Konferenz in Garmisch[3] und der Nachfolgekonferenz im darauf folgenden Jahr in Rom[4] wurde die Forderung erhoben, dass sich die Entwickler von der „Kunst“ der Programmierung lösen sollten, und die Softwareentwicklung vielmehr ingenieurmäßig betreiben sollten. Dafür wurde der Begriff Software-Engineering[5] gebildet.

Software Engineering, in Deutschland häufig auch Software-Technik genannt, definiert Balzert als die „zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen. Zielorientiert bedeutet dabei die Berücksichtigung z.B. von Kosten, Zeit, Qualität“[6]. Bei anderen Autoren finden sich ähnliche Definitionen[7].

---

[1] [Dijkstra1972].

[2] [Dijkstra1972].

[3] [NATO1968].

[4] [NATO1969].

[5] Der Begriff “Software Engineering” wurde von F.L. Bauer im Rahmen einer „Study Group on Computer Science“ der NATO geprägt.

[6] [Balzert96], S. 36.

[7] Etwa Barry W. Boehm: “Software Engineering: The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop,

Seit der Software Engineering Konferenz in Garmisch lag der Schwerpunkt der Forschung und Praxis im Finden und Etablieren effizienter Vorgehensweisen und Methoden. Sobald für einen Bereich Methoden gefunden wurden, versuchte man sie weltweit durchzusetzen und - wenn Schwachstellen auftraten - durch immer mehr Systematik und Regelungen zu verbessern. Unsere Verfahren und Vorgehensmodelle wurden abstrakter und komplexer, je mehr wir über die Methoden und Verfahren lernten[8].

Die Softwareerstellung wurde dadurch sicherlich auf eine sachliche Basis gestellt, die bei der Erstellung von Software auftretenden Probleme wurden aber nur unzureichend gelöst. Noch immer ist die Softwareerstellung in der Krise. 1994 stellte der CHAOS-Report der Standish Group fest, dass 31 % der Projekte vor der Fertigstellung abgebrochen wurden, 53 % der Projekte wesentlich teurer waren und viel länger dauerten, als ursprünglich geplant war und lediglich 16 % der Projekte finanziell und zeitlich wie geplant beendet werden konnten[9]. 2006 hatte sich die Situation zwar leicht verbessert, so dass nur noch 19 % der Projekte vor der Fertigstellung abgebrochen wurden, aber 46 % der Projekte waren immer noch wesentlich teurer und dauerten viel länger als ursprünglich geplant. Immerhin konnten mittlerweile 35 % der Projekte finanziell und zeitlich wie geplant beendet werden[10].

Wie ist das möglich, wo doch seit 1968 die entwickelten Prinzipien, Methoden, Konzepte, Notationen und Werkzeuge um soviel besser geworden waren? In den seit Mitte der 90er-Jahre entstandenen neuen Entwicklungsansätzen wird behauptetet, dass die teils sehr rigiden Vorgaben der Vorgehensmodelle ursächlich für die Misere der Softwareentwicklung sind. Diese Ansätze stellen daher im Gegensatz zu den klassischen Vorgehensmodellen leichtgewichtige, anpassungsfähige, kleine, schmale Ansätze und die explizite Einbindung vieler empirisch ermittelter "Best Practices" in den Mittelpunkt. Sie erlauben den Projektbeteiligten somit nicht nur ein hohes Maß an eigener Gestaltungsfreiheit für den Entwicklungsprozess, sondern sie fordern nachdrücklich zum regelmäßigen Überprüfen und Anpassen des Vorgehens auf. Zudem sollen die sozialen Probleme der Softwareentwicklung mehr Berücksichtigung finden, als dies bei klassischen Vorgehensmodellen der Fall ist[11].

Trotz ihrer unterschiedlichen Schwerpunkte bauen alle diese Vorgehensmodelle, wie Scrum, FDD und Extreme Programming auf ähnlichen Prinzipien auf. Im Februar 2001 traf sich daher eine Gruppe von Schöpfern und Verfechtern solcher Vorgehensmodelle in Utah, und einigte sich

operate, and maintain them" in Software Engineering, in: IEEE Transactions on Computers, Dec. 1976, S 1226 – 1241, zitiert nach [Balzert96], S. 35.

[8] [Broy2001], S. 11ff.

[9] [CHAOS1994].

[10] [CHAOS2006].

[11] [Wallmüller2007], S. 25.

auf die Gemeinsamkeiten ihrer jeweiligen Vorgehensmodelle. Diese Gemeinsamkeiten wurden dann in Form des so genannten agilen Manifests veröffentlicht[12]. Sie änderten dabei den bis dahin für diese Ansätze verwendeten Begriff "leichtgewichtig" in den Begriff "agil", der sich in der Fertigungsindustrie im letzten Jahrzehnt bewährt hatte[13].

Was bedeutet das nun für die Erstellung von Software für medizinische Geräte? Durch das Medizinproduktegesetz soll sichergestellt werden, dass nur medizinisch und technisch unbedenkliche Medizinprodukte in Verkehr gebracht werden. Daher wird für das Produkt die Einhaltung von grundlegenden Anforderungen gefordert, die abstrakt die Sicherheit des Produktes sicherstellen soll. Werden Medizinprodukte unter Berücksichtigung der einschlägigen harmonisierten Normen hergestellt, so wird eine Übereinstimmung mit diesen grundlegenden Anforderungen angenommen (Vermutungswirkung). Alle für Medizinprodukte harmonisierten Normen sind vom Europäischen Rat veröffentlicht und über das Internet zugänglich[14].

In den dort aufgeführten harmonisierten Normen werden zahlreiche Dokumente und Überprüfungen gefordert. Aber auch durch die Einhaltung dieser Normen kann die Sicherheit so hergestellter Software letztendlich nicht sichergestellt werden. [IEC prEN 62304] sagt dazu: „Es gibt wirklich keine bekannte Methode, um für irgendeine Art von Software eine hundertprozentige Sicherheit zu gewährleisten.

Es gibt drei wesentliche Prinzipien, die helfen, Sicherheit für Medizingeräte-Software sicherzustellen:

- Risikomanagement
- Qualitätsmanagement
- Software-Technik“[15]

Daher wird in der Praxis für die Erstellung medizinischer Software häufig ein schwergewichtiges Vorgehensmodell gewählt, das häufig an dem Wasserfallsmodell angelehnt ist. Ist dies aber wirklich notwendig? Lassen sich die Vorteile agiler Vorgehensmodelle nicht auch für die Entwicklung medizinischer Software einsetzen?

[12] http://agilemanifesto.org/, letzter Zugriff 26.04.2008

[13] [Poppendieck], Introduction, Page xxi.

[14] http://ec.europa.eu/enterprise/newapproach/standardization/harmstds/reflist/meddevic.html (letzter Zugriff: 25.04.2008) .

[15] [IEC prEN 62304], Anhang B 4, Seite 36.

1.1 Ziel der Arbeit

In dieser Arbeit soll nun untersucht werden, inwieweit solche agilen Vorgehensmodelle unter Einhaltung regulatorischer Vorgaben auch für die Entwicklung medizinischer Software geeignet sind. Basierend auf den regulatorischen Vorgaben, sind die Defizite agiler Verfahren, soweit es die Entwicklung medizinischer Software betrifft, aufzuzeigen und an Hand von konkreten Beispielen (XP, Scrum und FDD) zu illustrieren. Im Anschluss soll aufgezeigt werden, wie agile Vorgehensmodelle erweitert werden können, damit sie für die Erstellung medizinischer Software geeignet sind. Damit wird die zentrale Frage dieser Arbeit: „Wie können agile Vorgehensmodelle erweitert werden, um den regulatorischen Anforderungen für die Entwicklung medizinischer Software zu genügen?“ beantwortet.

2 Vorgehensmodelle

In diesem Kapitel wird zunächst erläutert, welche Bedeutung Vorgehensmodelle in dem Softwareentwicklungsprozess haben. Anschließend werden wichtige Eigenschaften von Vorgehensmodellen kurz skizziert. Danach wird erläutert, wie sich agile Vorgehensmodelle von Phasenmodellen und inkrementell-iterativen Vorgehensmodellen unterscheiden. Dabei wird detailliert auf die im agilen Manifest angeführten agilen Prinzipien und Werte eingegangen.

2.1 Was sind Vorgehensmodelle?

Als Antwort auf die Software-Krise entstand um 1970 eine produktorientierte Sichtweise, die Software als eigenständiges Produkt aus Programmen und Dokumentation und die Softwareentwicklung als Produktion im Sinne einer Fertigung betrachtete. Maßgebliche war dabei die Kontrollierbarkeit des Herstellungsprozesses, die Gewährleistung vorgegebener Anforderungen und Qualitätsmerkmale durch die Produkte und die Einhaltung eines fest kalkulierbaren finanziellen und terminlichen Rahmens[16].

Dazu ist es aber erforderlich, den organisatorischen Rahmen für die Softwareentwicklung festzulegen. Prozessmodelle, im deutschen Sprachraum auch als Vorgehensmodelle bekannt, haben nun den Anspruch, eine solche Strukturierung der Softwareentwicklung zu leisten. In der Literatur finden sich nun unterschiedliche Definitionen dessen, was genau unter einem Vorgehensmodell verstanden wird. [Balzert2000] stellt dazu fest: „Analog zu jedem Geschäftsprozess soll auch jede Softwareerstellung in einem festgelegten organisatorischem Rahmen erfolgen. [...] Ein Prozessmodell – auch Vorgehensmodell genannt – beschreibt jeweils einen solchen Rahmen. Ein definiertes Prozessmodell soll Folgendes festlegen:

- Reihenfolge des Arbeitsablaufes (Entwicklungsstufen, Phasenkonzepte),
- Jeweils durchzuführende Aktivitäten,
- Definition der Teilprodukte einschließlich Layout und Inhalt,
- Fertigstellungskriterien (Wann ist ein Teilprodukt fertig gestellt?).
- Notwendige Mitarbeiterqualifikationen,
- Verantwortlichkeiten und Kompetenzen,

[16] [Balzert2000], S. 446.

- Anzuwendende Standards, Richtlinien, Methoden und Werkzeuge.“ [17]

[Versteegen2] definiert hingegen etwas allgemeiner: „Ein Prozessmodell ist eine Beschreibung einer koordinierten Vorgehensweise bei der Abwicklung eines Vorhabens. Es definiert sowohl den Input, der zur Abwicklung der Aktivität notwendig ist, als auch den Output, der als Ergebnis der Aktivität produziert wird. Dabei wird eine feste Zuordnung von Workern vorgenommen, die die jeweilige Aktivität ausüben.“ [18]

[Winter] wiederum stellt fest: „Vorgehensmodelle stellen den Rahmen dar, in dem die Entwicklungstätigkeiten organisiert werden [...]. Sie zerlegen den Entwicklungsprozess gewöhnlich in zeitlich aufeinander folgende *Phasen*, die jeweils ein Zeitintervall mit den darin stattfindenden Aktivitäten beschreiben. Als *Aktivität* wird dabei eine inhaltlich zusammenhängende (Teil-) Tätigkeit bezeichnet, die von einigen wenigen Personen in einem bestimmten Zeitraum ausführbar ist. Jede Phase schließt mit einem Meilenstein ab, bei dem überprüft wird, ob die geforderten Ergebnisse erbracht worden sind.

Vorgehensmodelle beschreiben die notwendigen Aktivitäten nebst ihren Voraussetzungen und Ergebnissen und legen somit die Ablauforganisation fest; sie spezifizieren quasi die Geschäftsprozesse der Softwareentwicklung selbst.“[19]

Diese Definitionen machen klar, dass für die Entwicklung und den Einsatz von Vorgehensmodellen unterschiedliche Gliederungsansätze gewählt werden können: Man kann primär ergebnisorientierte[20] oder primär vorgangsorientierte[21] Ansätze wählen. Daneben muss kritisch geprüft werden, bis zu welchem Grad „kreative Arbeit“, um die es sich bei der Softwareentwicklung weitgehend handelt, überhaupt zweckmäßig reguliert werden kann. Im Hinblick auf das Rationalisierungsziel wird ein einerseits hoher Grad an Regulierung und im Hinblick auf die Motivation und die Entfaltung der Kreativität werden andererseits Freiräume und unregulierte Arbeitsabschnitte in der Entwicklung gefordert[22].

Vielfach beschreibt ein Vorgehensmodell allerdings nicht nur das Vorgehen bei der Software-Entwicklung, sondern wird um eine Notation, wie etwa die Unified Modeling Language (UML) sowie einer Reihe von Managementpraktiken ergänzt[23].

[17] [Balzert2000], S. 449.

[18] [Versteegen2000], Kapitel 2, S. 21ff.

[19] [Winter], Kapitel 3, S. 27.

[20] Das V-Modell 97 ist ein ergebniszentriertes Vorgehensmodell.

[21] Der Rational Unified Process (RUP) ist ein aktivitätsorientiertes Vorgehensmodell.

[22] [Wallmüller], S. 25, [Hamilton], S. 74ff, [Schwaber], S. 2f.

[23] [Versteegen], S. 51ff.

Demnach beschreibt ein Vorgehensmodell den Entwicklungszyklus eines Software-Produkts in Form von Aktivitäten und Ergebnissen. Durch das Vorgehensmodell wird weiterhin festgelegt, in welcher Reihenfolge welche Aktivitäten durchgeführt werden können und welche Qualifikationen die an der Erstellung der Ergebnisse beteiligten Mitarbeiter haben müssen. Ein Vorgehensmodell liefert dann zusätzlich Informationen über die einzusetzenden Entwicklungsmethoden und -werkzeuge.

Ein sehr bekanntes und in der Softwaretechnik früher häufig eingesetztes Vorgehensmodell ist das so genannte Wasserfallmodell. Der Entwicklungszyklus wird in diesem Modell in einzelne Phasen unterteilt, die jeweils vollständig abgeschlossen werden müssen, bevor mit der nächsten Phase begonnen werden kann. Die Phasen werden jeweils mit einem Endergebnis in Form von Dokumenten oder Programmcode beendet. Die erzeugten Ergebnisse werden in der darauf folgenden Phase als Basis für die weitere Entwicklung verwendet. Das Wasserfallmodell war eine Antwort auf die in den 1960er Jahren üblichen Ad-hoc-Modelle zur Softwareentwicklung und wurde erstmals im Jahr 1970 von dem amerikanischen Computerwissenschaftler Winston Royce beschrieben. Royce nennt in seinem Artikel folgende nach wie vor üblichen Kernphasen einer Softwareentwicklung[24]:

- **Anforderungen**: In der Anforderungsphase werden die Kundenanforderungen ermittelt und ein gemeinsames Verständnis mit dem Kunden aufgebaut.
- **Analyse**: In dieser Phase werden Kundenanforderungen analysiert und in Anforderungen an das System überführt. Gegebenenfalls wird ein Analysemodell erstellt.
- **Entwurf**: In der Entwurfsphase wird das Softwaresystem zerlegt. Festlegung der wesentlichen Eigenschaft des Systems.
- **Implementierung**: Die Implementierungsphase umfasst die eigentliche Programmierung der einzelnen Komponenten nach den Vorgaben der Entwurfsphase.
- **Testphase:** Die erstellten Implementierung des Gesamtsystems wird gegen die Anforderung geprüft.
- **Einsatz**: Schlussendlich wird das System in Betrieb genommen und gegebenenfalls nach den Wünschen der Anwender erweitert. Etwaige Fehler werden registriert und nach Möglichkeit behoben.

Die strikte Abfolge der Phasen entlang des Wasserfalls beruht auf der Annahme, dass eine umfassende und sorgfältige Planung zu verlässlichen Voraussetzungen führt, die eine schnelle und damit kostengünstige Durchführung der späteren Phasen ermöglichen. Zudem entsteht entlang

[24] [Royce], S. 328f.

der Entwicklungsprozesse eine detaillierte Dokumentation aller Vorgänge und Ergebnisse[25]. Diese kann als partielles Endprodukt dienen, das es ermöglicht, die Entwicklung temporär auszusetzen und wieder aufzunehmen oder neue Mitarbeiter schneller in ein Projekt zu integrieren. Ein weiterer Vorteil des Wasserfalls ist seine simple Struktur, die unterstützt, das Modell auch in großen Teams effizient zu kommunizieren und alle Prozesse von zentraler Stelle aus zu steuern.

Das Wasserfallmodell fand bereits in den 1970ern vielfältige Verwendung in der Entwicklung größerer Softwaresysteme und es wird bis heute in vielen Unternehmen – teilweise leicht modifiziert - eingesetzt und an Hochschulen gelehrt. Interessanterweise beschreibt Royce in seinem Artikel jedoch gerade keinen strikt sequentiellen Ablauf in der Entwicklung, sondern nimmt diesen nur als Basis für die Einsatz eines flexibleren Vorgehens, eine Basis, die isoliert angewendet, so Royce, „riskant und fehleranfällig ist“ [26]. Aufbauend auf seiner Kritik am Wasserfallmodell beschreibt Royce ein weniger striktes Vorgehensmodell, dessen Prozesse über kontrollierte Rückkoppelungen flexibler auf sich ändernde Bedingungen reagieren können. Royce wies nachdrücklich darauf hin, dass das Wasserfallmodell nur für besonders einfache Projekte geeignet ist, und empfahl das mehrfache Durchlaufen der Phasen: „If the computer program in question is being developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations areas are concerned“[27].

2.2 Iterative und inkrementelle Entwicklung

Wie oben ausgeführt liefern Phasenmodelle wie das Wasserfallmodell lediglich statische Strukturierungen des Softwareentwicklungsprozesses. Seine Stärken spielt es vor allem bei Projekten aus, deren Anforderungen sich zu Anfang sehr genau bestimmen lassen, deren Zielsetzung also vorhersehbar ist. Empfindlich reagiert es auf sich ändernde Bedingungen, demnach Entwicklungen, deren Ziele erst durch Erkenntnisse während des Verlaufs genauer bestimmt werden können. Bei der Entwicklung neuer und innovativer Produkte, muss von solchen „beweglichen Zielen“ ausgegangen werden. Dies trifft im erhöhten Maß zu, wenn der Kunde oder der Endnutzer als integrer Teil der Prozesse verstanden wird[28]. Der Kunde oder Endnutzer weiß nicht genau, was er will. Er weiß (teilweise), was er will, hat jedoch Schwierigkeiten seine Vorstellungen komplett und konkret darzulegen.

[25] [Versteegen2000], S. 28f.

[26] [Royce], S. 334.

[27] [Royce], S. 338.

[28] Uncertainty is inherent and inevitable in software projects and processes ([Larman2004], S. 102ff).

Details und Bedingungen werden ihm erst in der Umsetzung klar. Zudem sind vielfach die einzelnen Details seiner Anforderungen und deren Zusammenhänge komplexer als zu Beginn angenommen. Während der Entwicklung ändert er seine Meinung. Externe Einflüsse – wie Konkurrenzprodukte oder der Markt – können ebenfalls zu modifizierten oder verworfenen Anforderungen führen. Vor allem in der Entwicklung von Softwaresystemen spielen diese Faktoren eine gewichtige Rolle. Obwohl häufig als ein serieller Vorgang wahrgenommen, ist Softwareentwicklung daher in den meisten Fällen ein hoch dynamischer Prozess in einem sich rasant wandelnden Umfeld, der von dauerhaft hohen Änderungsraten ausgehen muss[29].

Neben diesen sequentiellen Vorgehensmodellen existieren bereits seit längerer Zeit alternative Ansätze in Form von iterativen Entwicklungsmodellen. Diese teilen die gesamte Entwicklung in mehrere einzelne Iterationen oder Kleinst-Projekte auf, in denen mehrere oder sogar alle Disziplinen von der Analyse bis zur Auslieferung enthalten sind. Diese Iterationen werden sodann mehrfach hintereinander oder sogar parallel durchgeführt. Das Ziel jeder Iteration ist die Veröffentlichung eines stabilen, integrierten und getesteten partiell funktionsfähigen Systems. Häufig handelt es sich dabei um eine Veröffentlichung zu Demonstrationszwecken; die eigentliche Auslieferung an den Markt erfolgt erst nach dem Durchlauf mehrerer oder aller Iterationen, d.h. man unterschiedet zwischen Iterationen und Releases. Jede Veröffentlichung hat jedoch den Zweck, möglichst viel Feedback zu generieren, auf dessen Grundlage die Anforderungen und Ziele der nächsten Iteration gefunden werden. Feedback kann dabei direkt vom Kunden oder Endnutzer kommen, von einer internen Test-Abteilung oder auch vom Quellcode selbst[30]. Die meisten agilen Modelle empfehlen Iterationen mit einer Länge zwischen einer und sechs Wochen. In großen Projekten sind auch drei Monate pro Iteration nicht unüblich. Tendenziell sollte eine Iteration jedoch so kurz wie möglich sein – so lang dies der Organisationsaufwand und die Effektivität des Teams zulassen[31].

Alle modernen Modelle empfehlen zudem das so genannte Time-Boxing. Dabei wird die Iteration zu Beginn mit einem fixen Zeitrahmen geplant, der nicht verändert werden darf. Normalerweise existieren vier „Stellschrauben“ im Projektmanagement: Zeit, Umfang, Ressourcen und Qualität. Beim Time-Boxing wird nun die Variable Zeit fixiert und so nur drei Möglichkeiten übrig gelassen, ein Projekt zu steuern. Bevorzugt wird bei auftretenden Problemen der Umfang justiert, indem an Hand der Priorität einzelne Aufgaben auf die nächste Iteration verschoben werden. Obwohl einzelne Iterationen meist zeitlich fixiert werden, sehen nur die wenigsten Modelle eine konstante Länge der Iterationen vor. Vielmehr kann jede Iteration – im Rahmen – variabel auf die anstehenden Aufgaben angepasst werden. Eine weitere wichtige Grundregel der meis-

[29] [Larman2004], S. 50.

[30] [Larman2004], S. 9 ff.

[31] [Larman2004], S. 267.

ten modernen Modelle besagt, dass der Umfang einer Iteration, während diese läuft, von niemandem außerhalb des Teams geändert werden sollte. Dies beugt chaotischen Zuständen vor, ermöglicht den Entwicklern sich auf einen bestimmten Umfang an zu erledigenden Aufgaben zu konzentrieren und die Komplexität einer Iteration zu beschränken[32].

Es existieren im Wesentlichen zwei Ansätze, den Umfang der nächsten Iteration zu bestimmen: die vom Risiko und die vom Benutzer gesteuerte Planung. Die risikogesteuerte Variante priorisiert die für den Erfolg des Produkts am kritischsten und am schwersten umzusetzenden Aufgaben. Dabei kann es sich sowohl um die für die Grundarchitektur essentiellen oder auch um die für die Marktfähigkeit außerordentlich wichtigen Produkteigenschaften handeln. Die vom Benutzer gesteuerte Planung lässt im Gegensatz dazu dem Kunden oder Endnutzer die Wahl, welche Features für ihn den meisten Wert haben. Alle iterativen Modelle empfehlen eine Mischung beider Ansätze, meist mit einem Schwerpunkt auf den „riskanten“ Teilen des Produkts, in frühen Iterationen kombiniert mit einem späteren Übergang zur benutzergesteuerten Planung[33].

Der grundlegende Zweck der iterativen Vorgehensmodelle ist eine möglichst häufige Rückkopplung der Anforderungen, des Designs und der Umsetzung mit der technischen sowie sozialen Realität. Im Gegensatz zum spät scheiternden [34] Wasserfallmodell werden Prognosen schneller validiert, Fehler somit früher aufgedeckt und behoben. Dies umfasst nicht nur die technische Machbarkeit des Produkts, sondern auch soziale Aspekte, wie die Fähigkeiten des Teams und die Zusammenarbeit mit dem Kunden. Eine iterative Entwicklung nähert sich schrittweise den realen Anforderungen an, anstatt diese früh zu prognostizieren. Man spricht in diesem Zusammenhang auch von adaptiver statt vorausschauender Planung. Änderungen der Anforderungen sind somit immanenter Teil der Fortentwicklung und sollten forciert statt vermieden werden. Beide Disziplinen haben in allen modernen iterativen Modellen ihren Schwerpunkt in den frühen Phasen, beschränken sich aber auf den unmittelbar zu diesem Zeitpunkt unabdingbaren Umfang[35]. Ein weiterer wesentlicher Vorteil gegenüber einem sequenziellen Vorgehen besteht in der sehr frühen Verfügbarkeit eines partiell fertig gestellten Produkts. Das Benutzerfeedback kann somit nicht nur häufiger und früher, sondern auf der Grundlage des realen Produkts gesammelt werden. Vor allem im Interfacedesign taucht immer wieder das Problem auf, dass der Benutzer, um seine Vision eines Produkts überprüfen zu können, mit einer Lösung unmittelbarer konfrontiert werden muss.

[32] [Larman2004], S. 13f. und S. 54f.

[33] [Larman2004], S. 12.

[34] [Larman2004], S. 62., im Original: fail-late

[35] [Larman2004], S.11 und S. 51

Eine auf die Interessen der Benutzer ausgerichtete Entwicklung von Software kann weder von den Entwicklern noch von den Benutzern allein durchgeführt werden. Die Entwickler sind die Experten im Bereich der Computertechnologie und die Benutzer verfügen über Fachwissen im Einsatzbereich der Software. Daher ist die gleichberechtigte Beteiligung von Benutzer und Entwickler am Entwicklungsprozess eine notwendige Bedingung für die Herstellung einer angemessenen Lösung. Weiterhin ist eine vollständige Ermittlung der Anforderungen hinsichtlich des Funktionsumfangs und der Qualitätsmerkmale des Softwaresystems prinzipiell unmöglich. Eine auf Sozialverträglichkeit und Wirtschaftlichkeit abzielende Systementwicklung muss von der evolutionären Entwicklung des Umfeldes der eingesetzten Software ausgehen. Es verändern sich beispielsweise normative, ökonomische und technologische Gegebenheiten, die unter Umständen auch durch den Einsatz der Software hervorgerufen werden können. Hieraus erwachsen veränderte und neue Anforderungen an das eingesetzte System.

2.3 Agile Softwareentwicklung

Die agile Softwareentwicklung entstand als Gegenbewegung gegenüber den vorhandenen, als zu schwergewichtig und bürokratisch angesehenen Vorgehensmodellen. Man wünschte sich Softwareentwicklungsprozesse, die mehr auf die sozialen Probleme bei der Softwareentwicklung eingehen, leicht umzusetzen und flexibel anzupassen sind.

Erste Ansätze sind dabei bereits Anfang der 1990er Jahre zu finden. Große Popularität erreichte die agile Softwareentwicklung erstmals 1999, als Kent Beck und andere das erste Buch zu „Extreme Programming“[36] veröffentlichten. Das große Interesse an „Extreme Programming“ ebnete den Weg auch für andere agile Prozesse und Methoden.

2.3.1 Das agile Manifest

Das Agile Manifest, im Februar 2001 auf einer Konferenz in Utah von 17 führenden Methodikern aus der Softwareentwicklung erarbeitet, stellt bis heute den gemeinsamen Nenner für die verschiedenen agilen Vorgehensmodelle dar. So unterschiedlich die konkret empfohlenen agilen Praktiken auch sind, befürworten doch alle Vertreter ein zum dokumentationsgestützten, schwergewichtigen Entwicklungsprozess konträres Modell und beziehen sich ausnahmslos auf die im Agilen Manifest formulierten agilen vier Werte und zwölf Prinzipien.

[36] Deutsch als [Beck2000]

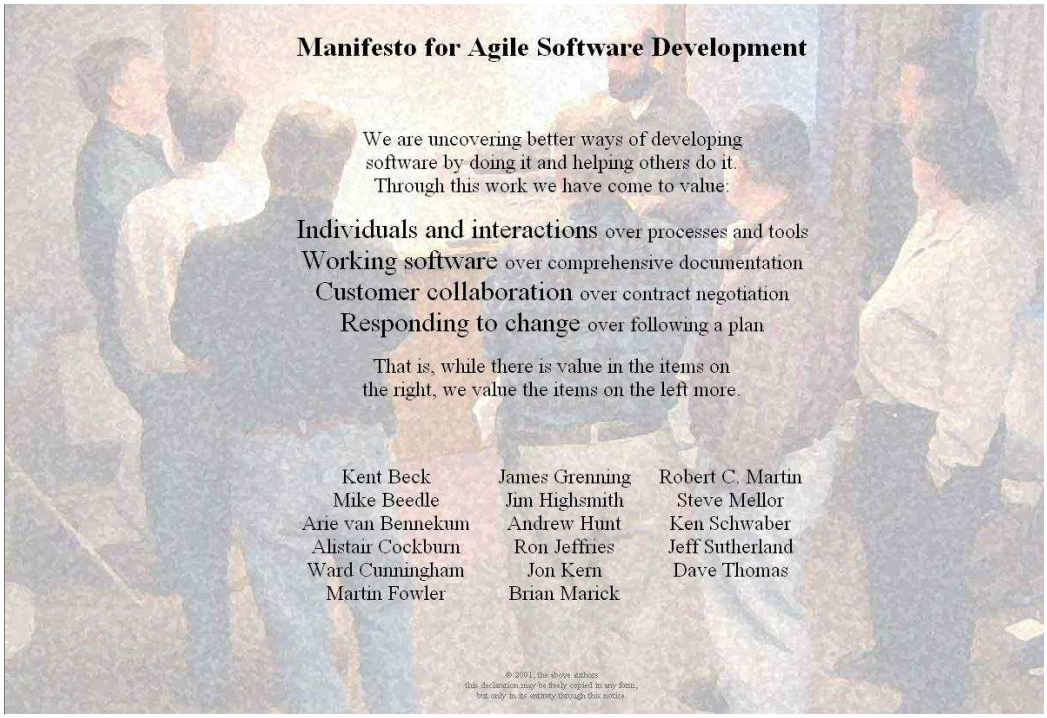

Abbildung 1 : Das agile Manifest[37]

2.3.1.1 Individuen und Interaktionen

„Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge“[38]

Menschen stehen im Zentrum agiler Vorgehensmodelle, mit ihnen steht und fällt ein Projekt. Großer Wert wird daher auf gute Zusammenarbeit und das Zwischenmenschliche im Team gelegt, die Entwickler und sonstigen Teammitglieder werden dadurch motiviert. Die Eigenverantwortung jedes Einzelnen wird stark betont. Ein guter Plan sorgt noch lange nicht für ein erfolgreiches Projekt, vielmehr schränken Prozesse vielfach den Handlungsspielraum ein und bieten, wenn es bei der Zusammenarbeit Probleme gibt, geringen Mehrwert. Darüber könne sie zur Demotivierung des Teams führen. Es soll daher nur das absolut notwenige Minimum geplant und festgeschrieben werden[39].

2.3.1.2 Lauffähige Software

„Funktionierende Software ist wichtiger als umfassende Dokumentation“[40]

[37] www.agilemanifesto.org, letzter Zugriff 20.04.2008

[38] Übersetzung nach [Dogs], S. 34.

[39] [Wallmüller2007], S. 25, [Bleek2008], S. 13f.

[40] Übersetzung nach [Dogs], S. 35.

Für die Autoren des Agilen Manifests hat funktionierende Software höchste Priorität. Eine verständlich formulierte Dokumentation ist nicht zwingend schlecht, doch bedeutet diese nicht nur einen hohen Erstellungs- und Pflegeaufwand. Jedes Team soll entscheiden, welche Parts einer Dokumentation erforderlich sind, und nur diese erstellen. Der Fokus muss daher immer auf der Software selbst liegen. Das agile Manifest propagiert daher ein evolutionäres Entwicklungsmodell mit frühen, häufigen und kontinuierlichen Auslieferungen funktionierender Software. Der empfohlene Rhythmus kann von einigen Wochen bis zu einigen Monaten betragen. Sehr zügig nach dem Start eines Projektes wird dem Kunden bereits eine erste Software vorgelegt; es folgen schnelles Feedback und Änderungen. Ein derartiges Vorgehen erspart vielfach die Verwendung von Dokumenten zur Kommunikation mit dem Kunden, und fördert darüber hinaus das Vertrauen des Kunden in das Team und den Softwareentwicklungsprozess[41].

2.3.1.3 Zusammenarbeit mit dem Kunden

„Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen“[42]

Die Zusammenarbeit mit dem Kunden steht im Mittelpunkt agiler Ansätze. Agile Vorgehensmodelle integrieren den Kunden daher in das Team und nehmen ihn so mit in die Verantwortung. Das Vertrauensverhältnis und die enge Zusammenarbeit sind wichtig. Nur der Kunde weiß, was er tatsächlich benötigt, der Kunde soll daher maßgeblich mitentscheiden, wann welche Funktionalitäten umgesetzt werden. Entwickler müssen daher eng mit dem Kunden zusammenarbeiten. Dieses Vorgehen verringert das Risiko einer niemals fertig gestellten oder den Bedürfnissen nicht entsprechenden Software[43].

2.3.1.4 Einstellen auf Änderungen

„Sich auf unbekannte Änderungen einzustellen ist wichtiger als einem zu folgen“[44]

Traditionelles Projektmanagement geht davon aus, dass die möglichst exakte Einhaltung eines Plans mit Erfolg gleichzusetzen ist und dieser wiederum mit Kundennutzen. Tatsächlich verläuft aber kaum ein Projekt nach Plan, und je granularer ein Plan ist, umso schneller weicht er von der Wirklichkeit ab. So entstehen Aufwände, die der Verfolgung des Projektziels nicht dienlich sind. Agile Verfahren verzichten nicht auf Pläne, planen aber nur soweit, wie es zweckmäßig ist.

[41] [Wallmüller2007], S. 25, [Bleek2008], S. 14.

[42] Übersetzung nach [Dogs], S. 36.

[43] [Wallmüller2007], S. 25, [Bleek2008], S. 14.

[44] Übersetzung nach [Dogs], S. 38.

Dabei sind – im Gegensatz zu herkömmlichen Entwicklungsprozessen – Änderungen an den Anforderungen immer willkommen.[45]

2.3.2 Agile Prinzipien

Die oben beschriebenen vier agilen Werte werden ergänzt durch zwölf agile Prinzipien, die ebenfalls in dem agilen Manifest aufgeführt sind. Ein agiles Prinzip beschreibt kein direkt umsetzbares Verfahren, sondern einen Grundsatz, den man auf Methoden, die einen Prozess bilden sollen, anwenden kann. Ein agiles Prinzip kann also als Leitsatz für die agile Arbeit aufgefasst werden. Die erwähnten zwölf agilen Prinzipien sind:

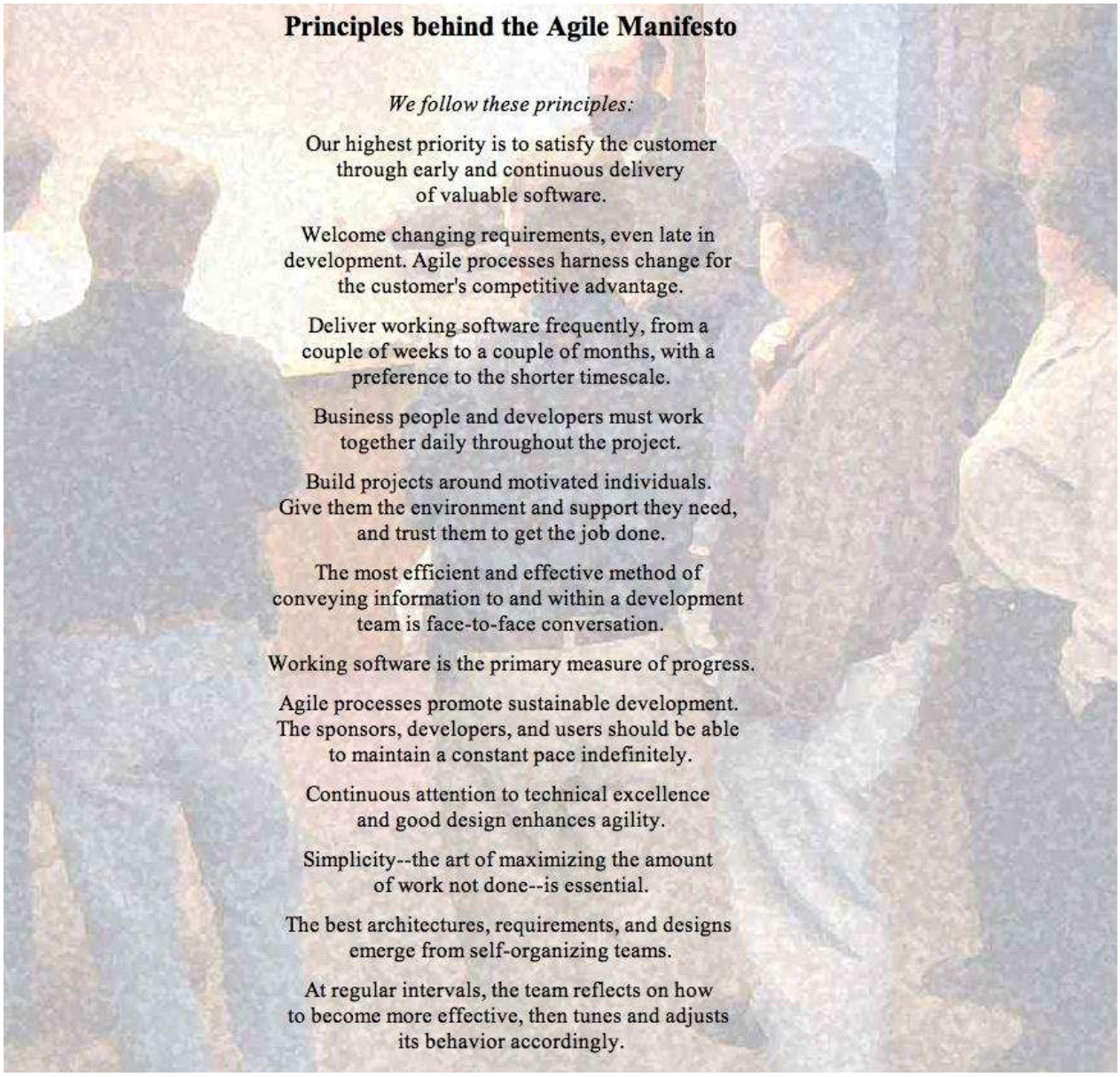

Abbildung 2: Die zwölf agilen Prinzipien[46]

[45] [Wallmüller2007], S. 26, [Bleek2008], S. 15.

Die Bedeutung dieser Prinzipien wird nun näher beleuchtet:

2.3.2.1 Die Bedürfnisse des Kunden haben die höchste Priorität

„Unsere höchste Priorität liegt darin, den Kunden durch frühzeitige und kontinuierliche Auslieferung von nützlicher Software zufriedenzustellen."

Dieses Prinzip bringt zum Ausdruck, dass es von zentraler Bedeutung ist, die Bedürfnisse des Kunden aufzunehmen und angemessen zu berücksichtigen. Das wichtigste Bedürfnis des Kunden ist es, funktionierende Software zu erhalten. Die Aktivitäten des Entwicklungsteams sollen dies berücksichtigen.[47]

2.3.2.2 Sich ändernde Anforderungen

„Begrüße sich ändernde Anforderungen, selbst wenn sie erst spät in der Entwicklung auftreten. Agile Prozesse machen Änderungen für Wettbewerbsvorteile des Kunden nutzbar."

Dieses Prinzip bringt zum Ausdruck, dass der Entwicklungsprozess davon ausgehen soll, dass sich die Anforderungen des Kunden ändern werden. Softwareprojekte sollen sich darauf einstellen. Änderungen sollen nicht als Problem angesehen werden, sondern als Wettbewerbsvorteil für den Kunden.[48]

2.3.2.3 Häufige Auslieferungen

„Liefere häufig funktionierende Software aus, innerhalb weniger Wochen oder Monate, wobei der kürzeren Zeitspanne der Vorzug gegeben werden soll"

Durch kurze Auslieferungszyklen hat der Kunde früher die Möglichkeit, Feedback an das Entwicklungsteam zurückliefern zu können. Daher ist es auch für das Entwicklungsteam leichter die Kontrolle über die Entwicklungsergebnisse zu behalten.[49]

2.3.2.4 Zusammenarbeit zwischen Benutzern und Entwicklern

„Benutzer und Entwickler müssen während des Projektes täglich zusammenarbeiten"

Dieses Prinzip bringt zum Ausdruck, dass nur eine enge Zusammenarbeit zwischen Benutzern und Entwicklern es erlaubt, Probleme und Missverständnisse frühzeitig zu erkennen und Maßnahmen zu deren Beseitigung zu treffen[50].

---

[46] http://agilemanifesto.org/principles.html, letzter Zugriff 26.04.2008

[47] [Dogs], S. 41.

[48] [Dogs], S. 42.

[49] [Dogs], S. 42.

2.3.2.5 Motivierte Mitarbeiter und Vertrauen

„Führe Deine Projekte mit motivierten Mitarbeitern durch. Gebe ihnen die benötigte Umgebung und Unterstützung und vertraue ihnen, dass sie ihre Arbeit erfolgreich durchführen."

Ein agiles Team soll die ihm übertragenen Aufgaben weitgehend selbsttätig lösen. Daher ist es wichtig, dass jedes Teammitglied sich in dem erforderlichen Umfang in das Team einbringt. Dazu ist es notwendig, dass das Management dem Team als Ganzem, und damit implizit allen Teammitgliedern, das dazu erforderliche Vertrauen entgegen bringt.[51]

2.3.2.6 Direkte Kommunikation

„Die effizienteste und effektivste Methode Informationen einem Entwicklungsteam zukommen zu lassen bzw. innerhalb des Teams auszutauschen, besteht in direkter Kommunikation durch persönlichen Kontakt."

Dieses Prinzip bringt zum Ausdruck, dass der Informationsfluss innerhalb eines Teams am Besten durch direkte Kommunikation erfolgt. Bei der direkten Kommunikation können Verständnisschwierigkeiten noch am einfachsten gelöst werden.[52]

2.3.2.7 Funktionierenden Software als Maßstab

„Funktionierende Software ist der wichtigste Maßstab des Fortschritts."

Der Kunde ist in erster Linie an funktionierender Software interessiert, umfangreiche Pläne und Dokumente sind nur von begrenztem Nutzen. Daher wird in agilen Entwicklungen der Fortschritt an funktionierender Software als Maßstab des Fortschritts genommen.[53]

2.3.2.8 Nachhaltiges Entwicklungstempo

„Agile Prozesse fördern die nachhaltige Entwicklung. Geldgeber, Entwickler und Anwender sollten in der Lage sein, beständiges Tempo dauerhaft beizubehalten."

Dieses Prinzip bringt zum Ausdruck, dass das Team innerhalb der regelmäßigen Arbeitszeit in der Lage sein soll, einen Entwicklungsfortschritt zu erzielen. Das Kunde oder das Management sollte das Team nicht auffordern Überstunden zu leisten, damit dem Team Zeit für die erforderliche Erholung zur Verfügung steht.[54]

---

[50] [Dogs], S. 43.

[51] [Dogs], S. 43.

[52] [Dogs], S. 43.

[53] [Dogs], S. 44.

[54] [Dogs], S. 44.

2.3.2.9 Aufmerksamkeit für technische Spitzenleistung

„Ständige Aufmerksamkeit für technische hervorragende Qualität und gutes Design erhöhen die Agilität."

Dieses Prinzip fordert, dass sich das Team und jedes Mitglied über technologische Fortschritte informiert, und diese Kenntnisse in die Entwicklung einbringt. Auf diese Art werden die Kenntnisse des Teams in Kundennutzen umgesetzt.[55]

2.3.2.10 Einfachheit

„Einfachheit – die Kunst, den Umfang der nicht erforderlichen Arbeit zu maximieren – ist wesentlich."

Dieses Prinzip fordert Einfachheit[56] 57. Dies meint hier einfache Lösungen insoweit, als unnütze, nicht zielführende, oder nicht angemessene Arbeit nicht durchgeführt werden soll. Es wird daher auf unfangreiche Anforderungsspezifikationen verzichtet, und es wird auch kein komplexes Design erstellt, wenn dies keinen Nutzen für den Kunden bringt.[58]

2.3.2.11 Selbstorganisierende Teams

Die besten Architekturen, Anforderungen und Entwürfe entstehen in selbstorganisierten Teams."

In diesem Prinzip wird gefordert, dass sich das Team selbst organisiert. Es ist kein Management erforderlich, das detailliert festlegt, wann die Teammitglieder welche Aufgabe durchführen. Die besten Lösungen entstehen, wenn das Team solche Entscheidungen selbst trifft.[59]

2.3.2.12 Regelmäßige Selbstreflexion

„Das Team reflektiert regelmäßig darüber, wie es effektiver wird und passt dann sein Verhalten entsprechend an."

Durch dieses Prinzip benutzt das Team einen empirischen Prozess für die Entwicklung. Das Team überprüft die eigenen Vorgehensweisen in regelmäßigen Abständen und korrigiert das

[55] [Dogs], S. 44.

[56] Dieses Prinzip ist im deutschen schwieriger als im Englischen zu formulieren. „simple" und „easy" werden im Deutschen beide Male mit „einfach" übersetzt.

[57] Dieses Prinzip ist im deutschen schwieriger als im Englischen zu formulieren. „simple" und „easy" werden im Deutschen beide Male mit „einfach" übersetzt.

[58] [Dogs], S. 44f.

[59] [Dogs], S. 45.

eigene Vorgehen, wenn es einen Bedarf dazu sieht. Auf diese Art kann das Team also laufend besser werden.[60]

2.3.3 Agile Vorgehensmodelle

In der Fachpresse findet häufig eine Vermischung zwischen abstrakten und konkreten Begriffen der Agilität statt. So wird häufig ein agiler Prozess in Ermangelung eines eigenen Namens einfach als "der Agile Prozess" bezeichnet, obwohl er nur ein möglicher agiler Prozess unter vielen ist. Gleiches gilt für Begriffe wie "Agile Methode", "Agiles Prinzip", "Agile Modellierung" usw. Ein Leser von Artikeln über agile Softwareentwicklung ist also stets gut beraten, zwischen allgemeinen, abstrakten Bestandteilen von Agilität und speziellen, konkreten Bestandteilen einer Umsetzung von Agilität zu unterscheiden. Unter einem agilen Vorgehensmodell wird hier ein auf Agilität ausgerichteter Softwareentwicklungsprozess verstanden, bei dem überwiegend agile Werte und Prinzipien zum Einsatz kommen.

Ein Ziel agiler Vorgehensmodelle ist es, den Softwareentwicklungsprozess durch Abbau der Bürokratisierung und durch die stärkere Berücksichtigung der menschlichen Aspekte effektiver und für alle Beteiligten vorteilhafter zu gestalten. Dazu bedient sich ein agiler Prozess hauptsächlich agiler Methoden. Um die vergleichsweise Einfachheit agiler Prozesse zu unterstreichen, spricht man oft von Agilen Methoden statt einem Agilen Prozess. Das unterstreicht die Einfachheit einiger Agiler Prozesse, die sich lediglich als Zusammenfassung einiger Agiler Praktiken verstehen.

Die meisten agilen Vorgehensmodelle versuchen weiter, die reine Entwurfsphase auf ein Mindestmaß zu reduzieren und im Entwicklungsprozess so früh wie möglich zu ausführbarer Software zu gelangen. Diese Software soll dann in regelmäßigen, kurzen Abständen dem Kunden zur gemeinsamen Abstimmung vorgelegt werden. Auf diese Weise soll es möglich sein, flexibler auf Kundenwünsche einzugehen und so die Kundenzufriedenheit insgesamt deutlich zu erhöhen. Sie stellen damit einen Gegensatz zu den klassischen Vorgehensmodellen wie dem V-Modell oder auch dem Rational Unified Process (RUP) dar[61].

Die klassischen Vorgehensmodelle der Softwareentwicklung kennen verschiedene Phasen bzw. Aspekte. Grob wird meist zwischen Modellierung und Entwicklung unterschieden, wobei sich die Modellierung in Analyse und Entwurf, sowie die Entwicklung in Test und Programmierung unterteilen lässt. Auch in agilen Vorgehensmodellen findet man eine Phasenstruktur, die aber deutlich gröber strukturiert ist. Häufig findet man eine Unterteilung in 3 Phasen, nämlich

[60] [Dogs], S. 45.

[61] [Versteegen2000], S. 52ff.

- eine Konzept- und Planungsphase, in der die Verfahren festgelegt werden und erste Anforderungen ermittelt werden,
- eine Entwicklungsphase, in der die eigentliche Entwicklung stattfindet und
- eine Abschlussphase, in der das Projekt abgeschlossen wird.[62]

In traditionellen Vorgehensmodellen findet man häufig eine Unterteilung zwischen Entwurf, Implementierung und Test. Auch in der Agilen Softwareentwicklung sind diese Aspekte von zentraler Bedeutung, sie werden allerdings nicht zeitlich über Phasen verteilt, sondern finden üblicherweise in jeder Iteration statt. Kent Beck spricht in diesem Zusammenhang von den vier Aktivitäten der Softwareentwicklung:

- Coding,
- Testing,
- Listening und
- Designing,

die in jeder Iteration ausgeführt werden, statt auf bestimmte Phasen verteilt zu werden.

Unter dem begrifflichen Dach der Agilen Entwicklung mit seinen im Agilen Manifest formulierten Leitgedanken und dreigeteiltem hierarchischen Aufbau (Werte, Prinzipien, Praktiken) finden sich zahlreiche Strömungen ein. Aus der – gewollten – Vielfalt agiler Ansätze kristallisieren sich bei genauerer Betrachtung sechs zentrale Punkte heraus, die in den meisten agilen Vorgehensmodellen umgesetzt werden[63]

- Bei Initiierung des Projekts sollen der generelle Umfang, Ziele und Risiken ermittelt und schriftlich dokumentiert werden.
- Die Entwicklung findet in kurzen, iterativen Entwicklungszyklen von fester Dauer statt, die kundenrelevante Teile beinhalten.
- Der Kunde wird in den Entwicklungsprozess einbezogen. Kundennutzen erfordert kontinuierliche Interaktion zwischen Kunden und Entwicklern.
- Kontinuierliches Feedback ist erforderlich, um auf dem richtigen Weg zu bleiben und sich zu verbessern.
- Ein technisch herausragendes Produkt zu erschaffen und aufrecht zu erhalten trägt maßgeblich zum jetzigen und zukünftigen Kundennutzen bei.

[62] [Beck2000], S. 1231ff und [Abrahamsson], S.29ff.

[63] [Highsmith], S.335ff.

Es ist umstritten, ob der Entstehungsprozess von Software so gut verstanden wird, dass eine planmäßige Herstellung möglich ist: Kritiker argumentieren, dass Software nichts anderes sei als "ausführbares Wissen". Wissen jedoch lässt sich nicht ingenieurmäßig herstellen, wie sich etwa eine Brücke oder ein Hochhaus herstellen lässt, sondern wird in einem kreativen Prozess gefunden.

Eine im März 2006 in den Vereinigten Staaten von Scott Ambler durchgeführte Umfrage[64] unter 4232 IT-Fachleuten lieferte als Ergebnis, dass eine Mehrheit der Unternehmen agile Praktiken erfolgreich einsetzt. Im Detail gelangte die Studie zu nachfolgenden Zahlen (vgl. Ambler 2006b): 65% der Befragten arbeiten in Unternehmen, die eine oder mehrere agile Praktiken einsetzen, 41% in Unternehmen, die eine oder mehrere agile Vorgehensmodelle einsetzen

- 60% berichten über eine Verbesserung der Produktivität
- 66% über eine Verbesserung der Qualität
- 58% über gestiegene Zufriedenheit auf Seiten der Stakeholder

Die Umfrageresultate lassen deutlich werden, dass sich agile Vorgehensmodelle in der Softwareentwicklung inzwischen eine breite Akzeptanzbasis erarbeitet haben. Zwar wird auch deutlich, dass Unternehmen eher dazu tendieren, einzelne agile Praktiken einzusetzen als den Sprung zu wagen, ein agiles Vorgehensmodell vollständig einzusetzen. Doch sollte dies nicht überraschen, da Umstrukturierungen oft inkrementell durchgeführt werden. Generell sind aber auch Anwender traditioneller Vorgehensmodelle gut beraten, agile Aspekte in ihre Vorgehensmodelle einzubringen, allein schon um die Änderungen, die ein Projekt zwangsläufig mit sich bringt, zu berücksichtigen und zu integrieren

[64] Dokumentiert in Scott W. Ambler: Agile Software Development: What's Really Going On, bcs-uc.e2bis.biz/rps/refmaterial/200705312027200.20070529_AgileDevelopment_**Scott**W**Ambler**.ppt. letzter Zugriff 02.05.2008

3 Beispiele agiler Vorgehensmodelle

Nachdem wir im vorherigen Kapitel die wesentlichen Eigenschaften agiler Vorgehensmodelle eher abstrakt beschrieben haben, werden in diesem Kapitel nun drei prominente Vertreter agiler Vorgehensmodelle (Scrum[65], XP[66] und FDD[67]) vorgestellt. Die ausführliche Darstellung jedes dieser Verfahren sprengt den Rahmen dieser Arbeit, die Vorstellung muss daher zwangsläufig kurz und unvollständig bleiben und konzentriert sich auf die zentralen Eigenschaften der jeweiligen Vorgehensmodelle.

3.1 Strukturelemente agiler Vorgehensmodelle

Dennoch bleibt die Frage, wie diese agilen Vorgehensmodelle beschrieben werden sollen. Eine Untersuchung der drei oben genannten Beispiele zeigt, dass die Vorstellungen, was ein agiles Vorgehensmodell im Einzelnen leisten soll, sich doch erheblich voneinander abweicht: Agile Vorgehensmodelle unterscheiden sich in ihrem fachlichen Umfang, in den eingesetzten Praktiken und in der benutzten Terminologie. Ist es unter diesen Voraussetzungen überhaupt möglich, einen strukturellen Rahmen zu finden, der es erlaubt, die drei agilen Vorgehensmodell Scrum, XP und FDD einerseits und die regulatorischen Anforderungen and die Softwareentwicklung andererseits angemessen abzubilden?

Eine einheitliche Beschreibung der drei Vorgehensmodelle ist wünschenswert. Derzeit gibt es noch keine einheitliche Notation für die Beschreibung und Modellierung von Vorgehensmodellen.[68] Für die Modellierung von Geschäftsprozessen haben die so genannten ereignisgesteuerten Prozessketten (EPK) zwar eine relativ weite Verbreitung gefunden, für die Modellierung von Prozessmodellen der Softwareentwicklung habe sie sich jedoch nicht durchgesetzt. Auch die bei

[65] Scrum wird etwa in [Pichler] oder [Schwaber] beschrieben. Beide Bücher beschreiben die Einsatz von Scrum, wobei [Pichler] auch auf den Einsatz in großen Projekten eingeht.

[66] XP wird in vielen Büchern beschrieben. Als Standardwerk gilt das Buch von Kent Beck ([Beck2000]), das nun in einer erweiterten neuen Auflage vorliegt ([Beck2004]). Mit vielen Erfahrungen aus dem deutschsprachigen Bereich wartet [Wolf2005] auf. Die Autoren haben langjährige Erfahrungen mit agilen Softwareprojekten. Eine kritische Sicht auf agile Praktiken findet man in [Stephens2003].

[67] Die Literatur zu FDD ist nicht so umfangreich, wie zu Scrum und XP. Erläuterungen zu FDD findet man in [Highsmith] und [Dogs].

[68] [Balzert2008], S. 451,.

der Softwareentwicklung häufig eingesetzte UML-Notation[69] ist dafür nicht optimal geeignet. In den letzten Jahren hat die OMG durch SPEM[70] ein Metamodell entwickelt, durch das beliebige Software-Entwicklungsmodelle abgebildet werden können. Momentan hat dieses Metamodell allerdings eine eher geringe Bedeutung[71]. Ob dies Metamodell auch für agile Vorgehensmodelle geeignet ist, ist noch wenig untersucht.[72] Zudem stellt sich die Frage, ob der Einsatz solcher Methoden zur Darstellung von agilen Vorgehensmodellen nicht überzogen ist und mehr Formalien in die Prozessbeschreibung einführt, als dann tatsächlich in dem Vorgehensmodell zu finden ist.

Daher wird in dieser Arbeit ein pragmatischer Ansatz zur Beschreibung von Vorgehensmodelle gewählt, in dem zunächst Strukturierungselemente in den Vorgehensmodellen gesucht werden, und dann eine Darstellung auf der Basis dieser Elemente gewählt wird. In der Literatur konnten die folgenden Strukturierungselemente gefunden werden, auch wenn diese teilweise unterschiedlich benutzt wurden.[73]

- **Rollen**: Rollen beschreiben die Qualifikationen und Befugnisse von Beteiligten am Entwicklungsprozess.

- **Phasen**: Eine **Phase** ist ein zeitlicher Gliederungsabschnitt eines Projektes. Eine Phase fasst eine Menge von Aktivitäten und zugehörigen Ergebnissen zu einer Planungs- und Kontrolleinheit zusammen. Am Ende einer Phase steht üblicherweise ein Meilenstein, durch den sichergestellt wird, das die Bedingungen für den Übergang in die nachfolgenden Phase erfüllt sind.[74]

- **Disziplinen** erlauben eine inhaltliche Gliederung der Entwicklungsaktivitäten. Disziplinen findet man sowohl in klassischen Vorgehensmodellen[75], als auch in moderneren Vorgehensmodellen, wie dem V-Modell 97[76] oder RUP[77], auch wenn sie dort teilweise anders

[69] www.omg.org enthält umfangreiche Literatur zu der Unified Modelling Language (UML).

[70] Software Process Engineering Metamodell, V 2.0 vom 04.01.2008, http://www.omg.org/technology/documents/formal/spem.htm, letzter Zugriff 09.05.2008.

[71] [Blazert2008], S. 452.

[72] SPEM ist durch IBM wesentlich beeinflusst worden. Der Fokus liegt auf schwergewichtigeren Vorgehensmodellen wie RUP oder OpenUP. Agile Vorgehensmodelle wie XP oder Scrum, werden dort eher nicht diskutiert.

[73] [Beck2004], [Schwaber], [Pichler] und viele Andere.

[74] [Oestereich2007], S. 19.

[75] In der Literatur findet man mehrere solcher klassischen Modelle, wie das Wasserfallmodell, das allgemeine V-Modell oder das Spiralmodell, vgl. [Pomberger], Kapitel 2, S. 11ff.

[76] [Dröschel], Einführung in das V-Modell Version 97, S. 57ff.

genannt werden. Disziplinen benötigen üblicherweise eine Anzahl von Eingaben, bestehen aus einem – mehr oder weniger komplexen Ablauf, bei dem eine Anzahl von Aktivitäten ausgeführt wird, und erzeugen oder verändern eine Anzahl von Produkten. Es ist also damit klar, dass jede Disziplin als Prozess aufgefasst werden kann.

- **Aktivitäten** beschreiben, was in einem Projekt zu tun ist und legen ggf. fest, welche Voraussetzungen, Ergebnisse und Abhängigkeiten damit verbunden sind. Die Granularität der Aktivitäten ist damit deutlich geringer als die von Phasen und Disziplinen. Aktivitäten werden benutzt, um die Disziplinen weiter zu strukturieren. Ein Beispiel einer Aktivität ist eine Produktverifizierung oder eine Risikokontrollmaßnahme. Eine bestimmte Aktivität kann prinzipiell in verschiedenen Disziplinen verwendet werden. Auch eine Aktivität verwendet Eingaben und erzeugt Ergebnisse, kann also ebenfalls als (Teil-) Prozess aufgefasst werden.

- **Artefakte**: Artefakte beschreiben die grundlegenden Eigenschaften der entstehenden Resultate aus den unterschiedlichen Phasen, Disziplinen und Aktivitäten.
- **Praktiken**: Eine agile Praktik ist eine etablierte Handlungsweise, die beschreibt, wie in einem ausgewählten Ausschnitt oder Aspekt der Softwareentwicklung agil vorzugehen ist oder die agilen Werte berücksichtigt werden sollen. Praktiken sind also eine Mischung aus Aktivitäten (Was ist zu tun?) und Methoden (Wie ist etwas zu tun?). Ein Praktik ist daher vielfach eine komplexe Kombination von Aktivitäten und Methoden, die auch Disziplinübergreifend sein kann.
- Unter einem **agilen Vorgehensmodell** verstehen wir dann eine konkrete Zusammenstellung von Phasen, Rollen, Artefakten und agilen Praktiken.

Für die Darstellung der drei Beispielvorgehensmodelle werden die vier Strukturelemente Rolle, Phase, Artefakt und Disziplin gewählt. Ein Dekomposition der Praktiken unterbleibt. Es ist möglich, dass diese vier Strukturierungselemente[78] den agilen Vorgehensmodellen nicht in allen Einzelheiten gerecht werden. Dies wird in aber an dieser Stelle in Kauf genommen, es in diesem Kapitel lediglich um eine Herausarbeitung der wesentlichen Kennzeichen der betreffenden Vorgehensmodelle geht.

[77] [Versteegen2], Kapitel 3.4, Die Phasen und Workflows des Rational Unified Process, S. 52ff.

[78] Also Phasen, Praktiken, Artefakte und Rollen.

3.2 Scrum

Scrum ist ein agiles Vorgehensmodell, das in den frühen 1990ern von Ken Schwaber und Jeff Sutherland entwickelt wurde[79]. Scrum hat den Anspruch, sich prinzipiell auf Softwareprojekte unterschiedlicher Größe anwenden zu lassen. Scrum legt jedoch lediglich Rahmenbedingungen für das Projektmanagement und den Umgang mit Anforderungen fest, definiert aber im Gegensatz zu anderen agilen Vorgehensmodellen keine konkreten Praktiken der Softwareentwicklung. Die Festlegung der für die Softwareentwicklung erforderlichen Methoden und Werkzeuge werden dem Team überlassen.

Scrum versteht sich selbst als empirischer Prozess, d.h. Arbeitsweise und Produkt werden regelmäßig begutachtet und gegebenenfalls angepasst. Am Ende einer jeden Iteration, in Scrum Sprint genannt, beurteilt der Product Owner die Angemessenheit der erzielten Ergebnisse, und das Team reflektiert über seine Zusammenarbeit und die Anwendung des Prozesses und der eingesetzten Techniken. So lernt das Projekt von Sprint zu Sprint dazu und kann sich kontinuierlich verbessern.

Scrum enthält keine konkreten Verfahrensanweisungen oder Arbeitsanweisungen, wann was zu tun ist, sondern überlässt solche Entscheidungen dem Team. Soweit diese als hilfreich erachtet werden, muss das Team solche Vorgaben selbst erarbeiten. Scrum erklärt wie folgt: Softwareentwicklungsprojekte weisen häufig nicht deshalb eine suboptimale Arbeitsorganisation auf, weil Management und Mitarbeiter nicht guten Willens sind. Das zentrale Problem traditioneller Vorgehensweisen besteht vielmehr darin, dass frühzeitig durch einen Plan versucht wird, alle Eventualitäten und Arbeitsdetails zu antizipieren und einzuplanen. Andererseits erfolgt die Rückmeldung über den tatsächlichen Fortschritt und die aufgetretenen Probleme sehr spät, meist erst dann, wenn die Software integriert und getestet wird. In Scrum werden daher alle Softwareentwicklungsaktivitäten prinzipiell innerhalb einer Iteration ausgeführt. So erhält das Team bereits nach wenigen Wochen Rückmeldung über den Fortschritt und etwaige Probleme und Hindernisse. Die Projektplanung kann dann auf dem tatsächlichen Fortschritt des Projekts aufsetzen.

3.2.1 Rollen

Scrum kommt mit drei Rollen aus. Dies sind der Product Owner, Der Scrum Master und das Team.

[79] [Sutherland2004].

3.2.1.1 Product Owner:

Der Product Owner nimmt eine zentrale Rolle ein: Er ist für das Erfassen der Kundenanforderungen und die Dokumentation dieser Anforderungen im sogenannten „Product Backlog“ verantwortlich. Dabei arbeitet er intensiv mit allen Stakeholdern, wie Kunden und Anwendern, zusammen.

Der Product Owner ist weiterhin für das Erreichen der Projektziele und damit für den Projekterfolg verantwortlich. Er entscheidet über den Auslieferungszeitpunkt und die erforderliche Funktionalität. Daher liegt auch die Erstellung des Releaseplans in seiner Zuständigkeit.

Der Product Owner arbeitet über die gesamte Projektdauer intensiv mit dem Team zusammen. Er unterstützt das Team beim Verstehen und Umsetzen der Anforderungen, indem er das Team mit detaillierten Informationen versorgt und die entstandenen Arbeitsergebnisse prüft und abnimmt[80].

3.2.1.2 Team

Das Team führt alle Arbeiten aus, die zur Umsetzung der Anforderungen in auslieferbare Produktinkremente erforderlich sind. Das Team ist interdisziplinär besetzt, besteht also aus allen erforderlichen Experten, wie Programmierern, Architekten und Testern.

Im Gegensatz zu einem konventionellen Team ist ein Scrum-Team aber bevollmächtigt[81] selbstständig zu entscheiden, wie viele Anforderungen in der nächsten Iteration umgesetzt werden. Somit legt es selbst fest, wie viel Arbeit es bewerkstelligen kann und wie die Organisation dieser Arbeit erfolgt: Das Scrum-Team organisiert sich selbst. Es sollte aus nicht mehr als 7 Personen bestehen.

Damit ein Team effektiv zusammenarbeitet, ist es erforderlich, dass es sich auf gemeinsame Normen einigt. Viele dieser Normen werden typischerweise von dem Team selbst formuliert und ggf. auch während des Projektes geändert, wenn sich herausstellt, dass sie kontraproduktiv sind. Andererseits ist aber auch denkbar, dass bestimmte Vorgaben, wie unternehmensweite Programmierrichtlinien, bereits zu Projektstart festgelegt werden[82].

[80] [Pichler], Kapitel 3.1, S. 9ff.

[81] Im Original wird von „empowered“ gesprochen.

[82] [Pichler], Kapitel 3.2, S. 13ff.

3.2.1.3 ScrumMaster

Die Aufgabe des ScrumMasters besteht darin, dem Team bei dem richtigen Einsatz von Scrum zu unterstützen und gegebenenfalls auftauchende Probleme aus dem Weg zu räumen. Er unterstützt das Team weiterhin bei der Teambildung. Er fungiert daher überwiegend als Moderator und Coach. Er hat aber weder eine aktive Rolle bei der Festlegung der Anforderungen noch bei der Umsetzung der Anforderungen[83].

3.2.2 Phasen

Scrum führt die Softwarenentwicklung in drei Phasen aus[84].

3.2.2.1 Pre-Game

Diese Phase besteht aus zwei Teilphasen. Die Planning-Phase hat den Zweck, eine Vision für das Projekt zu entwickeln und die Erwartungen an das Projekt festzusetzen. In der nachfolgenden Staging-Phase werden die ersten Anforderungen ermittelt, priorisiert, und der notwendige Aufwand für die Umsetzung grob geschätzt. Die Anforderungen werden in das sogenannte „Product Backlog“ eingestellt. Falls möglich, wird eine erste konzeptionelle Architektur für das Anwendungssystem erstellt, die als Basis der Entwicklung dient[85].

3.2.2.2 Development

In dieser Phase, auch als Game-Phase bezeichnet, wird das Softwaresystem durch eine Serie von Sprints umgesetzt. Diese Phase wird von Scrum als Black-Box betrachtet. Statt die verschiedenen Umstände bereits zu Beginn des Projektes in die Planung einzubeziehen, wird in einem Scrum-Projekt darauf geachtet, flexibel auf Änderungen zu reagieren und die erforderlichen Anpassungen vorzunehmen. Daher erfolgt die Entwicklung in Sprints, die erst zu Beginn eines jeden Sprints detailliert geplant werden. Jeder Sprint enthält alle erforderlichen Entwicklungsaktivitäten, wie Design, Implementierung und Test[86].

3.2.2.3 Post-Game

In dieser Phase wird das Projekt abgeschlossen. Die Phase wird begonnen, wenn der Product Owner zu dem Entschluss kommt, dass alle erforderlichen Anforderungen umgesetzt wurden.

[83] [Pichler], S. 19ff.

[84] [Larman2004], S. 113.

[85] [Abrahamsson], S. 29.

Das Projekt kann nun abgeschlossen werden, und noch ausstehende Aufgaben, wie Dokumentation und Benutzerschulung können durchgeführt werden[87].

3.2.3 Praktiken

Die von Scrum definierten Praktiken beziehen sich vor allem auf das Projekt- und Anforderungsmanagement. Spezifische Praktiken für die Softwareentwicklung, wie sie etwa XP definiert, fehlen vollständig. Es ist die Aufgabe des Teams, diese Techniken zu definieren.

3.2.3.1 Sprint

In Scrum finden alle Aktivitäten zur Umsetzung der Anforderungen in Iterationen statt, die Sprints genannt werden. Jeder Sprint wandelt Anforderungen in Produktinkremente um. Dazu beginnt jeder Sprint mit einer Planungssitzung, in der das Team festlegt, welche und wie viele Anforderungen in dem nachfolgenden Sprint umgesetzt werden können. Die Anforderungen sind für den nun beginnenden Sprint festgelegt, und können während des gesamten Sprints auch nicht geändert werden. Das Team führt nun die zur Umsetzung notwendigen Aktivitäten aus. Der Sprint wird durch ein Sprint-Review und eine Sprint-Retrospektive abgeschlossen. Im Sprint-Review wird dem Product Owner die neu erstellte Funktionalität präsentiert. Die Sprint-Retrospektive hat die Aufgabe, den Ablauf des letzten Sprints kritisch zu beleuchten und Verbesserungspotential zu identifizieren[88]. Ein Sprint sollte 30 Tage dauern.

[86] [Abrahamsson], S.29f.

[87] [Abrahamsson], S. 30.

[88] [Pichler], S. 81ff.

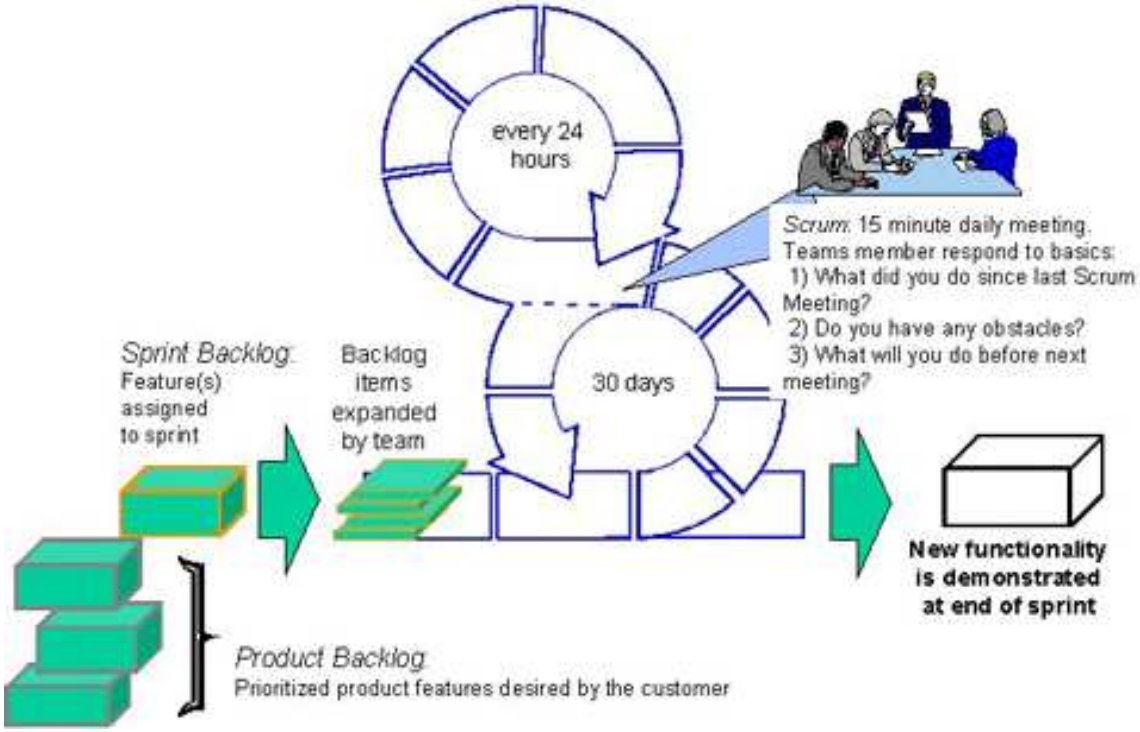

Abbildung 3: Der Scrum Sprint im Überblick[89]

Scrum kennt noch zwei weitere Arten von Sprints für besondere Gelegenheiten:

- **Explorationssprints**: Ziel eines Explorationssprints ist es, mit Hilfe von organisierten Experimenten das zur Umsetzung von Anforderungen erforderliche Wissen zu erlangen und damit die Umsetzungsrisiken zu verringern. Dies kann sich sowohl auf das Auffinden und Überprüfen von Entwurfsoptionen, die Überprüfung von Anforderungen an die Benutzerschnittstelle oder Ähnliches beziehen. Der wesentliche Unterschied zu normalen Iterationen besteht darin, dass kein Produktivcode entsteht. Wie viele Explorationssprints erforderlich sind, liegt an der Unsicherheit und den Risiken, die in dem Projekt bestehen.[90]
- **Releasesprint**: Manchmal ist es schwierig, alle erforderlichen Deploymentaktivitäten in jedem Sprint vorzunehmen, und so dem Product Owner am Ende eines jeden Sprint zu erlauben, die Software freizugeben und auszuliefern. Dies kann etwa dann der Fall sein, wenn eine kundenspezifische Konfiguration der Software erforderlich ist. In diesem Fall kann es zweckmäßig sein, nach mehreren Standardsprints einen Releasesprint explizit einzuplanen. In diesem Releasesprint werden dann alle auslieferungsbezogen Aktivitä-

[89] Aabhas. V. Paliwal, ArLinda A. Carroll, Scrum, www.utdallas.edu/~Kcooper/teaching/6354/6354spring06/adapted_SCRUM.ppt, letzter Zugriff 01.05.2008

[90] [Pichler], S.55f.

ten ausgeführt. Es wird jedoch empfohlen, auf Releasesprints zu verzichten, und die Auslieferungsaktivitäten in die Standardsprints zu integrieren[91].

3.2.3.2 Sprint Planning Meeting

Das Sprint Planning Meeting dauert maximal 8 Stunden und setzt sich aus zwei Teilen zusammen, die jeweils maximal 4 Stunden dauern. Teilnehmer der Besprechung sind der Product Owner, das gesamte Team und der Scrum Master.

Zunächst analysiert der Product Owner zusammen mit dem Team das Product Backlog, um die Anforderungen festzulegen, die in dem nächsten Sprint umgesetzt werden. Dabei ist die Festlegung der auswählbaren Anforderungen letztendlich in der alleinigen Verantwortung des Product Owners. Die Entscheidung, wie viele dieser Anforderungen tatsächlich im nächsten Sprint umsetzbar sind, liegt andererseits in der alleinigen Verantwortung des Teams[92].

Im zweiten Teil des Sprint Planning Meetings wird das Team nun planen, wie die Anforderungen in Produktinkremente umgesetzt werden. Daher werden die einzelnen Anforderungen in Aufgaben heruntergebrochen, in eine Reihenfolge gebracht und Teammitgliedern zugewiesen. Diese Tätigkeit liegt in der alleinigen Verantwortung des Teams. Der Product Owner hat für diesen Teil der Besprechung lediglich eine beratende Funktion. Das Ergebnis dieser Besprechung ist das Sprint Backlog. Das Sprint Backlog muss nicht zwangsläufig vollständig sein, aber genügend Informationen enthalten, um entscheiden zu können, ob die anstehenden Aufgaben im nächsten Sprint umsetzbar sind.

Die Besprechung endet mit der gemeinsamen Verpflichtung des Teams – dem Commitment - die soeben festgelegten Anforderungen innerhalb des Sprints in Produktinkremente umzusetzen[93].

3.2.3.3 Daily Standup Meeting

Das täglich Standup Meeting hat eine Dauer von 15 Minuten. Es soll jeden Tag am selben Ort und zur selben Zeit stattfinden. Es ist erforderlich, dass alle Teammitglieder teilnehmen. Es wird von dem Scrum Master geleitet. Zweck dieser kurzen Besprechung ist es, dass jedes Teammitglied einen kurzen Statusbericht an das Team gibt, alle Teammitglieder über existierende Probleme unterrichtet werden, und ggf. Maßnahmen zur Problembeseitigung eingeleitet werden.

[91] [Pichler], S. 57.

[92] [Schwaber], S. 136.

[93] [Schwaber], S. 136f.

3.2.3.4 Sprint Review

Das Sprint Review Meeting findet immer im Anschluss an einen Sprint statt. Es hat den Zweck, die in dem abgelaufenen Sprint fertig gestellten Funktionalitäten dem Product Owner vorzustellen und Fragen zu diesen Funktionalitäten zu beantworten. Soweit zweckmäßig, können weitere Stakeholder zu dieser Besprechung eingeladen werden. Obwohl die Bedeutung von „fertig“ von Projekt zu Projekt variieren kann, ist normalerweise damit gemeint, dass eine Funktionalität fertig entwickelt und getestet ist, und damit prinzipiell ausgeliefert werden kann Der Product Owner wird dann mit dem Team und den Stakeholdern an Hand des Feedbacks mögliche Änderungen am Product Backlog besprechen. Das Sprint Review Meeting soll nicht mehr als vier Stunden dauern[94].

3.2.3.5 Sprint Retrospective

Sprint-Retrospektive findet immer unmittelbar im Anschluss an das Sprint Review statt. Die Sprint-Retrospektive hat die Aufgabe, den Ablauf des letzten Sprints kritisch zu beleuchten und Verbesserungspotential zu identifizieren. An der Sprint-Retrospektive nehmen das Team, der Product Owner und der Scrum Master teil. Den Teammitgliedern werden zwei Fragen gestellt:

- Was lief während des letzten Sprints gut?
- Was kann im nächsten Sprint verbessert werden?

Das Team legt dann mögliche Verbesserungen fest und priorisiert sie.

3.2.4 Artefakte

Scrum als agiles Vorgehensmodell benötigt nur wenige Artefakte. Dies sind das Product Backlog mit den erforderlichen Anforderungen, der Releaseplan mit Anforderungen für das nächste Release, der Release Burndown Chart, der die Informationen über den Fertigstellungsgrad des nächsten Release enthält und das Sprint Backlog, das die Aufgaben für die aktuelle Iteration enthält.

3.2.4.1 Product Backlog

Das Product Backlog enthält alle priorisierten Anforderungen des zu erstellenden Anwendungssystems und sollte bereits zu Beginn soweit wie möglich aufgefüllt werden. Das Backlog enthält außer den funktionalen Anforderungen alle nicht-funktionalen Anforderungen, wie Skalierbarkeit, Robustheit, Performanz und Benutzbarkeit. Das Product Backlog kann jederzeit vom Product

[94] [SChwaber], S. 139f.

Owner ergänzt werden. Zudem können noch nicht umgesetzte Anforderungen jederzeit durch den Product Owner geändert werden. Die Anforderungen im Backlog sind stets priorisiert. Kriterien für die Priorisierung können Nutzen, Risiko oder Kosten sein. Für die Priorisierung ist ebenfalls der Product Owner zuständig[95].

3.2.4.2 Releaseplan

Die Planung einer Softwareversion wird im Releaseplan festgehalten. Der Releaseplan enthält im Wesentlichen die Anzahl der benötigten Iterationen und, soweit möglich, eine angenommene Reihenfolge der Umsetzung von Anforderungen. Der Releaseplan ist grobgranular und hat den Charakter einer Vorhersage. Der Product Owner ist für die Erstellung und Aktualisierung des Releaseplans zuständig.[96]

3.2.4.3 Sprint Backlog

Das Sprint Backlog enthält alle Aktivitäten, die erforderlich sind, um das Iterationsziel zu erreichen. Die Aktivitäten sind in Personenstunden geschätzt. Das Sprint Backlog wird zu Beginn einer Iteration erstellt, und wird spätestens an jedem Arbeitstag aktualisiert. Damit erlaubt es eine genau Verfolgung der im Sprint durchgeführten Aktivitäten. Das Sprint Backlog ist auch die Basis für den Sprint-Burndown-Bericht.

3.2.4.4 Sprint Burndown Chart

Der Sprint-Burndown-Chart zeigt den Fortschritt im Sprint auf. Er betrachtet, wie sich die Aufwände im Sprint Backlog von Tag zu Tag entwickeln und zeigt diese Entwicklung grafisch in einem Chart. Der Bericht ermöglicht es, den Fortschritt innerhalb des Sprints zu verstehen und zu kommunizieren. Anhand des Berichtes lässt sich der Sprint-Verlauf gut nachvollziehen und Verbesserungspotenzial identifizieren. Der Bericht kann daher auch für die Sprint-Retrospektive verwenden werden[97].

3.2.4.5 New Product increment

Innerhalb eines Sprints setzt das Team die im Sprint Backlog aufgeführten Anforderungen um. Welche Artefakte dabei im Einzelnen entstehen, wird durch Scrum nicht thematisiert. Insbesondere sagt Scrum nichts über die Erstellung von Design- und Architekturdokumenten oder Doku-

[95] [Pichler], S. 25ff.

[96] [Pichler], S. 51.

[97] [Pichler], S.117ff.

menten im Allgemeinen. Daher ist es durchaus möglich solche Dokumente zu erstellen, wenn es eine Notwendigkeit dafür gibt.

3.3 Extreme Programming (XP)

Extreme Programming (XP) ist vermutlich das populärste und bekannteste agile Vorgehensmodell. XP wurde von Kent Beck und Ward Cunningham definiert und bekannt gemacht. Der Sprung von informellen Vorgehensweisen zu einer eigenen Methode geschah im Frühling 1996: Kent wurde für ein Review eines Abrechnungsprojektes bei Chrysler zugezogen, welches historisch gesehen als das erste offizielle XP Projekt angesehen wird.[98]

XP ist als ein agiler Softwareentwicklungsprozess für kleine Teams konzipiert. Der Prozess ermöglicht, während der Entwicklung der Software auf vage und sich rasch ändernde Anforderungen angemessen zu reagieren. XP basiert auf vier Grundwerten: Kommunikation, Feedback, Einfachheit und Mut. Der Rest ist eine Ansammlung von meist bekannten und mehrfach verwendeten Praktiken, die nach Ansicht von Beck besonders für die agile Softwareentwicklung geeignet sind: "[...] none of the ideas in XP are new. Most are as old as programming. There is a sense in which XP is conservative - all its techniques have been proven […]".[99]

Einer der Schwerpunkte bei XP liegt auf Tests, oft unter dem Schlagwort *Testgetriebene Entwicklung* zusammengefasst. Auch wenn sämtliche Vorgehensmodelle Tests erwähnen, sticht XP dadurch heraus, dass es Tests zur Grundlage des Entwicklungsprozesses macht. Jeder Entwickler entwickelt simultan Produktions- und Testcode; die Tests werden kontinuierlich integriert und bilden so eine überaus stabile Grundlage für die zukünftige Entwicklung. XP definierte in der ersten Version vier, heute fünf Werte (Kommunikation, Feedback, Einfachheit, Mut, Respekt) und 12 Prinzipien[100]. Die Praktiken werden in vier Kategorien unterteilt: Praktiken für das gesamte Team, die Kunden, die Entwickler und das Management. Nur wenn alle diese Techniken in Projekten angewendet werden, kann davon gesprochen werden, das XP gelebt wird.[101]

“Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation”[102]

---

[98] [Stephens], S. 50ff.

[99] [cLab]

[100] Die Anzahl der Praktiken wurde in [Beck2004] auf 24 erhöht.

[101] In [Beck2004].

[102] [Jeffries01]

3.3.1 Rollen

Im XP-Vorgehensmodell sind die Rollen Kunde und Entwickler besonders wichtig. Es gibt aber noch weitere, die bedeutsam sind.

3.3.1.1 Kunde

Der Kunde spezifiziert durch sogenannte Story Cards die Funktionalität des zu entwickelnden Systems. Zudem nimmt er am Ende einer Iteration die Iterationsergebnisse der Entwickler ab. Durch Akzeptanztests wird geprüft, ob die Ergebnisse den Anforderungen entsprechen. Weiterhin steht er den Entwicklern zur Verfügung, um offen fachliche Fragen gemeinsam mit ihnen zu klären.[103]

3.3.1.2 Entwickler

Die Entwickler haben die Aufgabe, die Anforderungen umzusetzen. Dazu wird es üblicherweise erforderlich sein, die erforderlichen Funktionalitäten zu entwerfen, zu implementieren und testen. Diese Aktivitäten sind nicht weiter unterteilt, sondern dieser Zyklus wird immer wieder für jede Funktionalität separat durchgeführt. Damit der Entwickler die korrekte Funktionalität entwickeln kann, muss jede einzelne User Story korrekt verstanden sein. Daher ist eine regelmäßige und enge Zusammenarbeit mit dem Kunden zwingend erforderlich.[104]

3.3.1.3 Tester

Der Kunde soll die am Ende der Iteration gelieferte Funktionalität testen. Dafür sind Akzeptanztests erforderlich, für die der Kunde verantwortlich ist. Da der Kunde aber meistens nicht über die technischen Fähigkeiten zum Erstellen dieser Tests verfügt, wird er durch einen oder mehrere Tester unterstützt.

3.3.1.4 Tracker

Der Tracker ist für das Messen des Projektfortschritts zuständig. Dazu ermittelt er Metriken und gibt Fortschrittsberichte an Entwickler, Kunde und Management.[105]

[103] [Wolf2005], S. 163ff.

[104] [Wolf2005], S. 167f.

[105] [Wolf2005], S. 169f.

3.3.1.5 Coach

Die Aufgabe des Coaches ist es, ein Bewusstsein für den XP-Prozess bei den Teammitgliedern zu entwickeln, den Prozess gegebenenfalls an die Bedürfnisse anzupassen und ganz allgemein das Team zu unterrichten und bei Besprechungen zu moderieren.[106]

3.3.1.6 Consultant

Dabei handelt es sich um ein optionales Team-Mitglied, das hinzugezogen wird, wenn das Team Know-How für Verfahren oder Werkzeuge benötigt, über die es derzeit aber noch nicht verfügt.[107]

3.3.2 Phasen

Der Lebenszyklus eines XP-Projektes ist in mehrere Phasen unterteilt. Die untenstehende Abbildung gibt einen Überblick:

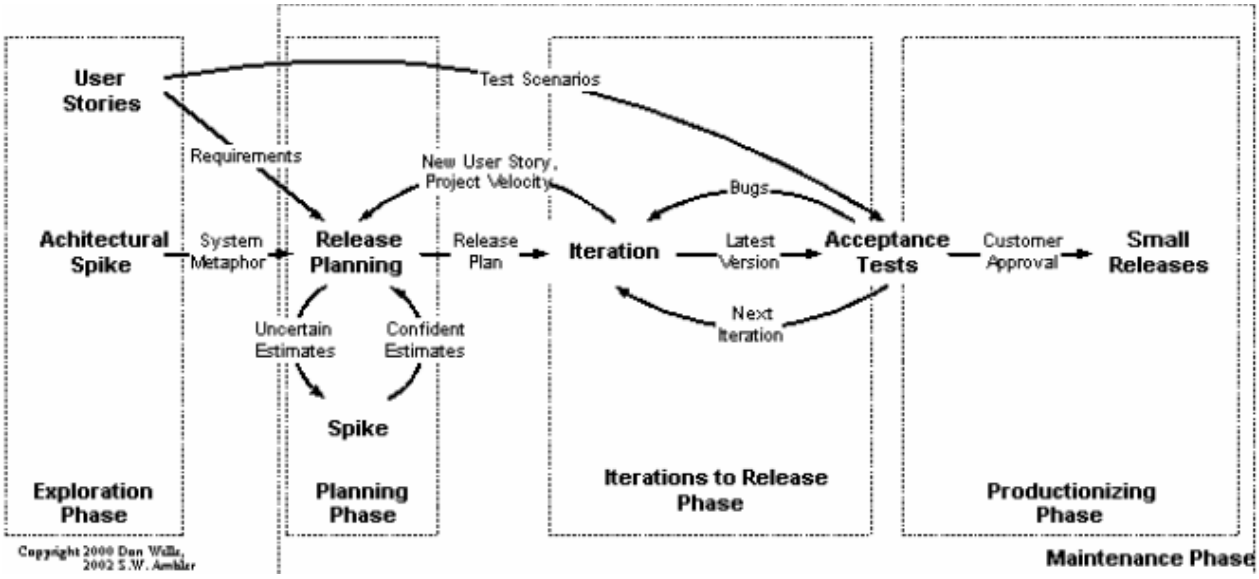

Abbildung 4: Der XP-Lebenszyklus im Überblick[108]

3.3.2.1 Exploration

Ein XP-Projekt beginnt üblicherweise mit der Explorationsphase. In dieser Phase wird das fachliche Umfeld erkundet, und die zentralen fachliche Konzepte identifiziert und festgelegt. Außer-

[106] [Wolf2005], S. 171f.

[107] [Wolf2005], S. 172f.

[108] nach: Markus Boos, Ingo Stilz, FH Mannheim, Fakultät Informatik, Seminar MSI – Extreme Programming, http://www.informatik.hs-mannheim.de/~knauber/MSc-MSI-05/A-02.pdf, letzter Zugriff 01.05.2008

dem werden die wesentlichen Risiken, wie sie z. B. durch den Einsatz neuer Technologien betrachtet. Weiterhin wird üblicherweise eine erste Grobschätzung für das Projekt hinsichtlich Aufwand und Dauer und Ressourcen erstellt. Die Explorationsphase hat also alle Aspekte einer typischen Konzeptphase, wie sie auch sonst vor dem eigentlichen Projektbeginn durchgeführt wird. Zudem wird das Projektteam zusammengestellt, und mit dem Vorgehen innerhalb des Projektes vertraut gemacht. Die Explorationsphase soll mehrer Wochen bis wenige Monate nicht überschreiten[109].

3.3.2.2 Planung

Sobald alle Projektbeteiligten sich in das Vorgehensmodell eingearbeitet haben, beginnt die Planungsphase für das erste Release. In der Planungsphase entscheiden Kunde und Entwickler gemeinsam durch das Planning Game, welche Funktionalität im nächsten Release enthalten sein soll und wann diese Funktionalitäten vom Entwicklungsteam implementiert werden sollen. Die Definition der umzusetzenden Funktionalität erfolgt dabei durch sogenannte Story Cards.[110]

3.3.2.3 Entwicklung

Nachdem die – erste – Release- und Iterationsplanung abgeschlossen ist, beginnt die Entwicklungsphase mit der ersten Iteration. Nach einer vorher festgelegten Dauer von einer bis vier Wochen ist die Iteration abgeschlossen. Das Ergebnis ist eine funktionierende Software, mit den für diese Iteration festgelegten Funktionalitäten. Nach Abschluss der Iteration führt der Kunde Akzeptanztests an der übergebenen Software durch. Dabei wird er häufig von einem Tester unterstützt.

Je nach Iterationsplanung wird nun entweder eine neue Iteration begonnen, oder die Entwicklungsphase wird durch die Freigabe und Übergabe an den Betrieb beendet. Häufig führt man vor der Übergabe an den Betrieb noch eine Überprüfung auf etwaige Mängel durch.[111]

3.3.2.4 Wartung

Nach der - ersten - Freigabe wechselt das Projekt in die Wartungsphase. Innerhalb dieser Phase wird das Anwendungssystem an geänderte Bedingungen angepasst und neue Anforderungen werden festgelegt und umgesetzt.

[109] [Beck2000], S. 131ff, [Wolf2005], S. 227ff.

[110] s. Kapitel 3.3.3.2.

[111] [Beck2000], S. 133f.

In der Wartungsphase werden prinzipiell die gleichen Aktivitäten ausgeführt, wie in der Planungsphase und der Entwicklungsphase. Allerdings herrschen in der Wartungsphase erschwerte Bedingungen, da das System bereits produktiv genutzt wird.[112]

3.3.3 Praktiken

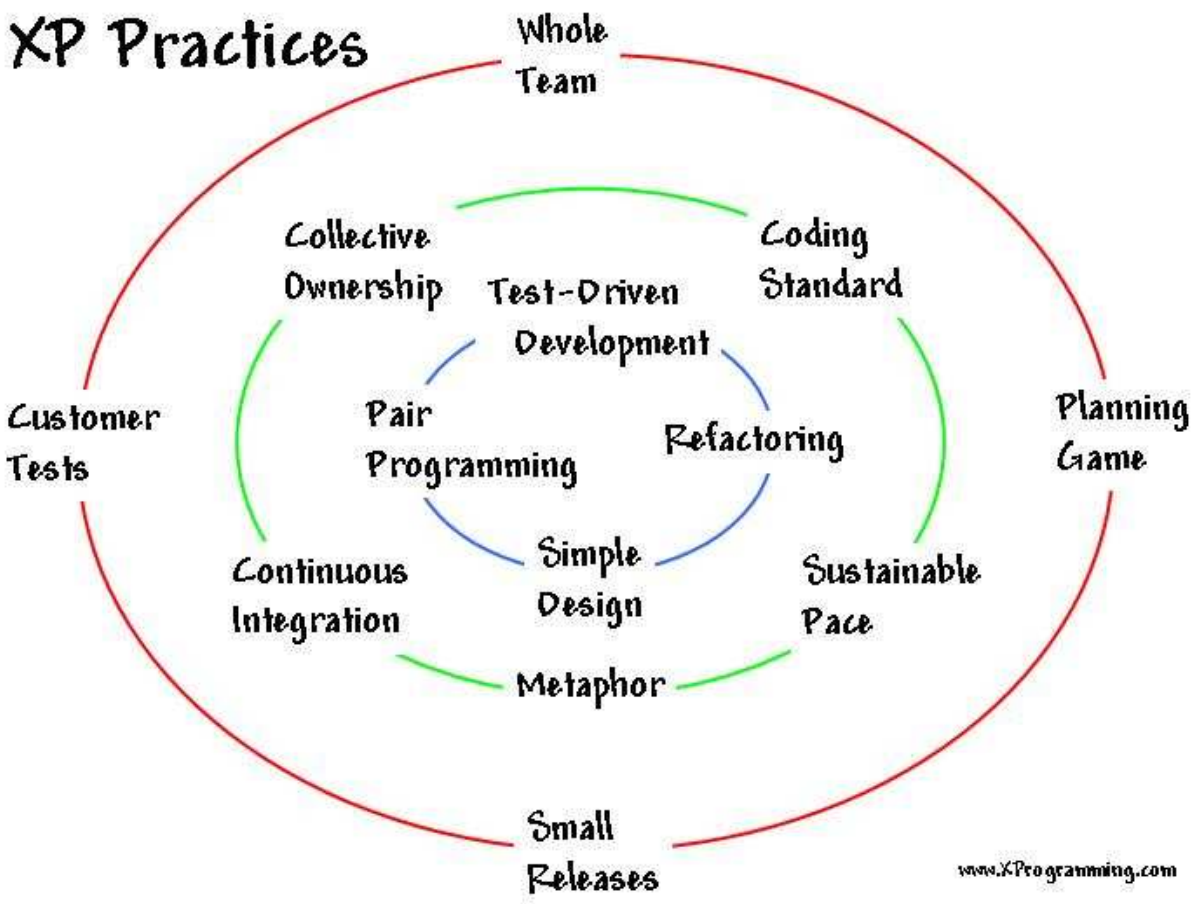

Abbildung 5: Übersicht über die XP-Praktiken[113]

Die obenstehende Abbildung gibt eine Übersicht über die in XP üblichen Praktiken.

3.3.3.1 Ganzes Team

Das Team als Ganzes hat die gemeinschaftliche Verantwortung. Es arbeitet ohne starre Rollenverteilung zusammen. Jedes Mitglied ist technisch und fachlich kompetent und im Team gleich-

[112] [Beck2000], S. 135f.

[113] Bild aus: www.xprogramming.com, letzter Zugriff 01.05.2008

berechtigt. Bringen sich alle Mitglieder des Teams gleichermaßen ein, so profitieren das Projekt und das ganze Team davon.[114]

3.3.3.2 Planungsmeeting

Jede Iteration beginnt mit einem Planungsmeeting, in dem das Kundenteam seine Geschichten erzählt und mit dem Programmierteam diskutiert. Die Programmierer schätzen den Aufwand grob ab, den sie zur Entwicklung jeder einzelnen Geschichte benötigen werden. Die Kunden wählen in Abhängigkeit der Aufwandsschätzungen den Kartenumfang für die Iteration aus, der ihren Geschäftsgegenwert maximieren würde. Die Programmierer zerlegen die geplanten Geschichten am Flipchart in technische Aufgaben, übernehmen Verantwortung für einzelne Aufgaben und schätzen deren Aufwände vergleichend zu früher erledigten Aufgaben. Aufgrund der genaueren Schätzung der kleinen Aufgaben verpflichten sich die Programmierer auf genau soviele Geschichten, wie sie in der vorhergehenden Iteration entwickeln konnten. Diese Planungsspiele schaffen eine sichere Umgebung, in welcher geschäftliche und technische Verantwortung zuverlässig voneinander getrennt werden.[115]

3.3.3.3 Kunde vor Ort

Das für die anstehenden Programmieraufgaben nötige Verständnis der Anforderungen wird fortlaufend in der Konversation mit den Kunden geprüft und vertieft. In kurzen Designsessions wird unter Umständen auf eine der Wandtafeln ein wenig UML gemalt oder es werden Szenarien mit Hilfe von CRC-Karten durchgespielt. Während der gesamten Entwicklung dienen die Kunden als direkte Ansprechpartner zur Bewältigung fachlicher Fragen. Die verbleibende Zeit verbringen die Kunden mit dem Schreiben und Ergründen neuer Benutzergeschichten und Akzeptanztests.[116]

3.3.3.4 Akzeptanztests

Die Kunden spezifizieren während der Iteration funktionale Abnahmekriterien. Ein Tester hilft üblicherweise dem Kunden, diese Tests zu erstellen und automatisch auszuführen. Spätestens zum Ende der Iteration müssen die Tests erfüllt sein, um die gewünschte Funktion des Systems zu sichern.[117]

[114] [Hüttermann], S. 31.

[115] [Wolf2005], S. 32ff.

[116] [Wolf2005], S. 25ff.

[117] [Wolf2005], S. 70ff.

3.3.3.5 Metapher

Entwickler und Kunde sollen eine gemeinsame Vision entwickeln, wie das zu entwickelnde System funktioniert. Diese gemeinsame Vision soll als Richtschnur für die Entwicklung des Systems dienen.[118]

3.3.3.6 Sustainable Pace

XP-Teams arbeiten eine längere Zeit zusammen. Regelmäßige Überstunden führen dazu, dass Effektivität und Motivation der Entwickler abnehmen. Damit das Team dauerhaft seine Leistung halten kann, muss sichergestellt werden, dass es sich nicht durch Überstunden selbst die Möglichkeiten nimmt, auch zukünftig noch produktiv zu sein.[119]

3.3.3.7 Collective Ownership

In einem XP-Projekt gehören der gesamte Quellcode und alle Dokumente dem gesamten Team. Jeder im Projekt kann und soll zu jeder Zeit alle Quelltexte und Dokumente ändern. Dies kann allerdings nur dann funktionieren, wenn jedes Teammitglied hinreichend Übersicht über das gesamte Projekt hat.[120]

3.3.3.8 Continous Integration

Das System wird mehrmals täglich durch einen automatisierten Build-Prozess neu gebaut. Der entwickelte Code wird in kleinen Inkrementen und spätestens am Ende des Tages in die Versionsverwaltung eingecheckt und ins bestehende System integriert. Die Unit Tests müssen zur erfolgreichen Integration zu 100% laufen.[121]

3.3.3.9 Kodierungsrichtlinien

Ein XP-Team verwendet Programmierrichtlinien, denen es folgt. Dies führt dazu, dass der Code immer gleich aussieht, unabhängig davon wer ihn geschrieben hat. Dies führt wiederum dazu, dass die Einarbeitung in den Code einfacher durchzuführen ist, da auch Änderungen durch Andere das Aussehen des Codes nicht beeinträchtigen.[122]

---

[118] {Wolf2005], S43ff. – Dieser Begriff ist mittlerweile in der zweiten Auflagen fallengelassen worden.

[119] [Wolf2005], S. 121ff.

[120] [Wolf2005], S107ff.

[121] [Wolf2005], S. 112ff.

[122] [Wolf2005]; S. 118ff.

3.3.3.10 Refaktorisierung

Das Design des Systems wird fortlaufend in kleinen, funktionserhaltenden Schritten verbessert. Finden zwei Programmierer Codeteile, die schwer verständlich sind oder unnötig kompliziert erscheinen, verbessern und vereinfachen sie den Code. Sie tun dies in disziplinierter Weise und führen nach jedem Schritt die Unit Tests aus, um keine bestehende Funktion zu zerstören.[123]

3.3.3.11 Einfaches Design

Design findet in einem XP-Projekt immer begleitend während der gesamten Entwicklung statt. Daher wird auch von inkrementellem Design gesprochen. So kann prinzipiell zu jedem Zeitpunkt das zu den Anforderungen optimale Design gewählt werden. Existiert bereits ein Design, wird es durch die Refaktorisierungen geändert und so an die neuen Anforderungen angepasst. Daher soll immer das einfachste Design gewählt werden, durch das die Anforderungen erfüllt werden.[124]

3.3.3.12 Pair Programming

Die Programmierer arbeiten stets zu zweit am Code und diskutieren während der Entwicklung intensiv über Entwurfsalternativen. Sie wechseln sich regelmäßig an der Tastatur ab und rotieren stündlich ihre Programmierpartner. Dies soll eine höhere Codequalität, größere Produktivität und eine bessere Wissensverbreitung bewirken.[125]

3.3.3.13 Testen

Gewöhnlich wird jede Zeile Code durch einen Testfall motiviert, der zunächst fehlschlägt. Die Unit Tests werden gesammelt, gepflegt und nach jedem Kompilieren ausgeführt.

Jeder Testfall wird auf die denkbar einfachste Weise erfüllt. Es wird keine Funktionalität implementiert oder getestet, die momentan nicht gefordert ist.[126]

3.3.4 Artefakte

XP definiert nur sehr wenige Artefakte. Dies sind die Story Cards mit den User Stories, Task Cards, die die User Stories in einzelne Tasks herunterbrechen, der Releaseplan mit Anforde-

[123] [Wolf2005], S. 88ff.

[124] [Wolf2005], S. 78ff.

[125] [Wolf2005], S. 95ff.

[126] [Wolf2005], S.62ff.

rungen für das nächste Release, und gegebenenfalls ein Plan, der die Aufgaben für die aktuelle Iteration enthält.

3.3.4.1 Story Cards

Mit Story Cards werden die fachlichen Anforderungen beschrieben[127]. Story Cards werden im Planungsspiel verwendet und während der gesamten Entwicklung als Basis der vom Entwickler umzusetzenden Funktionalität benutzt. Da die auf den Story Cards notierten User Stories meist sehr kurz formuliert sind, wird während der Umsetzung der Kunde zur Klarstellung offener Fragen zwingend benötigt.[128]

3.3.4.2 Task Cards

Task Cards dienen zum einen der Detaillierung und Zerlegung von schwer zu schätzenden und in einem Stück umzusetzenden User Stories. Weiterhin können sie verwendet werden, um technische Aufgaben zu beschreiben, die erforderlich sind, für den Kunden aber zunächst keinen erkennbaren Nutzen bringen, wie etwa die Erstellung von Infrastruktur-Komponenten oder größere Refaktorisierungen.[129]

3.3.4.3 Release-Plan

Der Releaseplan beschreibt, wann ein Release geplant ist. Weiterhin enthält der Releaseplan die Funktionen, die in dem Release enthalten sein sollen. Der Releaseplan wird häufig nur als einfacher Text oder kurze Präsentation erstellt. Eine detaillierte Planung mit einem Projektmanagementwerkzeug, wie MS-Projekt, wird weder als notwendig noch für sinnvoll gehalten.[130]

3.3.4.4 Iterations-Plan

Ein Release besteht in der Regel aus mehreren Iterationen. Iterationen werden durch die Priorisierung von User Stories und der Zuordnung zu Iterationen geplant. Iterationen werden meist auf der Basis von Kalenderwochen geplant.[131]

---

[127] In der Literatur wird nicht letztlich klar, ob es sich nun um Kundenanforderungen, oder um Anforderungen an das zu entwickelnde System handelt.

[128] [Wolf2005], S. 176ff.

[129] [Wolf2005], S. 181ff.

[130] [Wolf2005], S. 185ff.

[131] [Wolf2005], S. 185.

3.4 Feature Driven Development (FDD)

„Feature Driven Development" (FDD) ist ein iteratives und inkrementelles Vorgehensmodell, das sich seit über zehn Jahren in kleinen und großen Softwareprojekten bewährt hat[132]. FDD wurde von Jeff De Luca im Jahre 1997 als schlanke Methode definiert, um ein großes zeitkritisches Bankprojekt in Singapur durchzuführen[133]. Seitdem wurde FDD kontinuierlich weiterentwickelt. FDD stellt den Feature-Begriff in den Mittelpunkt der Entwicklung. Jedes Feature stellt einen Mehrwert für den Kunden dar. Die Entwicklung wird an Hand eines Feature-Plans organisiert. Eine wichtige Rolle spielt der Chef-Modellierer, der ständig den Überblick über die Gesamtarchitektur und die fachlichen Kernmodelle behält. Bei größeren Teams werden einzelne Entwicklerteams von Chef-Programmierern geführt.

FDD wird üblicherweise in der Form von fünf Prozessen beschrieben. Die Abfolge der Prozesse sieht auf den ersten Blick wasserfallartiger aus, als sie es in der Praxis ist. Zum einen werden für die ersten drei Prozesse nur kurze Zeiträume von nur wenige Tage investiert. Zum anderen werden die Prozesse #4 und #5 in ständigem Wechsel durchgeführt, da jedes Feature in maximal zwei Wochen realisiert wird. Dadurch kann FDD nicht ganz so schnell reagieren wie XP oder Scrum. Der höhere Planungsaufwand macht FDD etwas schwerfälliger, allerdings auch weniger anfällig für bestimmte Störungen im Entwicklungsprozess.

Das von FDD definierte Prozess- und Rollenmodell harmonisiert gut mit existierenden Strukturen in größeren Unternehmen. Auf Grund der Struktur von FDD fällt es vielen Unternehmen leichter, FDD zu verwenden als andere agile Methoden. Die Voraussetzung für FDD ist ein in den Grundzügen festgelegter Projektumfang beim Projektstart und ein moderates Tempo für Anforderungsänderungen. Ist das Projektteam groß und heterogen oder arbeiten die Entwickler nur einen begrenzten Zeitraum zusammen, bietet FDD eine Lösung für ein diszipliniertes und zielgerichtetes Vorgehen. Ein auffälliger Unterschied zwischen FDD und XP ist die Verantwortlichkeit für den erstellten Code: In FDD werden die erstellten Klassen fest Entwicklern zugeordnet[134]. FDD meidet die mit Collective-Ownership von XP üblicherweise in Verbindung gebrachten Gefahren, wie No-Ownership oder Strukturverlust im Code. Dafür muss mehr in die Planung investiert werden, als dies in einem XP-Projekt der Fall ist, insbesondere in die Zuordnung von Features zu Entwicklern.

[132] s. www.featuredrivendevelopment.com, letzter Zugriff 02.05.2008.

[133] 5 Monate, 50 Entwickler, s. www.nebulon.com, letzter Zugriff 02.05.2008.

[134] Sogenanntes Class-Ownership

3.4.1 Rollen

FDD kennt mehr Rollen, als die agilen Vorgehensmodelle XP und Scrum.

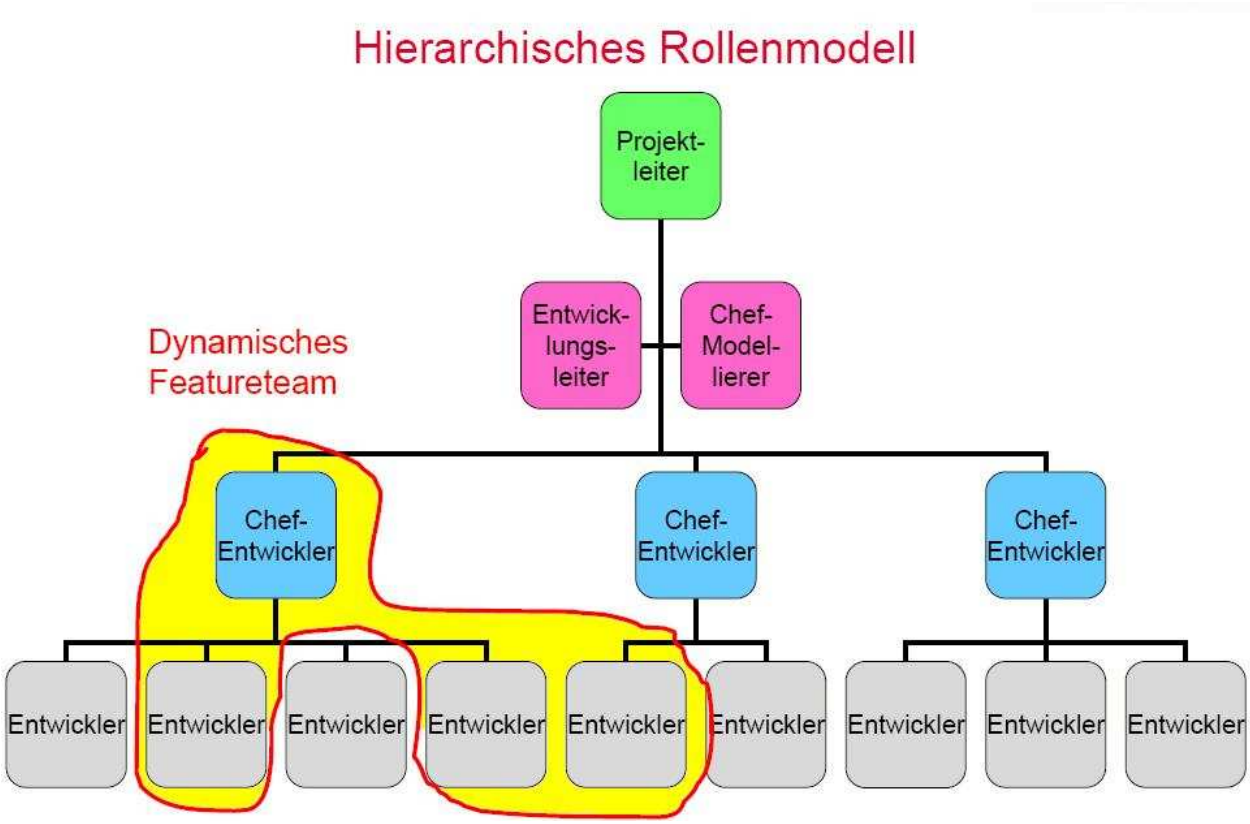

Abbildung 6: Rollen im FDD-Entwicklungsprozess[135]

3.4.1.1 Projektleiter

Der Projektleiter ist für den organisatorischen Ablauf des Projektes zuständig. Daher wird er regelmäßig von dem Chefentwickler über den Stand der in Entwicklung befindlichen Features informiert. Der Projektleiter berichtet gegenüber dem Management und dem Kunden.

3.4.1.2 Chefarchitekt

Der Chefarchitekt ist für das Design der Anwendung verantwortlich. Daher ist er bereits zu Beginn in die Erstellung des (fachlichen) Gesamtmodells involviert. Im Zuge der weiteren Entwicklung achtet er darauf, dass die Entwickler die im Gesamtmodell vorgesehen Prinzipien berücksichtigen. Daher berät er Entwickler gegebenenfalls bei dem Entwurf.

[135] www.it-agile.de, letzter Zugriff 01.05.2008.

3.4.1.3 Chefentwickler

Der Chefentwickler leitet ein oder mehrere Feature-Teams während der Implementierung eines Features. Der Chefentwickler trägt die Verantwortung für die Umsetzung eines Features.

3.4.1.4 Entwickler

Die Entwickler sind für den Entwurf, die Implementierung und den Test von Features zuständig.

3.4.1.5 Fachexperten

Fachexperten werden für die Festlegung der fachlichen Eigenschaften benötigt. Im Gegensatz zu XP und Scrum haben die Fachexperten jedoch keinen Einfluss auf die Reihenfolge, in der die Features entwickelt werden.

3.4.2 Phasen

Im FDD Vorgehensmodell ist nicht von Phasen die Rede. Von den aufgeführten 5 sogenannten Prozessen werden jedoch die ersten 3 Prozesse sequentiell ausgeführt, und die beiden anderen Prozesse in einer oder mehreren Iterationen. Insoweit kann man durchaus von 4 Phasen sprechen. Ein FDD-Projekt ist auf maximal 6 Monate begrenzt. Ist das Projekt größer, muss das Gesamtprojekt in mehrere FDD-Projekte aufgeteilt werden.

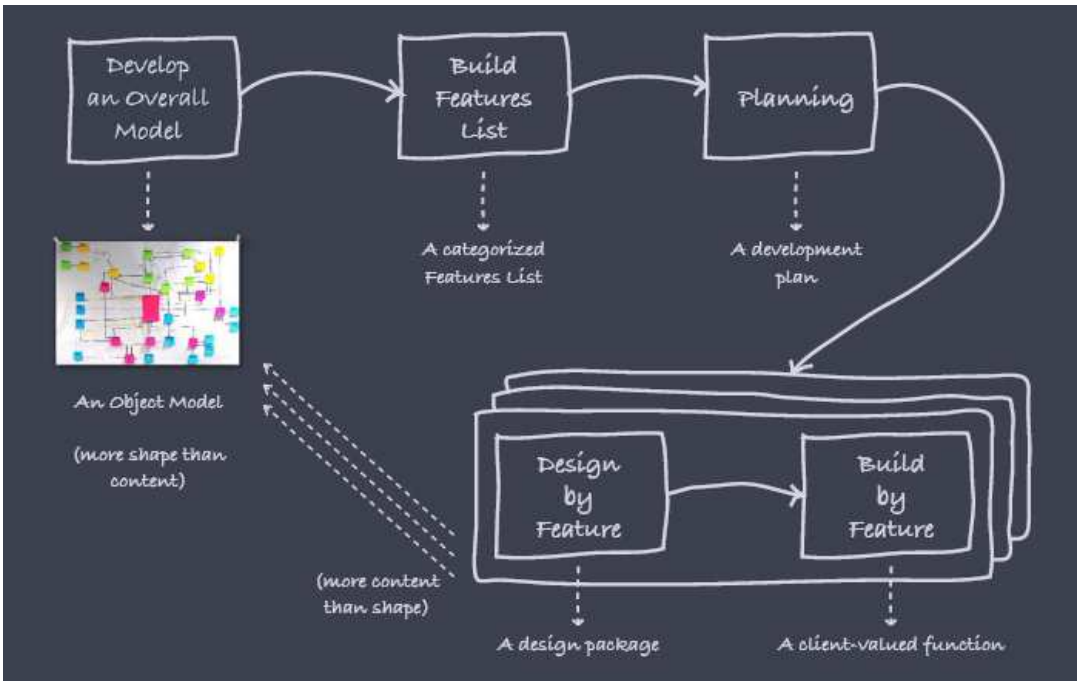

Abbildung 7: Übersicht über den FDD- Entwicklungsprozess[136]

[136] www.nebulon.com, letzter Zugriff 02.05.2008.

3.4.2.1 Gesamtmodell entwickeln

Das FDD-Vorgehensmodell beginnt damit, dass ein Modellierungsteam, bestehend aus Fachexperten und Entwickler unter Leitung des Chefarchitekten[137], Inhalt und Umfang des zu entwickelnden Systems definiert. In Kleingruppen werden Fachmodelle für die einzelnen Bereiche des Systems erstellt, die in der Gruppe vorgestellt, gegebenenfalls überarbeitet und schließlich integriert werden. Änderungsvorschläge können vom Chefarchitekten und in der Diskussion angebracht werden. Das Ziel dieses ersten Prozesses ist ein gemeinsames fachliches Verständnis über den zu unterstützenden Anwendungsbereich. Es wird sowohl analysiert als auch auf konzeptioneller Ebene das fachliche Kernmodell modelliert. Ein externes Review kann bei Bedarf durchgeführt werden. Die Ergebnisse des ersten Prozesses sind Klassendiagramme, eine informale Feature-Liste sowie Notizen zu den Alternativvorschlägen. FDD empfiehlt für die Modellierung „Modeling in Color“ einzusetzen, einen Modellierungsansatz, der durch wiederkehrende Archetypen und standardisierte Beziehungen zwischen fachlichen Klassen sehr schnell zu einheitlichen Modellen führt.[138]

3.4.2.2 Featureliste erstellen

Nachdem das Gesamtmodell erstellt wurde, wird es von den Chefprogrammierern funktional in Fachgebiete[139] aufgeteilt. Dann werden die darin enthaltenen Geschäftsaktivitäten[140] ermittelt und weiter in einzelne Geschäftsaktivitätschritte[141] unterteilt. Durch sogenannte Features werden die Geschäftsaktivitätschritte realisiert. Unter Features werden feingranulare Funktionen verstanden, die für den Kunden einen Nutzen darstellen. Die Features werden sehr prägnant nach dem einfachen Schema „Aktion/Ergebnis/Objekt“ notiert, z. B. „Berechne Gesamtsumme der Verkäufe”. Jedes Feature darf maximal zwei Wochen für seine Realisierung benötigen. Das Ergebnis dieses zweiten Prozesses ist eine Feature-Liste, kategorisiert nach Geschäftstätigkeiten und Fachgebieten. Diese Feature-Liste wird unter Mitwirkung der Fachexperten erstellt und muss auch für diese verständlich sein.

3.4.2.3 Featureplanung

In der nachfolgenden Phase planen der Projektleiter, der Entwicklungsleiter und die Chef Programmierer die Reihenfolge, in der Features realisiert werden sollen. Jede Geschäftsaktivität mit

[137] Der Chefarchitekt wird neuerdings Chefmodellierer genannt.

[138] [Balzert2008], S. 667.

[139] Subject areas

[140] Business activities

[141] Business activity steps

allen darin enthaltenen Features wird von einem Chefprogrammierer verantwortet. Auf Basis des Plans werden die Fertigstellungstermine je Geschäftsaktivität festgelegt. Bei der Planung richtet er sich nach den Abhängigkeiten zwischen den Features, der Auslastung der Programmierteams sowie der Komplexität der Features. Außerdem werden für die bekannten fachlichen Klassen Entwickler als Besitzer festgelegt.[142]

3.4.2.4 Entwicklung

In dieser Phase werden die Features entworfen, entwickelt und getestet. Das Feature-Team erstellt ein oder mehrere Sequenzdiagramme für die Features und verfeinert die Klassenmodelle des Gesamtmodells. Bei fachlichen Unklarheiten werden die Fachexperten hinzugezogen. Bei Bedarf werden erste Klassen- und Methodenrümpfe für die benötigten Klassen erstellt. Sodann wird der so erstellte Entwurf einem Review unterzogen. (Design Inspection).

Nach der Erstellung des Entwurfs wird das Feature implementiert. Auch hier arbeitet das Featureteam zusammen. Zusätzlich werden Klassen und Komponententest erstellt. Nachdem das Feature implementiert ist, wird der entstandene Code inspiziert.

3.4.3 Praktiken

In FDD werden Praktiken nicht explizit erwähnt. Die folgenden können aber den Beschreibungen entnommen werden:

3.4.3.1 Tägliches Projektstatusmeeting

An dem täglichen Projektstatusmeeting nehmen der Chefentwickler und die Entwickler teil. Das tägliche Projektstatusmeeting soll eine Dauer von 15 bis 30 Minuten haben. Es soll jeden Tag am selben Ort und zur selben Zeit stattfinden. Teilnehmer sind der Chefentwickler und die Entwickler. Zweck dieser kurzen Besprechung ist es, dass jedes Teammitglied einen kurzen Stausbericht an den Chefprogrammierer und die übrigen Entwickler gibt.[143]

3.4.3.2 Wöchentliches Projektstatusmeeting

In dem wöchentlichen Projektstatusmeeting berichten die Chefentwickler den Stand jedes einzelnen Features an den Projektmanager. Der Projektmanager akkumuliert diese Daten und verwendet sie als Basis für den Parking Lot Chart und den Trend Chart.[144]

---

[142] Die sogenannte Class Owner List.

[143] [Roock-Wolf-4]. S. 126.

[144] [Roock-Wolf-4]. S. 126.

3.4.3.3 Code-Inspektionen

Im Gegensatz zu XP verzichtet FDD auf Pair-Programming. Stattdessen werden Code-Inspektionen durchgeführt. Der Chefentwickler entscheidet, ob die Code-Inspektionen nur innerhalb des Featureteams durchgeführt werden, oder weitere Projektmitglieder hinzugezogen werden.[145]

3.4.3.4 Komponententests

FDD fordert nicht explizit die Verwendung von Test-Driven-Design, wie es in XP üblich ist. Dennoch wird die Erstellung von Komponententests auch in FDD verlangt. Wie umfangreich diese Tests sein sollen, wird von dem Chefentwickler festgelegt.[146]

3.4.3.5 Class-Ownership

Im Gegensatz zu XP gibt es in FDD für jede Klasse bzw. Komponente einen verantwortlichen Entwickler. Die Änderungen werden dann auch nur durch diesen Entwickler vorgenommen. Die Weitergabe des Wissens erfolgt über Code-Inspektionen und die Zusammenarbeit in Featureteams.[147]

3.4.3.6 Featureteams

Ein Featureteam besteht immer nur für die Zeit, in der ein Feature entwickelt wird. Diese Featureteams werden bereits bei der Planung zusammengestellt. Die Klassenverantwortlichen dieser Klassen bilden dann zusammen für die Zeit der Implementierung des Features ein Team. Ein Klassenverantwortlicher kann gleichzeitig in mehreren Featureteams Mitglied sein. [148]

3.4.4 Artefakte

Die folgenden Artefakte werden bei FDD erwähnt:

3.4.4.1 Gesamtmodell

Das Gesamtmodell wird zu Beginn der Entwicklung erstellt. Es enthält Klassendiagramme, die die zentralen fachlichen Abhängigkeiten dokumentieren, Notizen, die erläutern, warum bestimmte Modellentscheidungen getroffen wurden, und ggf. Sequenzdiagramme, die dynamisches Ver-

[145] [Roock-Wolf-3]. S. 128.

[146] [Roock-Wolf-3], S. 128.

[147] [Balzert2008], S. 669, [Roock-Wolf-3], S. 128.

[148] [Roock-Wolf-3], S. 126ff.

halten erläutern, sofern dies zu diesem Zeitpunkt zweckmäßig erscheint. Das Gesamtmodell wird während der Entwicklung regelmäßig erweitert.[149].

3.4.4.2 Featureliste

Die Feature-Liste enthält alle identifizierten Features. Die Feature-Liste wird unter Mitwirkung der Fachexperten erstellt und ist nach Geschäftstätigkeiten und Fachgebieten kategorisiert.[150]

3.4.4.3 Featureplan

Im Featureplan wird die Fertigstellung einer jeden Geschäftsaktivität festlegt. Daher enthält dieser Plan für jede Geschäftsaktivität

- den zuständigen Chefentwickler,
- die Features, die dieser Geschäftsaktivität zugeordnet sind,
- die Reihenfolge, in der die Features realisiert werden,
- für jedes Feature die fachlichen Klassen, die für das Feature relevant sind und
- den Aufwand, der für die Umsetzung eines jeden Features erwartet wird.[151]

3.4.4.4 Class-Owner-List

Diese Liste enthält den Verantwortlichen für jede fachliche Klasse. Sie wird benötigt, um die Featureteams zusammenzustellen.

3.4.4.5 Parking Lot Chart

Das Parking Lot Chart wird von dem Projektmanager erstellt und dient der Kommunikation mit dem Management und dem Kunden. Das Parking Lot Chart zeigt den Zustand jeder einzelnen Geschäftsaktivität. Jede Geschäftsaktivität wird durch ein Rechteck dargestellt, das den Namen der Geschäftsaktivität, den geplanten Fertigstellungstermin und den Fertigstellungsgrad (in Prozent) enthält. Farben geben darüber Auskunft, ob die Entwicklung begonnen wurde, beendet wurde oder verspätet ist.[152]

[149] [Roock-Wolf-2], S. 126ff. und [Balzert2008], S. 667.

[150] [Roock-Wolf-2], S. 127f.

[151] [Roock-Wolf-2], S. 128.

[152] [Roock-Wolf-4]. S. 126f.

3.4.4.6 Trend Chart

Das Trend Chart zeigt für jede Woche, wie viele Features offen, in Arbeit und wie viele abgeschlossen sind. Das Trend Chart zeigt sowohl die wöchentlichen Werte der Vergangenheit als auch die prognostizierte Entwicklung der Umsetzung der Features für die Zukunft. Das Trend Chart wird wöchentlich vom Projektmanager aktualisiert.[153]

[153] [Roock-Wolf-4]. S. 127.

4 Medizinische Software agil entwickeln

In diesem Kapitel soll nun konkret gezeigt werden, wie agile Vorgehensmodelle für die Entwicklung medizinischer Software eingesetzt werden können. Wie bereits oben ausgeführt, liegt der Fokus von Scrum auf den Bereichen Projektmanagement und Anforderungsmanagement. Ergänzt man nun dieses Vorgehensmodell um die Bereiche, die von den regulatorischen Anforderungen verlangt werden, so erhält man eine Anzahl von Freiheitsgraden, die in der Umsetzung vielfach wie Willkür erscheinen können. Daher bietet es sich an, diese freien Stellen durch ein bereits vorhandenes agiles Vorgehensmodell zu füllen. XP bietet im Gegensatz zu Scrum wiederum konkrete Praktiken für die Entwicklung, hat aber Defizite bei dem Projektmanagement. Eine Kombination dieser beidem Vorgehensmodelle erscheint also sinnvoll.[154]

Die Darstellung dieses Vorgehensmodells erfolgt so, dass zunächst die für dieses Vorgehensmodell benötigten Phasen inhaltlich beschrieben werden. Die in den jeweiligen Phasen auszuführenden Aktivitäten werden erläutert. Daran anschließend werden die für das Vorgehensmodell benötigten Rollen und die bei der Entwicklung entstehenden Artefakte beschrieben. Nachdem wir nun einen Eindruck von dem Vorgehensmodell erhalten haben, werden anschließend alle Aktivitäten mit den beteiligten Rollen und den entstehenden Artefakten detailliert beschrieben. Dabei wird ein Mapping auf die regulatorischen Anforderungen durchgeführt, um die Eignung des Vorgehensmodells für die Entwicklung medizinischer Software nachzuweisen.

4.1 Rollen

Scum kennt nur die Rollen Product Owner, Team und ScrumMaster[155], hat also ein sehr spartanisches Rollenmodell. Das ist aber verständlich, da in Scrum lediglich die organisatorischen Aspekte der Softwareentwicklung agil geregelt sind. In XP gibt es außer den beiden Rollen Kunde und Entwickler weitere Rollen, wie Coach, Tester, Tracker und Consultant[156]. Dabei entspricht der Product Owner in Scrum im Wesentlichen dem Kunden in XP. Der Rolle des ScrumMasters wird in XP nicht als so bedeutend gesehen, soweit erforderlich, wird diese Rolle dort durch den Coach eingenommen.

---

[154] Ich bin nicht der Erste, der auf diese Idee gekommen ist. Mike Breedle hat bereits eine solche Kombination entwickelt, und setzt sie unter dem Namen „Enterprise Agile Process“ ein. Informationen sind unter http://www.e-architects.com/AE/index.html verfügbar, letzter Zugriff 08.05.2008.

[155] s. Kapitel 3.2.1.

[156] s. Kapitel 3.3.1.

Bei der Entwicklung medizinischer Software mit so wenig Rollen auszukommen, erscheint mir sehr schwierig. Vor allem der organisatorische Aspekt ist in agilen Vorgehensmodellen ja gewollt unterrepräsentiert, um auch die Entscheidungsstrukturen flexibel zu belassen und möglichst viele Entscheidungen an das Team zu delegieren oder im Team zu entschieden.

4.1.1 Produktverantwortlicher

Der Produktverantwortliche entspricht dem Kunden in XP und dem Product Owner von Scrum[157]. Der Produktverantwortliche hat eine zentrale Rolle in dem Entwicklungsprozess. Zu seinen Aufgaben gehört:

- Das Erfassen der Kundenanforderungen und die Dokumentation dieser Anforderungen,
- das Priorisieren der Anforderungen,
- die Planung der Releases und das Festlegen der Releaseziele,
- das Erstellen des Projektplans und das Festlegen von Meilensteinen,
- die Planung von Inhalten, Zielen und Terminen für die Iterationen zusammen mit dem Chefentwickler, dem Qualitätsbeauftragten und den Entwicklern,
- die Festlegung von geeigneten Mess- und Kontrollmaßnahmen für das Projekt,
- die Steuerung und Kontrolle des Entwicklungsprozesses zusammen mit dem Chefentwickler und dem Qualitätsverantwortlichen
- das Reporting an das höhere Management.

Damit der Produktverantwortliche diese Aufgabe erfüllen kann, sollte er über die folgenden Qualifikationen verfügen:

- E kennt den Kunden und andere für das Projekt wichtige Personen und Institutionen,
- er hat ein umfassendes Verständnis von den fachlichen Anforderungen und Rahmenbedingungen,
- er ist Experte für die Begrifflichkeit des Anwendungsgebietes, d.h. spricht die Sprache des Anwenders, Kunden und Benutzers,
- er kennt die vorhandenen Systeme im Umfeld des Anwendungsgebietes,

[157] Der Name Kunde erscheint nicht angemessen, da der Produktverantwortliche lediglich das Sprachrohr des Kunden und weiterer Stakeholder ist, insofern entspricht er im Wesentlichen eher dem Product Owner von Scrum. In dieser Arbeit sollte aber ein deutscher Ausdruck benutzt werden.

- er hat Kenntnis darüber, welche Anforderungen stabil sind und welche sich womöglich ändern und welche Qualität sie benötigen,
- er kennt die für das Projekt relevante technische und fachliche Architektur, soweit vorhanden,
- er verfügt über grundlegende Kenntnisse des Benutzungskonzeptes,
- er kennt sich mit Anforderungsermittlung und Anforderungsanalyse aus und
- er verfügt über kommunikative Fähigkeiten und soziale Kompetenz.

4.1.2 Qualitätsverantwortlicher

Der Qualitätsverantwortliche verfügt über die Verantwortlichkeiten des Coaches in XP und des Scrum Masters in Scrum[158]. Zusätzlich ist er für die Qualitätssicherung und das Qualitätsmanagement innerhalb des Projektes zuständig. Weiterhin achtet er auf die Einhaltung der regulatorischen Vorgaben. Der Produktverantwortliche hat damit eine zentrale Rolle in dem Entwicklungsprozess. Zu seinen Aufgaben gehören im Einzelnen:

- Die Entwicklung eines Bewusstseins für den Entwicklungsprozess bei den Teammitgliedern,
- die Anpassung des Entwicklungsprozesses zusammen mit den Teammitgliedern, soweit dies erforderlich ist,
- die Organisation und Moderation von Besprechungen und Reviews,
- das Festlegen des erforderlichen Verfahrens für Verifikation und Validierung zusammen mit dem Team,
- die Erstellung der Testspezifikationen unter Mitwirkung des Teams,
- die Erstellung des Validierungsplans,
- die Durchführung der Validierung mit dem Team und
- er ist Ansprechpartner der benannten Stelle.

[158] Ob der Name „Qualitätsverantwortlicher“ angemessen ist, kann sicherlich in Frage gestellt werden. Da diese Rolle aber einerseits für die Prozesse innerhalb des Projektes zuständig ist, und andererseits die Organisation dieser Prozesse sicherlich einen Qualitätsaspekt besitzt, erscheint dieser Name nicht völlig verkehrt. Der Qualitätsbeauftragte ist weiterhin auch für die Qualität des entstehenden Produktes zuständig.

Der Qualitätsverantwortliche hat jedoch weder eine aktive Rolle bei der Festlegung von Anforderungen an das System, noch bei der Entwicklung des Systems. Damit der Qualitätsverantwortliche diese Aufgabe erfüllen kann, sollte er über die folgenden Qualifikationen verfügen:

- Er kennt sich mit Software-Entwicklungsprozessen aus und kennt agile Vorgehensmodelle,
- er hat ein umfassendes Verständnis der fachlichen Anforderungen,
- er sollte bereits selbst Software entwickelt haben,
- er kennt die regulatorischen Anforderungen an die Entwicklung medizinischer Software,
- er kennt sich mit Qualitätsmanagementsystemen wie ISO 9001 und ISO 13495 aus,
- er verfügt über Kenntnisse der analytischen und konstruktiven Qualitätssicherung und
- verfügt über kommunikative Fähigkeiten und soziale Kompetenz.

4.1.3 Chefentwickler

Die Rolle eines Chefentwicklers kennt weder Scrum noch XP, eine solche Rolle ist jedoch in agilen Vorgehensmodellen nicht völlig unbekannt. In FDD gibt es sowohl die Rolle eines Chefentwicklers als auch die einen Chefarchitekten. In dieser Arbeit wird die Auffassung vertreten, das es hilfreich ist, eine Rolle vorzusehen, die das Entwicklerteam als Ganzes leitet, technisch zentrale Entscheidungen verantwortet und als Ansprechpartner gegenüber dem Management fungiert. All dies sind Aufgaben des Chefentwicklers. Damit hat auch der Chefentwickler eine zentrale Rolle in dem Entwicklungsprozess. Zu seinen Aufgaben gehört im Einzelnen:

- Die Erstellung der fachlichen Systemstruktur,
- die Konzeption von Schnittstellen zu externen Anwendungen,
- die Identifikation und Klassifikation von Systemschnittstellen,
- die Definition der technischen Systemstruktur,
- die Beratung der Entwickler hinsichtlich Fragen zu Architektur, Design und Entwicklung,
- die Überwachung der Architektur- und Designvorgaben,
- die Festlegung und Überprüfen von Entwicklungsrichtlinien,
- das Leiten und Führen des Teams aus Entwicklern und Testern[159] und

[159] Es erscheint auf den ersten Blick irritierend, dass der Chefentwickler die Tester leitet, und nicht der Qualitätsverantwortliche. Der Qualitätsverantwortliche kann aber eine seine Hauptaufgaben, nämlich als

- das Leiten des täglichen Standup-Meetings[160]

Damit der Produktverantwortliche diese Aufgabe erfüllen kann, sollte er über die folgenden Qualifikationen verfügen:

- Er kennt die beteiligten Systeme in ihrer Gesamtheit,
- er kennt die Zusammenhänge und Wechselwirkungen zwischen den Subsystemen,
- er hat für das Entwicklungsteam die Verantwortung und Entscheidungskompetenz,
- er kennt Programmiermodelle und Entwicklungsrichtlinien und trägt diese mit,
- er hat Erfahrungen und Kenntnis über funktionale und nicht-funktionale Anforderungen an das zu entwickelnde System,
- er hat Erfahrung in der Analyse und dem Design von Softwaresystemen,
- er hat Erfahrung im Datenbankdesign,
- er verfügt über grundlegende Kenntnisse des Benutzungskonzeptes,
- er hat Erfahrung in der Anforderungsanalyse und
- er verfügt über kommunikative Fähigkeiten und soziale Kompetenz.

4.1.4 Projektkernteam

Das Projekt verfügt nicht über einen Projektleiter, der letztendlich alle wesentlichen Entscheidungen trifft. Vielmehr wird vielfach das Entwicklungsteam als Ganzes die Entscheidung treffen und dann auch verantworten, wie dies ja auch in Scrum und XP gebräuchlich ist. Für manche Entscheidungen erscheint dieses Verfahren aber doch zu umständlich. Daher wird das Projekt als Ganzes von einem Kernteam bestehend aus Produktverantwortlichem, Qualitätsverantwortlichem und Chefentwickler geleitet.[161] Das Kernteam ist dann als Ganzes für zentrale Entscheidungen zuständig und verantwortet sie gemeinsam.

Moderator zu fungieren, nur dann richtig ausüben, wenn er nicht anderseits die Tester anleitet. Daher werden in Zusammenhang mit ihm die Testspezifikationen erstellt, die Umsetzung und Ausführung der Testspezifikation obliegt aber dem Team und dem Chefentwickler.

[160] In Scrum wäre dies Aufgabe des ScrumMasters. Hier erscheint aber der Chefentwickler besser geeignet, da er näher an den Entwicklern ist, und deren Probleme besser versteht.

[161] Ich kann aus eigener Erfahrung berichten, dass ein solches Kernteam durchaus effektiv und effizient arbeiten kann.

4.1.5 Entwickler und Tester

Entwickler und Tester gibt es sowohl in Scrum als auch in XP. In Scrum werden beide nicht weiter differenziert, und es wird immer nur von dem Team als Ganzem gesprochen. In XP wird zum Einen von Entwicklern gesprochen, die für die Entwicklung der Iterationsergebnisse zuständig sind. Zu der Entwicklung gehören auch Designentscheidungen, Modifikationen des bestehenden Codes in Form von Refaktorisierungen und Testen des erstellten Codes durch Modul- und Komponententests. Zum Anderen gibt es Tester, die aber lediglich den Kunden bei der Erstellung von Akzeptanztests unterstützen.

In dieser Arbeit wird diese Unterscheidung zwischen Entwicklern und Testern nicht für zweckmäßig gehalten. Natürlich besteht das Team aus Personen mit unterschiedlichen Qualifikationen und unterschiedlichen Zuständigkeiten. Es ist illusorisch anzunehmen, jeder Entwickler verfügt über alle im Projekt erforderlichen Qualifikationen. Niemand wird einen Datenbankspezialisten für die Entwicklung von Weboberflächen einsetzen. Aber es ist wichtig, dass alle in einem Team zusammenarbeiten.

Die Erstellung von System- und Akzeptanztests ist ein integraler Bestandteil der Entwicklung. Die Separierung dieser Aktivitäten in ein separates Team erweckt aber den Eindruck, dass die Tests auf Systemebene etwas sind, das nach der Entwicklung kommt. Das soll aber gerade vermieden werden. „Fertig“ ist ein Iterationsinkrement erst dann, wenn es alle Tests absolviert hat.

4.1.6 Risikomanager

Diese Rolle wird in den agilen Vorgehensmodellen nicht beschrieben, da sie dort nicht benötigt wird. Bei der Entwicklung medizinischer Software ist aber ein Risikomanagementprozess erforderlich, über den sichergestellt wird, dass das Produkt kein unangemessenes Risiko für Patienten, Anwender und Dritte darstellt. Der Risikomanager ist dafür zuständig, dass alle Risiken identifiziert, bewertet und, soweit angemessen, Korrekturmaßnahmen eingeleitet und umgesetzt werden. Zu seinen Aufgaben gehört im Einzelnen:

- Analyse des Produkts, um mögliche Risiken zu identifizieren,
- Durchführung der Risikoanalyse, um mögliche und bekannte Gefährdungen zu identifizieren, und deren Schwere und Wahrscheinlichkeit zu bewerten,
- Festlegung von Maßnahmen zur Risikoverringerung,
- Überprüfung der korrekten Umsetzung der Maßnahmen zur Risikoverringerung,
- Bewertung des verbleibenden Restrisikos und
- Analyse und Bewertung von Beschwerden und Ereignissen.

Damit der Risikomanager diese Aufgabe erfüllen kann, sollte er über die folgenden Qualifikationen verfügen:

- Er kennt den Risikomanagementprozess (ISO 14971),
- er kennt Verfahren zur Identifizierung von Risiken, wie FMEA oder Fehlerbaumanalyse,
- er kennt sich mit Software-Entwicklungsprozessen aus und kennt agile Vorgehensmodelle,
- er hat ein umfassendes Verständnis von den fachlichen Anforderungen,
- er kennt die regulatorischen Anforderungen an die Entwicklung medizinischer Software und
- verfügt über kommunikative Fähigkeiten und soziale Kompetenz.

In kleinen Projekten kann diese Rolle gegebenenfalls von dem Qualitätsverantwortlichen übernommen werden.

4.1.7 Konfigurations- und Rolloutmanager

Diese Rolle wird in den agilen Vorgehensmodellen nicht beschrieben, da sie dort üblicherweise von dem Team ausgeführt wird. Bei der Entwicklung medizinischer Software ist aber ein Konfigurationsmanagementprozess erforderlich, der deutlich höhere Anforderungen an das Konfigurationsmanagement stellt, als dies in vielen anderen Projekten der Fall ist. Daher scheint eine Rolle für dieses Aufgabengebiet angemessen. Zu seinen Aufgaben gehört im Einzelnen:

- Oragnisieren und Verantworten des Konfigurationsmanagements,
- Planen des Build-Prozesses mit Ausführung der automatisierten Tests,
- Planen und Durchführen des Rollouts und
- Erstellen des Betriebskonzeptes und –handbuchs

Damit der Konfigurationsmanager diese Aufgabe erfüllen kann, sollte er über die folgenden Qualifikationen verfügen:

- Er kennt die Systemarchitektur, Zweck und Zusammenwirkung ihrer Bestandteile, die Paketstruktur usw.,
- er kennt die Abläufe und die beteiligten Werkzeuge zur Erstellung einer Konfiguration, des Gesamtsystems und der Teilsysteme,
- er verfügt über Grundkenntnisse zur Systemadministration der vorhandenen Betriebsysteme,

- er verfügt über Grundkenntnisse in Skriptsprachen und
- verfügt über kommunikative Fähigkeiten und soziale Kompetenz.

In kleinen Projekten kann diese Rolle gegebenenfalls von dem Produktverantwortlichen übernommen werden.

4.1.8 Weitere Rollen

XP kennt weitere Rollen, die bei Bedarf eingesetzt werden können, wie Consultant oder Tracker. Solche oder andere Rollen können bei Bedarf ergänzt werden. Ziel dieser Arbeit ist, ein minimales System von Rollen zu definieren, mit denen der Entwicklungsprozess auskommt. Gerade bei größeren Projekten wird man zwangsläufig die beschriebenen Rollen auf mehrere Personen verteilen müssen. Wie eine solche Erweiterung auf mittlere und große Projekte aussehen könnte, wäre das Thema einer weiteren Arbeit, und wird daher hier nicht untersucht.[162]

4.2 Phasen

Es ist – auch bei der Verwendung agiler Vorgehensmodelle - illusorisch zu glauben, man könne zu Beginn des Projektes direkt mit der Entwicklung beginnen, ohne weitere Vorbereitungen zu treffen. Daher kennen auch agile Vorgehensmodelle zeitlich begrenzte Abschnitte, die der Vorbereitung und Planung des anstehenden Projektes dienen. Scrum kennt dafür die Pre-Game-Phase, die dort weiter in eine Planning- und eine Staging-Phase unterteilt wird. XP benutzt vor der Entwicklung eine Explorationsphase und eine Planungsphase für das erste Release, bevor die Entwicklung beginnt. An diese beiden vorbereitenden Phasen schließt sich eine Entwicklungsphase an, in der die gesamte Entwicklung durchgeführt wird. Nachfolgend wird in beiden Vorgehensmodellen eine Phase für den Übergang in die Nutzung vorgesehen. XP schließt dann noch eine Wartungsphase an, die für die Weiterentwicklung gedacht ist. An diesen Unterteilungen orientieren auch wir uns in dieser Arbeit. In diesem Kapitel werden die in den Phasen ausgeführten Aktivitäten in dem Umfang beschrieben, wie dies zum Verstehen der Bedeutung der jeweiligen Phasen erforderlich ist. Eine ausführliche Erläuterung der Aktivitäten mit den beteiligten Akteuren, den benötigten und erzeugten Artefakten und der einzusetzenden Methoden wird in Kapitel 5 erfolgen.

[162] Wir gehen von einer „normalen“ Teamgröße aus, also einem Team von 7 bis 10 Entwicklern und Testern.

4.2.1 Konzeptphase

Diese Phase dient der grundlegenden Orientierung für das durchzuführende Projekt. Sie entspricht insoweit der Planningphase von Scrum[163] und einem wesentlichen Teil der Explorationsphase von XP[164]. Das wichtigste Ergebnis dieser Phase ist die Entscheidung, ob das Projekt durchgeführt werden soll.

In dieser Phase wird die Aktivität „Produktkonzept erstellen“ durchgeführt. Die wichtigsten Tätigkeiten dieser Aktivität sind:

- Erkundung des fachlichen Umfeldes mit Identifikation und Festlegung der zentralen fachlichen Konzepte, basierend auf ersten Anforderungen von Kunden und anderen Stakeholdern,
- Identifikation der wesentlichen fachlichen, organisatorischen und technischen Risiken und
- Durchführung einer ersten groben Schätzung für das Projekt hinsichtlich Kosten, Dauer und Ressourcen.

Diese Phase unterscheidet sich also nicht wesentlich von dem, was auch in einem konventionellen Vorgehensmodell durchgeführt würde. Diese Aktivität wird nicht zwangsläufig von den Mitarbeitern durchgeführt, die später an dem Projekt beteiligt sein werden, jedoch sollte versucht werden, soweit wie möglich auf solche Mitarbeiter bereits in dieser Phase zurückzugreifen. Diese Phase wird nicht iterativ mehrfach durchlaufen, sondern einmalig vor Beginn des Projektes durchgeführt.

Das Ergebnis ist ein Produktkonzept, in dem die Vision eines neuen oder verbesserten Produktes beschrieben ist und die prinzipielle Durchführbarkeit des Projektes dargestellt wird. Dieses Konzept wird von dem Management benutzt, um zu der Entscheidung zu gelangen, ob das Projekt durchgeführt werden soll. Ein Übergang in die nachfolgende Phase erfolgt, nachdem das Produktkonzept freigeben wurde, und das Management den Projektauftrag erteilt hat.

4.2.2 Planungsphase

Nachdem die Entscheidung gefallen ist, das Projekt durchzuführen, muss das Projekt geplant werden. Planung meint eine zunächst grobe Strukturierung der wichtigsten Tätigkeiten an Hand von Meilensteinen und eine feinere Planung der nächsten Aktivitäten. Zudem müssen die Vorbereitungen getroffen werden, die es erlauben, die Entwicklung des Systems in die Wege zu leiten.

[163] S. Kapitel 3.2.2.1.

Diese Phase unterscheidet sich also nicht wesentlich von dem, was auch in einem konventionellen Vorgehensmodell durchgeführt würde. Diese Aktivität wird bereits von den Mitarbeitern durchgeführt, die auch an den späteren Phasen des Projekt beteiligt sein werden, deshalb ist es unumgänglich, bereits zu Beginn dieser Phase die Projektmitarbeiter auszuwählen und zentrale Rollen wie den Produktverantwortlichen, den Qualitätsbeauftragten und den Chefentwickler zu besetzen. Diese Phase wird, wie auch die Konzeptphase nicht iterativ mehrfach durchlaufen, sondern lediglich einmalig zu Beginn des Projektes durchgeführt.

4.2.2.1 Entwicklung planen

Die regulatorischen Anforderungen legen fest, dass der benutzte Entwicklungsprozess definiert und dokumentiert wird. Dies beinhaltet die Aktivitäten für das Risikomanagement nach ISO 14971 und die Verfahren zur Verifikation und Validierung des zu erstellenden Produktes. Diese Verfahren werden zu diesem Zeitpunkt noch nicht detailliert spezifiziert, sondern lediglich die Struktur der Aktivitäten skizziert und soweit detailliert, wie dies zu diesem Zeitpunkt möglich ist. Zudem müssen Anforderungen an die Dokumentation und die verwendeten Normen und Standards festgelegt werden. Die wesentlichen Ergebnisse dieser Aktivität sind der Entwicklungsplan, der Risikomanagementplan und der Verifikations- und Validierungsplan.

4.2.2.2 Anforderungen festlegen

Es werden die ersten Anforderungen ermittelt, und durch den Produktverantwortlichen in das „Product Backlog“[165] eingestellt und priorisiert. Die Anforderungen werden so vollständig erfasst, wie dies zu diesem Zeitpunkt möglich ist. Es wird aber immer davon ausgegangen, dass die Anforderungen nicht vollständig sein werden. Wie viele Anforderungen zu diesem Zeitpunkt tatsächlich ermittelt werden können, hängt nicht zuletzt davon ab, wie innovativ das Produkt ist. Es werden außer Kundenanforderungen auch Anforderungen aus Marketing, Service und Wartung berücksichtigt. Weiterhin sollten bereits alle bekannten nicht-funktionalen Anforderungen aufgenommen werden. Zudem wird der bestimmungsgemäße Gebrauch festgelegt.

4.2.2.3 Releases und Iterationen planen

Anschließend legen der Produktverantwortliche, der Chefentwickler und der Qualitätsverantwortliche gemeinsam den Releaseplan fest. Darin wird festgelegt, wie viele Releases für die Produktentwicklung erforderlich sind, und welche Funktionalität in den Releases enthalten sein soll.

---

[164] S. Kapitel 3.3.2.1.

[165] Den Begriff „Product Backlog“ wird mit Absicht aus Scrum benutzt und nicht verdeutsch, um deutlich zu machen, dass es sich eben nicht um das typische Anforderungsdokument handelt.

Es wird weiterhin geschätzt, wie viele Iterationen für das erste Release benötigt werden, und welche Funktionalität in welcher Iteration umgesetzt wird. Alle diese Angaben sind noch ungenau, da die Detailplanung erst zu Beginn eines Release beginnt, und für jede einzelne Iteration zu Beginn der Iteration weiter detailliert wird. Als Ergebnis dieser Aktivität erhalten wir einen Projektplan mit den Funktionalitäten für jedes geplante Release und den Meilensteinen für die Fertigstellung.

4.2.2.4 Konzeptionelle Architektur erstellen

Es wird eine erste konzeptionelle Architektur für das Anwendungssystem erstellt, die als Basis der Entwicklung und der Kommunikation der Entwickler untereinander dient. In dieser Architektur, wird bereits die grundsätzliche fachliche Aufteilung des Systems in Teilsysteme vorgenommen. Daher wird in dieser Architektur herausgearbeitet, welche Teilsysteme für das Produkt sinnvoll sind, für welche Zwecke und Aufgaben diese Teilsysteme benötigt werden und welche Beziehungen und Abhängigkeiten zwischen den Subsystemen bestehen. Das Ergebnis dieser Aktivität ist das Systemarchitekturdokument.

4.2.2.5 Infrastruktur festlegen

Es wird die Infrastruktur für das Entwicklungsteam festgelegt. Dies beinhaltet die Festlegungen für die Versionskontrolle, das Verfahren für das tägliche Build, das Verfahren für das automatisierte Ausführen von Modul- und Komponententests und der weiter damit in Zusammenhang stehenden Tätigkeiten. Das Ergebnis dieser Aktivität ist ein Infrastrukturdokument, das die getroffenen Festlegungen enthält und die Umsetzung dieser Anforderungen selbst. Am Ende dieser Aktivität soll also eine lauffähige Infrastruktur verfügbar sein.

4.2.3 Konstruktionsphase

In der Konstruktionsphase wird nun das Produkt entwickelt und soweit vorbereitet, dass es in die Nutzung überführt werden kann. Medizinische Software unterscheidet sich von Software, die für ein nicht reguliertes Umfeld gedacht ist, nicht zuletzt dadurch, dass Sicherheitsrisiken explizit berücksichtigt werden müssen. Daher kann ein Einsatz unter produktiven Bedingungen nicht so einfach hergestellt werden, wie im Allgemeinen üblich. Es ist auch nicht möglich, eine funktionseingeschränkte Version des Softwaresystems auszuliefern und produktiv einzusetzen, wenn die fehlenden Funktionen die Sicherheit von Patienten oder Benutzern beeinträchtigen können. Es ist aber sehr wohl möglich, eine solche Software auszuliefern, wenn diese Software nur zu Testzwecken benutzt wird. Daher unterschieden wir Arten von Releases:

- Ein einfaches Release, das nicht für den Produktivbetrieb gedacht ist, aber sehr wohl an den Kunden ausgeliefert wird, damit der Kunde Erfahrung mit dem Produkt sammeln kann und
- ein freigegebenes Release, das durch einen formalen Freigabeprozess gegangen ist.

Die Konstruktionsphase schließt mit der Fertigstellung des ersten freigegeben Releases ab, jedes sich daran anschließende Release wird dann in der Wartungsphase ausgeführt.

4.2.3.1 Release planen

Jedes Release beginnt mit einem Planungsmeeting. An diesem Meeting nehmen der Produktverantwortliche, der Chefentwickler, der Qualitätsbeauftragte, der Risikomanager und der Konfigurationsmanager teil, es soll maximal 8 Stunden dauern. In diesem Planungsmeeting wird die Funktionalität festgelegt, die in diesem Release umgesetzt werden soll. Dazu werden:

- die Anforderungen durch den Produktverantwortlichen priorisiert und dem Release zugeordnet,
- der Erstellungsaufwand für diese Anforderungen durch den Chefentwickler grob geschätzt und dann die Anzahl der Iterationen und der Meilenstein für die Fertigstellung festgelegt,
- der Aufwand für die Erstellung von System- und Akzeptanztests durch den Qualitätsbeauftragten bestimmt,
- die Anforderungen von dem Risikomanager gesichtet, und der Aufwand für die Risikobetrachtung festgelegt.

Alle Festlegungen stehen unter dem Vorbehalt der Anpassung bei der Iterationsplanung. Es ist allen Beteiligten bekannt, dass die in diesem Meeting ermittelten Daten nur erste Schätzungen sein können. Das Ergebnis dieser Aktivität ist ein aktualisierter Projektplan und ein Releaseplan für das anstehende Release.

4.2.3.2 Iteration planen

Jede Iteration beginnt mit einem Planungsmeeting[166]. Dieses Planungsmeeting dauert maximal 8 Stunden und setzt sich aus zwei Teilen zusammen, die jeweils maximal 4 Stunden dauern. Teilnehmer der Besprechung sind der Produktverantwortliche, der Chefentwickler, der Qualitätsbeauftragte und das gesamte Team.

[166] Ein solches Meeting wird sowohl von Scrum (s. Kapitel 3.2.3.2) als auch XP gefordert (s. Kapitel 3.3.3.2). Die Inhalte orientieren sich aber mehr an Scrum als an XP.

Jede Iteration beginnt mit einem Planungsmeeting, in dem der Produktverantwortliche die von ihm priorisierten Anforderungen vorstellt und mit dem Entwicklungsteam diskutiert. Die Entwickler schätzen den zur Entwicklung benötigten Aufwand. Der Produktverantwortliche wählt in Abhängigkeit der Aufwandsschätzungen den Umfang für die Iteration aus, von dem das Team glaubt, ihn in der nachfolgenden Iteration umsetzen zu können. Dabei wählt der Produktverantwortliche letztendlich aus, welche Anforderungen umgesetzt werden sollen. Die Entscheidung, über den Umfang der umzusetzenden Anforderungen, liegt andererseits in der alleinigen Verantwortung des Teams und des Chefentwicklers.

Im zweiten Teil des Planungsmeetings wird das Team nun planen, wie die Anforderungen in Produktinkremente umgesetzt werden. Daher werden die einzelnen Anforderungen in Aufgaben heruntergebrochen, in eine Reihenfolge gebracht und Teammitgliedern zugewiesen. Diese Tätigkeit liegt in der alleinigen Verantwortung des Teams und wird vom Chefentwickler moderiert. Der Produktverantwortliche hat für diesen Teil der Besprechung lediglich eine beratende Funktion. Das Ergebnis dieser Besprechung ist das Iteration Backlog.

Die Besprechung endet mit der gemeinsamen Verpflichtung des Teams – dem Commitment - die soeben festgelegten Anforderungen innerhalb des Sprints in Produktinkremente umzusetzen. Das Ergebnis dieser Aktivität ist ein aktualisierter Releaseplan und Iterationsplan für die anstehende Iteration.

4.2.3.3 Iteration durchführen

Innerhalb einer Iteration werden alle vorbereiteten Arbeitsaufträge umgesetzt. In diesen Arbeitsaufträgen finden also alle geplanten Entwicklungsaktivitäten statt.

Es werden die folgenden Arten von Iterationen unterschieden:

- **Standarditeration**: Ziel einer Standarditeration (oder auch einfach nur Iteration) ist es, die Produktinkremente zu erstellen und damit das betreffende Produkt weiterzuentwickeln. Innerhalb einer Iteration finden daher regelmäßig alle elementaren Entwicklungstätigkeiten statt, also die Analyse der Anforderungen, die Konzeption der Lösung, die Erstellung der Lösung und die Überprüfung der Lösung. Es handelt sich hier aber nicht um Phasen der Entwicklung, sondern um Aktivitäten die in jeder Iteration immer neu anfallen, und so das Produkt inkrementell fertig stellen. Die genaue Ausgestaltung der Aktivitäten bleibt jedoch den einzelnen Teams und Mitarbeitern überlassen.
- **Explorationsiteration**: Ziel eines Explorationssprints ist es, mit Hilfe von organisierten Experimenten das zur Umsetzung von Anforderungen erforderliche Wissen zu erlangen und damit die Umsetzungsrisiken zu verringern. Der wesentliche Unterschied zu norma len Iterationen besteht also darin, dass kein Produktivcode entsteht. Eine Explorationsite-

ration wird immer dann eingeschoben, wenn die Entwicklungsrisiken zu groß für eine Standarditeration sind.

- **Releaseiteration**: Manchmal ist es schwierig, alle erforderlichen Deploymentaktivitäten in jedem Sprint vorzunehmen, und so dem Produktverantwortlichen am Ende eines jeden Sprints zu erlauben, die Software auszuliefern. Dies kann etwa dann der Fall sein, wenn eine kundenspezifische Konfiguration der Software erforderlich ist, oder die Umstände eine umfangreiche Integration mit Hardwareelementen verlangen[167]. In diesem Fall kann es zweckmäßig sein, vor Ende des Releases eine Releaseiteration explizit einzuplanen. In dieser Releaseiteration werden dann alle auslieferungsbezogen Aktivitäten ausgeführt. Es wird jedoch empfohlen, auf Releaseiterationen zu verzichten und die Auslieferungsaktivitäten in die Standarditerationen zu integrieren.

4.2.3.4 Iteration Review Meeting

Das Iteration Review Meeting[168] findet immer im Anschluss an eine Iteration statt. Es hat den Zweck, die in dem abgelaufenen Sprint fertig gestellten Funktionalitäten dem Produktverantwortlichen vorzustellen und dessen Fragen zu den fertig gestellten Funktionalitäten zu beantworten. Soweit zweckmäßig, können weitere Stakeholder zu dieser Besprechung eingeladen werden. Fertig bedeutet in diesem Fall, dass die Funktionalität fertig entwickelt und getestet ist, und damit prinzipiell ausgeliefert werden kann. Der Produktverantwortliche wird dann mit dem Team und den Stakeholdern an Hand des Feedbacks mögliche Änderungen am Product Backlog besprechen. Das Iteration Review Meeting soll nicht mehr als vier Stunden dauern.

4.2.3.5 Iteration-Retrospektive

Die Iteration-Retrospektive[169] findet immer unmittelbar im Anschluss an das Iteration Review Meeting statt. Die Iteration-Retrospektive hat die Aufgabe, den Ablauf der letzten Iteration kritisch zu beleuchten und Verbesserungspotential zu identifizieren. An der Sprint-Retrospektive nimmt das gesamte Entwicklungsteam teil. Das Team legt dann mögliche Verbesserungen fest und priorisiert sie. Der Chefentwickler ist zusammen mit dem Qualitätsbeauftragten für die Umsetzung zuständig.

---

[167] Aus eigener Erfahrung weiß ich etwa über die Erstellung der kompletten Betriebssoftware einer Protonentherapieanlage zu berichten. In diesem Fall wird zunächst eine mehrtägige Installation benötigt, an die sich ein einwöchiger Integrationstest anschließt, durch den sichergestellt wird, dass alle Soft- und Hardwaresysteme einwandfrei zusammenarbeiten.

[168] Dieses Meeting basiert auf Scrum, s. Kapitel 3.2.3.4.

[169] Dieses Meeting basiert ebenfalls auf Scrum, s. Kapitel 3.2.3.5

4.2.3.6 Release freigeben

Während der Konstruktionsphase werden die Anforderungen in der Reihenfolge umgesetzt, die der Produktverantwortliche für sinnvoll und zweckmäßig hält. Nachdem alle Iterationen eines Releases durchgeführt wurden, erfolgt noch eine Überprüfung, ob wirklich alle erforderlichen Aktivitäten korrekt abgeschlossen wurden. Dies beinhaltet auch eine Überprüfung der erforderlichen Dokumente wie der Anforderungsdokumentation, dem Verifizierungsplan und vorgesehenen Risikobeherrschungsmaßnahmen. Diese Überprüfung erfolgt in der Form eines Reviews. Sofern Abweichungen gefunden werden, werden diese bewertet. Finden sich nicht tolerierbare Abweichungen wird eine weitere Iteration eingeschoben, um die gefundenen Abweichungen zu beseitigen. Nachdem das Release freigegeben wurde, kann das Produkt an den Kunden zum Testen übergeben werden. Das Team kann mit der Entwicklung des nachfolgenden Releases beginnen oder die Konstruktionsphase beenden und mit der Übergabephase beginnen.

4.2.4 Übergabephase

In dieser Phase haben wir erstmalig die gesamte Funktionalität zur Verfügung, die benötigt wird, um das Medizinprodukt im Produktivbetrieb zu benutzen. Bevor dies aber geschehen kann, müssen wir durch einen Abnahmetest und durch die Validierung des Produktes den Nachweis erbringen, dass das Produkt für den beabsichtigten Zweck geeignet ist. Weiterhin werden wir regelmäßig spätestens zu diesem Zeitpunkt nochmals alle Dokumentationen auf Vollständigkeit prüfen. Dies beinhaltet auch Dokumentationen, die während der Entwicklung nicht unwichtig, aber von geringerer Bedeutung waren, wie Schulungs- und Wartungsunterlagen.

4.2.4.1 Validierung

Durch die Validierung soll überprüft werden, inwieweit das Medizingerät Benutzeranforderungen erfüllt, für den beabsichtigten Gebrauch geeignet ist und inwieweit die verbleibenden Restrisiken den vorgegebenen Abnahmekriterien genügen. Um diesen Nachweis zu erbringen ist es erforderlich, einen Validierungsplan zu erstellen, und die Validierung nach diesem Plan durchzuführen. Der Validierungsplan muss die Abhängigkeiten zwischen der Validierung und Entwicklung beschreiben und Verfahren zur Validierung der Benutzerschnittstelle müssen eingeschlossen werden.

4.2.4.2 Freigabe

Bevor die Software eingesetzt werden kann, muss sie freigegeben werden. Bevor die Software freigegeben werden darf, muss sichergestellt werden, dass die Verifizierung der Software vollständig ist, nicht gelöste Anomalien bewertet und dokumentiert wurden und die Dokumentation

vollständig ist. Zudem muss sichergestellt sein, dass die Software-Freigabe bei Bedarf wiederholt werden kann.

4.2.5 Wartungsphase

Die Wartungsphase dient der Fehlerbeseitigung und Weiterentwicklung des bereits eingesetzten Produktes. Die Aktivitäten innerhalb der Wartungsphase decken sich im Wesentlichen mit denen der Konstruktionsphase. Es gibt jedoch einen wesentlichen Unterschied: Während der Konstruktionsphase ist ein fertig gestelltes Release nicht für den Produktivbetrieb vorgesehen, in der Wartungsphase hingegen schon. Daher schließen sich in der Wartungsphase an die Fertigstellung eines Releases immer die Aktivitäten der Übergangsphase an. Die Aktivitäten können in der Wartungsphase aber durchaus parallel ausgeführt werden.

4.3 Artefakte

Dieses Kapitel gibt einen Überblick über alle innerhalb des Entwicklungsprozesses erstellten Artefakte. In diesem Kapitel werden lediglich die Artefakte und deren Eigenschaften beschrieben, der Kontext, in dem sie erstellt werden, wird im nachfolgenden Kapitel näher beleuchtet.

4.3.1 Produktkonzept

Das Produktkonzept fungiert zunächst als Entscheidungsgrundlage, ob das Entwicklungsprojekt durchgeführt werden soll. Nach der Entscheidung, das Projekt durchzuführen, dient es häufig als Visionsdokument. Wichtige Bestandteile des Produktkonzeptes sind:

- wichtige Kundenanforderungen an das neue Produkt,
- gesetzliche und regulatorischen Anforderungen,
- organisatorische Randbedingungen,
- beabsichtigter Gebrauch des Produktes,
- Übersicht über die vorgesehen Grätefunktionen mit typischen Szenarien, in denen das Gerät benutzt wird,
- Bewertung der ermittelten Anforderungen.

4.3.2 Entwicklungsplan

Der Entwicklungsplan dokumentiert alle erforderlichen Vorgänge, die bei der Entwicklung des Software-Systems verwendet werden. Dabei ist auch festzulegen, wie die Aktivitäten und Auf-

gaben anderer Prozesse die Entwicklung beeinflussen oder in die Entwicklung integriert sind. Zudem sind auch die bei der Ausführung der Prozesse anfallenden Produkte festzulegen und zu dokumentieren. Es wird also der gesamte Entwicklungsprozess detailliert dokumentiert. Dabei kann auf andere Dokumente Bezug genommen werden. Der Entwicklungsplan besteht daher aus den folgenden Teilen:

- Die organisatorische Struktur des Projektes mit Zeitrahmen, Ressourcen und Personal sowie Verantwortlichkeiten,
- Die zeitliche Aufteilung des Projektes in Phasen mit den zugehörigen Meilensteinen,
- Die eingesetzten Normen, Methoden und Werkzeuge,
- Die Referenz auf den Verifikations- und Validierungsplan,
- Die Referenz auf eine Beschreibung für das eingesetzte Problemlösungsverfahren. Das Problemlösungsverfahren wird in diesem Vorgehensmodell beschrieben,
- Die Referenz auf eine Beschreibung für das eingesetzte Änderungsmanagement. Das Änderungsmanagement wird in diesem Vorgehensmodell beschrieben,
- Die Referenz auf eine Beschreibung für das eingesetzte Konfigurationsmanagement. Das Konfigurationsmanagement wird in diesem Vorgehensmodell beschrieben,
- Die Referenz auf eine Beschreibung für das eingesetzte Risikomanagement. Das Risikomanagement wird in diesem Vorgehensmodell beschrieben,
- Die Referenz auf eine Beschreibung in der die Beziehung zwischen Anforderungen, Verifizierungen und Risikokontrollmaßnahmen beschrieben wird,
- Die Referenz auf die Anforderungen an das zu entwickelnde Gerät.

Zu Beginn der Entwicklung werden üblicherweise nicht alle Informationen vorliegen, oder zumindest nicht vollständig vorliegen. Während der Entwicklung können diese Informationen ergänzt werden, sobald sie vorliegen. Auch alle sich während der Entwicklung ergebenden Änderungen werden in dem Entwicklungsplan dokumentiert werden. Der Entwicklungsplan ist daher das zentrale Dokument, wenn ermittelt werden soll, was und wie das Gerät hergestellt wird. Ein Großteil der Anforderungen an dieses Dokument ist regulatorischen Anforderungen geschuldet.

4.3.3 Verifikations- und Validierungsplan

Der Verifizierungsplan legt die Rahmenbedingungen fest, unter denen die Verifizierung stattfindet. Wesentliche Angaben in diesem Dokument sind:

- Zu testende Objekte mit den zugehörigen Leistungsmerkmalen,

- die eingesetzte Teststrategie,
- Abnahmekriterien,
- Kriterien für den Testabbruch und die Testfortsetzung,
- Verantwortlichkeiten und Zuständigkeiten,
- Benötigte Ressourcen und Personal sowie
- Planungsrisiken.

Der Validierungsplan legt analog die Rahmenbedingungen fest, unter denen die Validierung stattfindet. Er ist strukturell dem Verifikationsplan sehr ähnlich.

4.3.4 Risikomanagementplan

Der Risikomanagementplan legt die Rahmenbedingungen fest, unter denen der Risikomanagementprozess durchgeführt wird. Der Risikomanagementplan muss Folgendes enthalten:

- Anwendungsbereich des Planes, wobei insbesondere die Lebenszyklusprozesse aufzuführen sind, für die der Plan gilt,
- einen Verifizierungsplan,
- die Anforderungen an die Bewertung der Risikoaktivitäten,
- Kriterien für die Vertretbarkeit von Risiken und
- die Zuordnung von Verantwortlichkeiten.

Die Bestandteile des Risikomanagementplans sind im Wesentlichen den Anforderungen aus [ISO 14971] geschuldet.

4.3.5 Konzeptionelle Systemarchitektur

Die konzeptionelle Architektur wird zu Beginn der Entwicklung erstellt. Sie enthält eine Strukturierung des Gesamtsystems in fachliche Komponenten, die die zentralen fachlichen Abhängigkeiten dokumentieren. Die konzeptionelle Architektur kann durch Anmerkungen ergänzt werden, die erläutern, warum gewisse Modellierungsentscheidungen getroffen wurden. Durch Sequenzdiagramme kann dynamisches Verhalten erläutert werden, soweit dies angemessen erscheint. Die konzeptionelle Architektur dient als Basis von Design und Implementierung und wird während der Entwicklung regelmäßig aktualisiert und ergänzt.

4.3.6 Infrastrukturdokument

In diesem Dokument wird die Infrastruktur für das Entwicklungsteam festgelegt. Dies beinhaltet die Festlegungen für die Versionskontrolle, das Verfahren für das tägliche Build, das Verfahren für das automatisierte Ausführen von Modul- und Komponententests und weiter damit in Zusammenhang stehenden Tätigkeiten.

4.3.7 Product Backlog

Das Product Backlog enthält alle priorisierten Anforderungen des zu erstellenden Anwendungssystems. Das Backlog enthält außer den funktionalen Anforderungen alle nicht-funktionalen Anforderungen, wie Skalierbarkeit, Robustheit, Performanz und Benutzbarkeit. Die Anforderungen im Backlog sind stets priorisiert.

4.3.8 Release Plan

In dem Releaseplan wird dokumentiert welche Funktionen in dem Release enthalten sein sollen. Die Funktionen werden im Releaseplan nicht ausführlich dokumentiert, sondern enthalten lediglich Referenzen auf Einträge in dem Product Backlog. Der Releaseplan enthält weiterhin eine Planung über die Anzahl der benötigten Iteration und eine vorläufige Zuordnung der Funktionen zu den Iterationen. Zudem enthält der Releaseplan einen Meilenstein mit dem geplanten Fertigstellungsdatum des Release.

4.3.9 Iterationsplan

Der Iterationsplan enthält alle Tätigkeiten, die erforderlich sind, um das Iterationsziel zu erreichen. Die Tätigkeiten sind in Personenstunden geschätzt. Der Iterationsplan wird zu Beginn einer Iteration erstellt, und an jedem Arbeitstag aktualisiert. Damit erlaubt er eine genaue Verfolgung der in der Iteration durchgeführten Aktivitäten. Der Iterationsplan dient als Basis für den Iteration-Burndown-Bericht.

4.3.10 Product Burndown Bericht

Der Product-Burndown-Bericht zeigt den Fortschritt im Projekt auf. Er betrachtet, wie sich die Aufwände im Release Backlog von Woche zu Woche über das gesamte Projekt entwickeln und

zeigt diese Entwicklung grafisch in einem Chart. Der Bericht ermöglicht es, den Fortschritt innerhalb des Projektes zu verstehen und zu kommunizieren.[170]

4.3.11 Release Burndown Bericht

Der Release-Burndown-Bericht zeigt den Fortschritt im Release auf. Er betrachtet, wie sich die Aufwände im Release Backlog von Woche zu Woche entwickeln und zeigt diese Entwicklung grafisch in einem Chart. Der Bericht ermöglicht es, den Fortschritt innerhalb des Release zu verstehen und zu kommunizieren. Anhand des Berichtes lässt sich der Release-Verlauf gut nachvollziehen und Verbesserungspotenzial identifizieren.

4.3.12 Iteration Burndown Bericht

Der Iteration-Burndown-Bericht zeigt den Fortschritt in der Iteration auf. Er betrachtet, wie sich die Aufwände im Iterationsplan von Tag zu Tag entwickeln und zeigt diese Entwicklung grafisch in einem Chart. Der Bericht ermöglicht es, den Fortschritt innerhalb der Iteration zu verstehen und zu kommunizieren. Anhand des Berichtes lässt sich der Iterations-Verlauf gut nachvollziehen und Verbesserungspotenzial identifizieren. Der Bericht kann daher auch für die Iterations-Retrospektive verwenden werden.[171]

4.3.13 Trend Chart

Das Trend Chart zeigt für jede Woche, wie viele Anforderungen offen, in Arbeit und abgeschlossen sind. Das Trend Chart zeigt sowohl die wöchentlichen Werte der Vergangenheit als auch die prognostizierte Entwicklung der Umsetzung der Features für die Zukunft. Das Trend Chart wird wöchentlich vom Projektverantwortlichen aktualisiert.[172]

[170] Der Product Backlog wird weder in XP noch in Scrum verlangt. Dieser Bericht scheint dennoch sinnvoll, da so ein konsolidierter Bericht vorliegt, der den aktuellen Zustand des gesamten Projektes wiedergibt. Bei geeigneter IT-Unterstützung kann dieser Bericht automatisch erstellt werden, etwa aus den Daten des Release Backlog.

[171] Dieser Bericht ist Bestandteil von Scrum (s. Kapitel 3.2.4.3).

[172] Der Trend Chart ist Bestandteil von Scrum (s. Kapitel XXX).

4.3.14 Architektur- und Designdokument

Für das Gesamtsystem liegt bereits eine Architektur in Form der konzeptionellen Architektur vor. Für größere Systeme erfolgt häufig eine Unterteilung in Teilsysteme. Der Umfang der Dokumentation orientiert sich an der Sicherheitsklasse, die dem betreffenden System zugewiesen wurde.

4.3.15 Testspezifikation

Die im Verifikationsplan festgelegte Teststrategie bestimmt, welche Testmethoden verwendet werden. In der Testspezifikation werden dann die Testfälle unter Verwendung dieser Testmethoden spezifiziert.

4.3.16 Testprotokoll

Die Durchführung von Tests muss protokolliert werden. An Hand der Testprotokolle muss die Durchführung der Tests nachvollziehbar sein. Aus dem Testprotokoll muss hervorgehen, welche Tests von wem, wann und mit welchem Ergebnis getestet wurden.

4.3.17 Problem-Log

Für jedes identifizierte Problem muss Bericht erstellt werden. Das Problem-Log enthält alle diese berichte mit Angaben hinsichtlich Typ, Umfang und Kritikalität und Zustand der Problemmeldung.

5 Aktivitäten des Vorgehensmodells

Im vorherigen Kapitel wurden bei der Beschreibung der Phasen bereits die benötigten Aktivitäten oberflächlich beschrieben. In diesem Kapitel werden diese Aktivitäten nun systematisch dargestellt. Dadurch ergibt sich zwangsläufig ein gewisses Maß an Überschneidung. Dies wird aber in Kauf genommen, damit zunächst der Rahmen des Vorgehensmodells erläutert werden kann, bevor die Details der erforderlichen Aktivitäten dargestellt werden.

5.1 Konzeptphase

In der Konzeptphase sind lediglich zwei Aktivitäten vorgesehen, eine zum Erstellen des Produktkonzeptes, die andere zum Freigeben des Produktkonzeptes

5.1.1 Produktkonzept erstellen

In dieser Aktivität wird die Produktidee entwickelt und formuliert.

5.1.1.1 Beschreibung

Die Entwicklung eines Systems beginnt damit, dass die grundsätzliche Zielsetzung und Produktidee formuliert wird. Was soll mit dem neuen Produkt getan werden? Wodurch unterscheidet es sich von Konkurrenzprodukten? Welche Funktionen werden von Patienten und Benutzer von einem solchen Produkt gefordert? Wie soll das neue Produkt eingesetzt werden. Welche gesetzlichen und regulatorischen Beschränkungen sind zu berücksichtigen? Welche organisatorischen Beschränkungen sind durch die eigene Organisation gegeben?

5.1.1.2 Beteiligte

Der für das neue Produkt vorgesehen Produktverantwortliche erstellt das Produktkonzept und verantwortet das Ergebnis. Beteiligt sind alle Personen, die ein mögliches Interesse an dem Produkt haben, wie Marketing, Vertrieb, Produktentwicklung, Service, Wartung.

5.1.1.3 Voraussetzungen

Keine.

5.1.1.4 Ergebnisse

Das Produktkonzept.

5.1.2 Projekt freigeben

In dieser Aktivität wird entschieden, ob das im Produktkonzept beschriebene Produkt entwickelt werden soll.

5.1.2.1 Beschreibung

Das erstellte Produktkonzept wird einer Bewertung unterzogen. Basierend auf den Möglichkeiten der Organisation und der Tragfähigkeit des Produktkonzepts wird dann entschieden, ob das Produkt erstellt werden soll. In diesem Fall wird die Projektfreigabe erteilt.

5.1.2.2 Beteiligte

Produktverantwortlicher und Management im erforderlichen Umfang.

5.1.2.3 Voraussetzungen

Das fertig gestellte Produktkonzept.

5.1.2.4 Ergebnisse

Die Projektfreigabe

5.2 Planungsphase

In der Planungsphase erfolgt eine zunächst grobe Strukturierung des zukünftigen Projektablaufs. Zudem werden wichtige Festlegungen getroffen, die es erlauben, in der nachfolgenden Phase die Entwicklung des Produktes durchzuführen.

5.2.1 Entwicklung planen

In dieser Aktivität werden die wesentlichen Rahmenbedingungen für die Entwicklung des Projektes definiert.

5.2.1.1 Beschreibung

Der Qualitätsverantwortliche und der Chefentwickler erstellen den Entwicklungsplan. In dem Entwicklungsplan werden zunächst die organisatorische Struktur des Projektes mit Zeitrahmen,

Ressourcen, Personal und Verantwortlichkeiten festgelegt. Weiterhin werden die eingesetzten Normen, Methoden und Werkzeuge definiert und in dem Entwicklungsplan dokumentiert. Der Entwicklungsplan wird zusammen mit dem Produktverantwortlichen einem Review unterzogen und dann gemeinsam durch das Projektkernteam freigegeben.

5.2.1.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlichen
- Chefentwickler

5.2.1.3 Voraussetzungen

- Die Projektfreigabe ist erfolgt.

5.2.1.4 Ergebnisse

- Der freigegebene Entwicklungsplan

5.2.2 Anforderungen ermitteln

In dieser Aktivität werden die ersten Anforderungen systematisch ermittelt,

5.2.2.1 Beschreibung

Bevor mit der Entwicklung begonnen werden kann, muss ein gewisser Umfang an Anforderungen vorliegen, um einen hinreichend präzisen Eindruck von dem zu entwickelnden System zu erlangen. Alle Anforderungen werden in das Product Backlog eingestellt. Alle Anforderungen werden so präzise und umfangreich dokumentiert, wie dies notwendig ist. Eine skizzenhafte Beschreibung durch User Stories, wie dies in XP üblich ist, erscheint aus regulatorischen Gründen nicht zweckmäßig[173]. Es wird hier aber keine bestimmte Art und Weise der Dokumentation von Anforderungen vorgeschrieben. Es sind Kundenanforderungen, Anforderungen an die Gebrauchstauglichkeit, regulatorische Anforderungen und nicht-funktionale Anforderungen, wie

[173] Die regulatorischen Vorgaben verlangen, dass die Anforderungen systematisch ermittelt, dokumentiert und bewertet werden. Die so ermittelten Anforderungen sind auf Vollständigkeit, Eindeutigkeit und Widerspruchsfreiheit zu prüfen. Zudem sollen die Anforderungen in Worten ausgedrückt werden, die Mehrdeutigkeit vermeiden und es ermöglichen, dass Prüfungskriterien festgelegt werden und Prüfungen durchgeführt werden, um festzustellen, ob die Prüfungskriterien eingehalten werden

Leistungsfähigkeit, zu berücksichtigen. Der Produktverantwortliche ist zuständig für die Ermittlung aller Anforderungen.

Um Missverständnissen zuvorzukommen, sei ausdrücklich darauf hingewiesen, das die Anforderungen im Product Backlog die Sprache des Kunden und Anwenders entsprechen. Es handelt sich also nicht um konkrete Anforderungen an das System oder ein Teilsystem. Diese Systemanforderungen werden erst zu einem späteren Zeitpunkt aus den Anforderungen des Product Backlog abgeleitet.

Hingegen sind Anforderungen an die Gebrauchstauglichkeit Bestandteil des Product Backlog. Für die Gebrauchstauglichkeit des Systems ist die Kenntnis über die verschiedenen Benutzergruppen des Gerätes, die übliche Geräteverwendung, aber auch mögliche, aber ungewöhnliche Arten der Geräteverwendung, die funktionalen Eigenschaften des Gerätes, die Eigenschaften der Umgebung, unter denen das Gerät betrieben wird und die Beziehungen zwischen Benutzer, Umgebung und Gerät zu ermitteln, zu dokumentieren und zu bewerten.

Die Festlegung von Softwareanforderungen schließt mit einem Review einschließlich einer ersten Risikoanalyse des Medizingerätes ab. Dabei werden die risikobehafteten Funktionen identifiziert, bewertet und Risiko-Beherrschungsmaßnahmen festgelegt, die die identifizierten Risiken ausreichend verringern. Für das Gesamtsystem wird eine Sicherheitsklasse festgelegt. Das Ergebnis dieses Reviews ist ein freigegebenes Product Backlog mit einer Festlegung des beabsichtigten Gebrauchs.

5.2.2.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler
- Risikomanager

5.2.2.3 Voraussetzungen

- Die Projektfreigabe ist erfolgt

5.2.2.4 Ergebnisse

- Freigegebenes Product Backlog
- Risikomanagementakte

5.2.3 Konzeptionelle Architektur erstellen

In dieser Aktivität wird eine fachliche Strukturierung der Anforderungen aus dem Product Backlog vorgenommen.

5.2.3.1 Beschreibung

Nach Festlegung der ersten fachlichen Anforderungen des zu erstellenden Systems kann die konzeptionelle Strukturierung des Systems in fachliche Komponenten erfolgen, die die zentralen fachlichen Abhängigkeiten dokumentieren. Durch Sequenzdiagramme kann dynamisches Verhalten ergänzt werden, soweit dies zweckmäßig erscheint. Durch Text werden weitere Anforderungen an das System präzisiert. Die konzeptionelle Architektur dient als Basis von Design und Implementierung und wird während der Entwicklung regelmäßig aktualisiert. Die konzeptionelle Architektur übersetzt die Anforderungen in konkrete Anforderungen an das System.[174]

Die Erstellung der konzeptionellen Architektur schließt mit einem Review ab. Das Ergebnis dieses Reviews ist eine freigegebene konzeptionelle Architektur.

5.2.3.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler
- weitere Entwickler soweit erforderlich

5.2.3.3 Voraussetzungen

- Product Backlog freigegeben.

5.2.3.4 Ergebnisse

- Konzeptionelle Architektur

[174] Um keine Techniken und Methoden an dieser Stelle zu fordern, bleiben die Anforderungen an die konzeptionelle Architektur zwangsläufig etwas unpräzise. Gewollt ist eine Umsetzung der Anforderungen aus dem Product Backlog in Anforderungen an das System, die hinreichend präzise sind, um das System zu erstellen. Andererseits ist gerade kein BDUF, kein Big Design Up Front, gemeint. Die Strukturierung erfolgt nur soweit, wie es für die Entwicklung erforderlich ist. Deshalb wird diese Aufgabe auch nicht von Architekten übernommen, sondern von Entwicklern ausgeführt, die später auch Entwicklungsaufgaben an dem System übernehmen.

5.2.4 Infrastruktur festlegen

In dieser Aktivität wird Infrastruktur festgelegt, die bei der Entwicklung genutzt wird.

5.2.4.1 Beschreibung

Die Entwicklung von Software benutzt eine Vielzahl von Infrastrukturkomponenten, wie etwa eine Software zur Versionsverwaltung, eine Software zur Aufgabenverwaltung, integrierte Entwicklungsumgebungen, zum Editieren und Kompilieren von Quellcode und zum Debuggen von fehlerhaftem Code, Quellcodegeneratoren, Werkzeuge zum Projektmanagement und Projektreporting und vieles mehr. Welche Werkzeuge bei der Entwicklung eingesetzt werden, wird in dieser Aktivität festgelegt.

5.2.4.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler
- weitere Entwickler soweit erforderlich

5.2.4.3 Voraussetzungen

- Die Projektfreigabe ist erfolgt

5.2.4.4 Ergebnisse

- Infrastrukturdokument mit der Dokumentation der benötigten und eingesetzten Infrastruktur

5.3 Konstruktionsphase

In dieser Phase wird das neue Produkt erstellt.

5.3.1 Anforderungen anpassen

In dieser Aktivität werden neue Anforderungen in das Product Backlog eingestellt oder bereits eingestellte Anforderungen modifiziert.

5.3.1.1 Beschreibung

Der Produktverantwortliche ist allein zuständig und berechtigt neue Anforderungen in das Produkt Backlog einzustellen oder bestehende Anforderungen zu ändern. Dazu kommuniziert er regelmäßig mit allen Stakeholdern, wie Kunden und Benutzern. Bei der Einstellung der Anforderungen werden die Anforderungen durch den Produktverantwortlichen priorisiert und einem Release zugeordnet. Neue Anforderungen dürfen nicht dem aktuellen Release zugeordnet werden.

5.3.1.2 Beteiligte

- Produktverantwortlicher
- Stakeholder

5.3.1.3 Voraussetzungen

- Keine

5.3.1.4 Ergebnisse

- Geändertes Product Backlog

5.3.2 Release planen

Die Entwicklung des Produktes erfolgt in mehreren durch Meilensteine abgegrenzte Releases, die sich aus mehreren Iterationen zusammensetzen. In dieser Aktivität wird ein einzelnes Release geplant.

5.3.2.1 Beschreibung

Bereits in der Entwicklungsplanung wurden die Ziele für jedes Release festgelegt. Zu Beginn eines jeden Releases werden in einem Planungsmeeting

→ diese Anforderungen und Ziele weiter konkretisiert,

→ die Umsetzung dieser Ziele auf mehrere Iterationen geplant.

→ der Meilenstein für die Fertigstellung des Releases geplant und

→ die Anforderungen auf mögliche Risiken analysiert.

Die Ergebnisse des Planungsmeetings werden in dem Releaseplan dokumentiert. Der Entwicklungsplan wird angepasst. In den Fällen, in denen Unsicherheit hinsichtlich der Umsetzung besteht, können zusätzlich eine oder mehrere Explorationsiterationen zu Beginn vorgesehen wer-

den. Der Produktverantwortliche ist für die Erstellung und Aktualisierung des Releaseplans zuständig.

5.3.2.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler (bei Bedarf)
- Risikomanager
- Konfigurationsmanager

5.3.2.3 Voraussetzungen

- Vorheriges Release abgeschlossen

5.3.2.4 Ergebnisse

- Releaseplan
- aktualisierter Entwicklungsplan

5.3.3 Iteration planen

Die Umsetzung der für ein Release vorgesehenes Release erfolgt in mehreren Iterationen. In dieser Aktivität werden die für eine Iteration vorgesehenen Tätigkeiten geplant.

5.3.3.1 Beschreibung

Eine Iteration hat üblicherweise immer die gleiche Länge. In Ausnahmefällen kann davon abgewichen werden. Jede Iteration beginnt mit einem Planungsmeeting. Das Planungsmeeting besteht aus zwei Teilen.

Im ersten Teil wird festgelegt, welche Funktionalitäten in der nächsten Iteration umgesetzt werden sollen. Dazu analysiert das gesamte Team die für die nächste Iteration vorgesehene Funktionalität, wie sie im Releaseplan beschrieben ist. Der Produktverantwortliche erläutert bei Bedarf unklare Punkte. Für die umzusetzenden Funktionalitäten werden nochmals möglich Risiken betrachtet. Der Produktverantwortliche entscheidet alleine über die Funktionalität, die umgesetzt werden soll. Das Entwicklungsteam entscheidet gemeinsam, wie viele der Funktionalitäten in der Iteration umsetzbar sind. Der Qualitätsbeauftragte und der Chefentwickler haben bei dieser Entscheidung kein Mitspracherecht. Der erste Teil des Planungsmeetings soll vier Stunden nicht

überschreiten. Andernfalls muss das Meeting unterbrochen und am nächsten Tag fortgesetzt werden.

Im zweiten Teil des Planungsmeetings wird das Team nun die Umsetzung der vorgesehen Anforderungen planen. Dabei werden die einzelnen Anforderungen in Aufgaben heruntergebrochen, in eine Reihenfolge gebracht und Teammitgliedern zugewiesen. Diese Aufteilung erfolgt durch das Entwicklungsteam und den Chefentwickler. Die verbleibenden Teammitglieder haben dabei lediglich beratende Funktion. Die Aufteilung in Aufgaben kann nicht erschöpfend sein, aber doch so filigran erfolgen, das entschieden werden kann, ob die anstehenden Aufgaben in der nächsten Iteration umgesetzt werden können. Auch der zweite Teil des Planungsmeetings soll vier Stunden nicht überschreiten. Andernfalls muss das Meeting unterbrochen und am nächsten Tag fortgesetzt werden.

5.3.3.2 Beteiligte

- Das gesamte Team

5.3.3.3 Voraussetzungen

- Eine neue Iteration soll begonnen werden.

5.3.3.4 Ergebnisse

- Iterationsplan, mit den Aufgaben für die nächste Iteration
- aktualisierter Releaseplan
- aktualisierter Entwicklungsplan
- aktualisierter Konfigurationsplan
- aktualisierte Risikomanagementakte

5.3.4 Anforderungen analysieren

In dieser Aktivität werden Anforderungen aus dem Product Backlog analysiert und in Systemanforderungen übersetzt.

5.3.4.1 Beschreibung

Bevor die Anforderungen aus dem Product Backlog in ein Iterationsinkrement umgesetzt werden kann, müssen die Anforderungen erst in Anforderungen an das System umgesetzt werden. Teilweise ist dies bereits bei der Erstellung der konzeptionellen Architektur geschehen. Teilweise müssen die Angaben dort aber weiter konkretisiert, bei geänderten Anforderungen modifiziert

und bei neuen Anforderungen ergänzt werden. Weiterhin müssen die Systemtests für diese Funktionalitäten erstellt werden. Zudem müssen nochmals mögliche Risiken für Patienten und Anwender betrachtet werden.

Die Aufgabe erfolgt durch zwei Entwickler, zweckmäßiger werden dies bereits diejenigen Entwickler sein, die betrachtete Funktionalität auch umsetzen oder zumindest daran beteiligt sind. Bei Fragen kann der Produktverantwortliche, der Qualitätsverantwortliche oder der Chefentwickler zugezogen werden. Die Aufgabe schließt mit einem Review ab, an dem die beiden Entwickler, der Chefentwickler, der Qualitätsverantwortliche und der Risikomanager teilnehmen.

5.3.4.2 Beteiligte

- Entwickler (2)
- Risikomanager (teilweise)
- Chefentwickler (bei Bedarf)
- Produktverantwortlicher (bei Bedarf)
- Qualitätsverantwortlicher (bei Bedarf)

5.3.4.3 Voraussetzungen

- Aktivität wurde in dem Iterationsplan geplant

5.3.4.4 Ergebnisse

- Aktualisierte konzeptionelle Architektur
- Aktualisierte Testspezifikation
- Aktualisierte Risikomanagementakte

5.3.5 Lösung entwerfen, implementieren und testen

In dieser Aktivität wird ein Produktinkrement entworfen, erstellt und getestet.

5.3.5.1 Beschreibung

In dieser Aktivität wird das Produktinkrement erstellt. Zu der Erstellung gehören zwangsläufig auch der Entwurf und der Test des Inkrements. Der Umfang der Entwurfsdokumentation richtet sich dabei nach der Sicherheitsklassifikation des betreffenden Teilsystems. Je nachdem ist nur eine Dokumentation der technischen Architektur, oder aber eine detaillierte Entwurfdokumentation erforderlich. Es wird aber nicht verlangt, dass bei der Erstellung sequentiell vorgegangen

wird, und der Entwurf daher abgeschlossen sein muss, bevor die Implementierungsphase beginnt. Es wird vielmehr, ganz in Sinne agiler Entwicklung, unterstellt, dass alle Teilaktivitäten, also entwerfen, implementieren und testen mehr oder weniger parallel ausgeführt werden. Im welchem Ausmaß dies im Einzelnen geschieht wird den beiden Hauptakteuren überlassen[175]. Auch ist es nicht wichtig, ob die Modultests nun vor, während oder nach der Implementierung der zugehörigen Funktionalität erstellt werden. Wichtig ist, dass sie erstellt werden und zum Ende der Aktivität fehlerfrei ausführbar sind. Der Fertigstellungsstand der Aktivität wird täglich in dem Iterationssplan aktualisiert.[176]

5.3.5.2 Beteiligte

- Entwickler (2)
- Chefentwickler (bei Bedarf)
- Produktverantwortlicher (bei Bedarf)
- Qualitätsverantwortlicher (bei Bedarf)

5.3.5.3 Voraussetzungen

- Iterationsplanung sieht die Umsetzung dieser Funktionalität in dieser Iteration vor

5.3.5.4 Ergebnisse

- Fertiggestelltes Produktinkrement
- Aktualisierter Iterationsplan
- Fertiggestellte und lauffähige Modultests
- Fertiggestellte Testspezifikation für Modultest
- Architektur- und Designspezifikation

[175] Ich bin nicht wirklich von den Vorzügen des Pair Programming überzeugt. Die Erstellung des Produktinkrements mit anschließendem Codereview, wie in FDD üblich, scheint zumindest eine praktikable Alternative zu sein.

[176] Die Burndown-Berichte wie auch der Trend-Chart sollten auf dieser Basis im Iterationsplan und Releaseplan automatisch erstellt werden. Falls dies nicht möglich ist, muss noch eine PM-Aktivität für die Erstellung dieser Berichte ergänzt werden.

5.3.6 Systemtest erstellen

In dieser Aktivität werden die Systemtests erstellt.

5.3.6.1 Beschreibung

Die Systemtests werden, analog zu dem Produkt, ebenfalls inkrementell erstellt. Es werden daher üblicherweise in der Iteration, in der ein Produktinkrement erstellt wird, auch die dafür erforderlichen Systemtests erstellt werden. Sollte dies aus organisatorischen Gründen nicht möglich sein, sollten die Systemtests spätestens zum Ende der nachfolgenden Iteration fertig gestellt sein. Systemtests müssen vor dem Abschluss des Releases verfügbar sein. Fallen bei der Erstellung Lücken oder Fehler in der Testspezifikation auf, muss die Testspezifikation korrigiert werden.

5.3.6.2 Beteiligte

- Tester
- Entwickler
- Qualitätsbeauftragter

5.3.6.3 Voraussetzungen

- Das Produktinkrement mit der zu testenden Funktionalität wird erstellt

5.3.6.4 Ergebnisse

- Systemtest
- Korrigierte Testspezifikation

5.3.7 Systemtest durchführen

In dieser Aktivität werden die erstellten Systemtests durchgeführt

5.3.7.1 Beschreibung

Nach der Erstellung bzw. Ergänzung des Systemtests kann dieser ausgeführt werden. Bei der inkrementellen Entwicklung des Produktes ist es einfach, Fehler in das Produkt hineinzubringen. Daher sollen die Systemtests so konzipiert werden, dass sie möglichst automatisch ausgeführt werden können. Durch kontinuierliche Integration der fertig gestellten Produktinkremente kann

so regelmäßig die Funktionsfähigkeit des Produktes auf der Ebene der (Teil-) systeme sichergestellt werden. Die Ergebnisse des Systemtests werden in einem Testprotokoll dokumentiert.

5.3.7.2 Beteiligte

- Tester

5.3.7.3 Voraussetzungen

- Produktinkrement fertig gestellt
- Systemtest erstellt

5.3.7.4 Ergebnisse

- Testprotokoll

5.3.8 Probleme behandeln

In dieser Aktivität werden identifizierte Probleme gehandelt.

5.3.8.1 Beschreibung

Jedes identifizierte Problem wird in das Problem-Log eingetragen. Dies gilt auch für Anomalien, die während des Systemtest und der Validierung gefunden werden. Das Kernteam trifft sich einmal wöchentlich, um die die offenen Berichte des Problem-Logs nach Typ, Umfang und Kritikalität zu klassifizieren und zu bewerten.

Das Problem muss untersucht werden und, soweit möglich, die Ursachen des Problems identifiziert werden. Danach muss entschieden werden, ob und wie das Problem gelöst werden soll. Sofern Änderungen erforderlich sind, ist das definierte Änderungsverfahren zu verwenden (s. Kapitel 5.3.1).

5.3.8.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler
- Entwickler (bei Bedarf)
- Tester (bei Bedarf)

5.3.8.3 Voraussetzungen

- Jederzeit möglich

5.3.8.4 Ergebnisse

- Geändertes Problem-Log
- Geändertes Product Backlog

5.3.9 Iteration Review Meeting

In dieser Aktivität werden die in einer Iteration erstellten Produktinkremente dem Produktverantwortlichen vorgestellt.

5.3.9.1 Beschreibung

Das Iteration Review Meeting findet immer im Anschluss an eine Iterarion statt. Es hat den Zweck, die in dem abgelaufenen Sprint fertig gestellten Funktionalitäten dem Produktverantwortlichen vorzustellen und dessen Fragen zu diesen Funktionalitäten zu beantworten. Soweit zweckmäßig, können weitere Stakeholder zu dieser Besprechung eingeladen werden. Der Produktverantwortliche wird mit dem Team und den Stakeholdern an Hand des Feedbacks mögliche Änderungen an den Anforderungen im Product Backlog besprechen. Das Iteration Review Meeting soll nicht mehr als vier Stunden dauern. Die Ergebnisse des Review Meeting werden dokumentiert.

5.3.9.2 Beteiligte

- Alle Teammitglieder
- Weitere Stakeholder bei Bedarf

5.3.9.3 Voraussetzungen

- Iteration abgeschlossen

5.3.9.4 Ergebnisse

- Iteration Review Protokoll

5.3.10 Iteration Retrospektive

In dieser Aktivität wird das Vorgehen innerhalb der Iteration hinterfragt und Verbesserungspotential identifiziert.

5.3.10.1 Beschreibung

Die Iteration-Retrospektive findet immer unmittelbar im Anschluss an das Iteration Review Meeting statt. Die Iteration -Retrospektive hat die Aufgabe, den Ablauf der letzten Iteration kritisch zu beleuchten und Verbesserungspotential zu identifizieren. An der Sprint-Retrospektive nimmt das gesamte Entwicklungsteam teil. Das Team legt dann mögliche Verbesserungen fest und priorisiert sie. Der Chefentwickler ist zusammen mit dem Qualitätsbeauftragten für die Umsetzung zuständig. Die Ergebnisse des Review Meetings werden dokumentiert. Nach mehreren Iterationen ist es durchaus möglich, das dieses Meeting sehr kurz ist, und nur wenige oder gar keine Verbesserungen gefunden werden.

5.3.10.2 Beteiligte

- Alle Teammitglieder

5.3.10.3 Voraussetzungen

- Iteration abgeschlossen

5.3.10.4 Ergebnisse

- Iteration Retrospektive Protokoll

5.3.11 Release freigeben

In dieser Aktivität wird das Release begutachtet und anschließend ggf. freigegeben.

5.3.11.1 Beschreibung

Nachdem alle Iterationen eines Releases durchgeführt wurden, erfolgt eine Überprüfung, ob wirklich alle erforderlichen Aktivitäten korrekt abgeschlossen wurden und die geforderte Funktionalität in der geforderten Qualität zur Verfügung steht. Dies beinhaltet auch eine Überprüfung der benötigten Dokumente. Diese Überprüfung erfolgt in der Form eines Reviews. Sofern Abweichungen gefunden werden, werden diese bewertet. Finden sich nicht tolerierbare Abweichungen wird eine weitere Iteration eingeschoben, um die gefundenen Abweichungen zu beseitigen. Es wird weiterhin geprüft, ob alle Risikobeherrschungsmaßnahmen korrekt umgesetzt und

alle Konfigurationselemente vollständig identifiziert wurden. Das Ergebnis wird in dem Releaseplan notiert. Nachdem das Release freigegeben wurde, kann das Produkt an den Kunden zum Testen übergeben werden.

5.3.11.2 Beteiligte

- Alle Teammitglieder

5.3.11.3 Voraussetzungen

- Alle Iterationen eines Releases abgeschlossen

5.3.11.4 Ergebnisse

- aktualisierter Releaseplan
- aktualisierter Entwicklungsplan
- aktualisierter Konfigurationsplan
- aktualisierte Risikomanagementakte

5.3.12 Validierungsplan erstellen

In dieser Aktivität wird der Validierungsplan erstellt.

5.3.12.1 Beschreibung

Der Validierungsplan wird zweckmäßigerweise über die gesamte Entwicklungszeit hinweg erstellt. Durch die Validierung soll überprüft werden, inwieweit das Medizingerät Benutzeranforderungen erfüllt, für den beabsichtigten Gebrauch geeignet ist, und inwieweit die verbleibenden Restrisiken den vorgegebenen Abnahmekriterien genügen. Der Validierungsplan muss Verfahren zur Validierung der Benutzerschnittstelle beinhalten. Der Validierungsplan dokumentiert alle die für die Validierung erforderlichen Prüfungen. Weiterhin dokumentiert der Validierungsplan die Abhängigkeiten zwischen der Validierung und Entwicklung. Häufig werden die Prüfungen zur Validierung manuell ausgeführt. Es sollte dennoch versucht werden, auch diese Prüfungen so weit wie möglich zu automatisieren. Der Validierungsplan wird nach einer Änderung einem Review unterzogen.

5.3.12.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher

- Chefentwickler
- Tester (bei Bedarf)

5.3.12.3 Voraussetzungen

- Keine

5.3.12.4 Ergebnisse

- Aktualisierter Validierungsplan

5.4 Übergabephase

In der Übergabephase wird das neu entwickelte Medizinprodukt erstmalig produktiv genutzt.

5.4.1 Validierung durchführen

In dieser Aktivität werden die im Validierungsplan spezifizierten Prüfungen durchgeführt.

5.4.1.1 Beschreibung

Durch die Validierung soll überprüft werden, inwieweit das Medizingerät Benutzeranforderungen erfüllt, für den beabsichtigten Gebrauch geeignet ist und inwieweit die verbleibenden Restrisiken den vorgegebenen Abnahmekriterien genügen. Die Validierung erfolgt häufig in einer produktiven Umgebung.

5.4.1.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher

5.4.1.3 Voraussetzungen

- Funktionalitäten des Product Backlog komplett abgearbeitet
- Systemtest vollständig

5.4.1.4 Ergebnisse

- Validierungsprotokoll

5.4.2 System freigeben

In dieser Aktivität wird das System für den produktiven Betrieb freigegeben.

5.4.2.1 Beschreibung

Bevor die Software eingesetzt werden kann, muss sie freigegeben werden. Bevor die Software freigegeben werden darf, muss sichergestellt werden, dass die Verifizierung der Software vollständig ist, nicht gelöste Anomalien bewertet und dokumentiert wurden und die Dokumentation vollständig ist. Zudem muss sichergestellt sein, dass die Software-Freigabe bei Bedarf wiederholt werden kann. Die Freigabe erfolgt durch ein Review, indem die entsprechenden Dokumente geprüft werden.

5.4.2.2 Beteiligte

- Produktverantwortlicher
- Qualitätsverantwortlicher
- Chefentwickler

5.4.2.3 Voraussetzungen

- Funktionalitäten des Product Backlog komplett abgearbeitet
- Systemtest vollständig
- Validierung abgeschlossen

5.4.2.4 Ergebnisse

- Produktfreigabe

6 Umsetzung der regulatorischen Vorgaben

Nachdem das das agile Vorgehensmodell für die Entwicklung medizinischer Software umfänglich in den beiden vorherigen Kapitel vorgestellt wurde, bleibt zwei Fragen offen: Kann dieses Vorgehensmodell denn tatsächlich noch als agiles Vorgehensmodell bezeichnet werden und erfüllt das Vorgehensmodell denn tatsächlich die regulatorischen Vorgaben. Die zweite Frage wird in diesem Kapitel beantwortet.

In den regulatorischen Anforderungen werden umfangreiche Anforderungen an den Software-Entwicklungsprozess gestellt. Diese Anforderungen sind teils nach Disziplinen, wie Anforderungsanalyse, Design, Verifikation und Validierung aufgeteilt, teils werden eigenständige Teilprozesse gefordert, wie etwa für Konfigurationsmanagement oder Risikoanalyse. Die Darstellung in diesem Kapitel orientiert sich an dieser Strukturierung.[177]

6.1 Konzepterstellung

Bevor das Projekt gestartet oder die Produktentwicklung begonnen werden kann, sollen erste Kundenanforderungen aufgenommen[178], regulatorische und gesetzliche Anforderungen ermittelt [179], Anforderungen an Organisation festgelegt [180] und die so ermittelten Anforderungen anschließend bewertet werden.[181]. Das wesentliche Ergebnis ist ein Produktkonzept, in dem die Vision eines neuen oder verbesserten Produktes beschrieben ist und die prinzipielle Durchführbarkeit des Projektes dargestellt wird. Alle diese Anforderungen werden auch an die Aktivität „Produktkonzept erstellen“ gestellt.

6.2 Entwicklungsplanung

Nachdem Einigung über das Konzept des Gerätes erzielt wurde, muss die Entwicklung des Gerätes geplant werden. Dies bedeutet, dass der benutzte Softwareentwicklungsprozess definiert und dokumentiert wird. Dieser Entwicklungsprozess muss die verschiedenen Verifizierungsakti-

[177] Die Darstellung der regulatorischen Vorgaben erfolgt hier nur oberflächlich. Eine umfangreichere Darstellung kann in den betreffenden Normen oder in [Fischer] gefunden werden.

[178] [ISO 13485], Kapitel 7.2.1 a).

[179] [ISO 13485], Kapitel 7.2.1 c).

[180] [ISO 13485], Kapitel 7.2.1 d).

[181] [ISO 13485], Kapitel 7.2.2.

vitäten, einen Risikomanagementprozess, einen Problemlösungsprozess und einen Konfigurationsmanagementprozess festlegen. Zudem müssen Anforderungen an die Dokumentation und die verwendeten Normen und Standards festgelegt werden. Weiterhin müssen ein Verifizierungs- und Validierungsplan erstellt werden und die Anforderungen an die Qualifikation des Personals festgelegt werden. [182],[183],[184]

Diese Anforderungen werden in der Aktivität „Entwicklung planen“ umgesetzt.

6.3 Anforderungsanalyse

Im Anforderungsmanagement wird generell zwischen Anwenderanforderungen und Systemanforderungen unterschieden. Nutzer- oder Anwenderanforderungen beschreiben, welche Anforderungen der Kunde an das zu entwickelnde System stellt, wohingegen Systemanforderungen spezifizieren, was das System leisten soll.[185]

Die regulatorischen Anforderungen erwähnen sowohl Benutzeranforderungen als auch Systemanforderungen. So wird gefordert, dass die Anforderungen an das Produkt ermittelt und dokumentiert werden.[186] Die Anforderungen müssen von Anforderungen an das Gesamtsystem abgeleitet sein[187], Informationen zu den risikobezogenen Funktionen und ggf. Risikobeherrschungsmaßnahmen enthalten.[188] Weiterhin werden die Festlegung einer Sicherheitsklassifizierung und eine Verifizierung der Anforderungen verlangt.[189] Zudem müssen die Hauptbedienfunktionen, sowie die Gebrauchstauglichkeit und [190] der bestimmungsgemäße Gebrauch festgelegt werden.[191]

Das Vorgehensmodell unterscheidet ebenfalls zwischen Nutzeranforderungen und Systemanforderungen. Nutzeranforderungen und weitere in dem Bereich des Anwenders und Patienten

---

[182] [IEC 60601-1-4], Kapitel 52.203.1, 52.203.2, 52.203.3, 52.203.4, 52.203.6, 52.204.2, 52.204.2, 52.209.2, 52.210.2, 52.210.4.

[183] [IEC prEN 62304], Kapitel 4.1, 5, 5.1, 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 5.1.9, 5.1.10 und 5.1.12.

[184] [ISO 14971] Kapitel 3.4.

[185] Zu verschiedenen Definitionen von Anforderungen, s. [Versteegen1], S. 3f.

[186] [ISO 13485], Kapitel 7.1, 7.2.1 und 7.3.2

[187] Sofern es sich nicht um eines reines Softwaresystem handelt.

[188] [IEC 60601-1-4], Kapitel 52.206

[189] [IEC prEN 62304], Kapitel 4.3, 5.1.3 und 5.2.

[190] [IEC 60601-1-6], Kapitel 46.202.1 und 46.202.2.

[191] [ISO 14971], Kapitel 4.2.

liegenden Anforderungen werden im Product Backlog festgehalten, wohingegen Systemanforderungen, also Anforderungen an das System, die von den Nutzeranforderungen abgeleitet wurden, in der Konzeptionellen Architektur dokumentiert werden.

Diese Anforderungen werden in der Aktivität „Anforderungen ermitteln“ und in der Aktivität „Konzeptionelle Architektur erstellen“ umgesetzt.

6.4 Systementwurf

Basierend auf den Anforderungen an das zu entwickelnde Produkt kann nun das System selbst entwickelt werden. Das konkrete Verfahren, durch das das Softwareprodukt spezifiziert wird, wird durch die regulatorischen Anforderungen offen gelassen. Das Ergebnis des Systementwurfs wird hier - unabhängig von dem Unfang und der Art der eingesetzten Verfahren - als Designspezifikation bezeichnet. Wie umfangreich die Designspezifikation sein muss, machen die regulatorischen Vorgaben von dem Risiko abhängig, das das betreffende System für Benutzer, Patienten und Dritte darstellt.

Die regulatorischen Vorgaben beschreiben lediglich Prinzipien, die bei dem Softwareentwurf berücksichtigt werden sollen. Es wird gefordert, das System hierarchisch in Subsysteme und Komponenten zu zergliedern, wobei der aus regulatorischen Gründen erforderliche Umfang der Zergliederung von der Gefährdung abhängig ist, die von dem System oder den Komponenten ausgehen kann. Für jedes System sollen die Softwareanforderungen in eine Architektur umgewandelt werden, die den Systemanforderungen genügt und die Anforderungen zur Risikobeherrschung berücksichtigt.[192] Diese Architektur muss dokumentiert und verifiziert werden. Dies gilt auch für die Schnittstellen zwischen den verschiedenen Subsystemen und SOUP[193]-Komponenten[194]. Das Ergebnis des Architekturentwurfs ist also eine Architekturspezifikation, die die oben beschriebenen Informationen in geeigneter Form enthält.

In Abhängigkeit von der Sicherheitsklasse, die der Komponente bzw. dem System zugeordnet wurde, muss die hierarchische Zerlegung weitergeführt werden. [IEC prEN 62304] [195] fordert für Komponenten der Sicherheitsklasse B und C[196] ein detailliertes Design.

[192] [IEC 60601-1-4], Kapitel 52.207.

[193] SOUP: Software unbekannter Herkunft. Software-Komponenten, die nicht entwickelt wurden, um in das in Entwicklung befindliche Medizingerät eingefügt zu werden, und für die der Entwicklungsprozess unbekannt ist. s. [IEC prEN 62304], S. 10.

[194] [IEC prEN 62304], Kapitel 5.3.

[195] [IEC prEN 62304], Kapitel 5.4.2.

Für solche Systeme bzw. Systemteile muss der Hersteller die Software-Architektur soweit verfeinern, bis sie durch Software-Einheiten dargestellt wird[197]. Für jede Software-Einheit der Sicherheitsklasse C muss ein detailliertes Design entwickelt werden[198]. Für alle Schnittstellen zwischen einer Software-Einheit der Sicherheitsklasse C und externen (Hardware- oder Software-) Schnittstellen, sowie zu anderen Software-Einheiten muss ein detailliertes Design entwickelt werden[199]. Das Ergebnis des detaillierten Designs ist eine Designspezifikation, die die oben beschriebenen Informationen in geeigneter Form enthält.

Diese Anforderungen werden in der Aktivität „Anforderungen analysieren“ und in der Aktivität „Lösung entwerfen, implementieren und testen“ umgesetzt.

6.5 Implementierung und Modultest

Letztendlich muss die einzelne Software-Einheit implementiert werden. Nach der Implementierung muss die erstellte Implementierung hinsichtlich der zuvor festgelegten Akzeptanzkriterien verifiziert werden. Verifizierungsmaßnahmen für die einzelnen Module werden lediglich für Softwareeinheiten der Sicherheitsklasse B und C gefordert. Es ist allerdings nicht verkehrt, Verifizierungsmaßnahmen in jedem Fall durchzuführen.

In verschiedenen Dokumenten werden zwar Beispiele für angemessene Verifizierungsmaßnahmen beschrieben, es werden aber keine bestimmten Verifizierungsmaßnahmen zwingend vorgeschrieben. Es muss jedoch ein Verifizierungsprozess definiert sein, in dem die Akzeptanzkriterien für die Verifizierung der Softwareeinheiten festgelegt werden. [200] Bei der Verifizierung der Softwareeinheiten müssen sodann diese Akzeptanzkriterien überprüft werden. Weiterhin muss geprüft werden, ob Programmierverfahren und Richtlinien eingehalten wurden, es keine Widersprüche mit den Schnittstellen des detaillierten Designs gibt und Anforderungen und Risikokontrollmaßnahmen korrekt implementiert wurden.[201]

Diese Anforderungen werden in der Aktivität „Lösung entwerfen, implementieren und testen“ umgesetzt.

---

[196] Die Festlegung der Gefährdungsklasse in [IEC prEN 62304] und die Festlegung des „Levels of Concern“ der FDA unterscheidet sich bestenfalls in Details. Für die Betrachtung in dieser Arbeit ist dies jedoch irrelevant und daher werden Minor mit A, Moderate mit B und Major mit C identifiziert.

[197] [IEC prEN 62304], Kapitel 5.4.1.

[198] [IEC prEN 62304], Kapitel 5.4.2.

[199] [IEC prEN 62304], Kapitel 5.4.3.

[200] [IEC prEN 62304], Kapitel 5.5.

[201] [IEC prEN 62304], Kapitel 5.5.2, 5.5.3, 5.5.4 und 5.5.5.

6.6 Verifizierung

Verifizierung nennt man den Vorgang, durch den der Nachweis erlangt wird, dass das System oder Komponenten vorgegeben Kriterien genügen. Um zu verifizieren, dass das Software-System die vorgegeben Anforderungen erfüllt, muss es unter kontrollierten Bedingungen mit vorgegebenen Eingaben ausgeführt werden. Die dabei beobachteten Ausgaben werden dabei mit erwarteten Ausgaben verglichen und anschließend, einschließlich eventuell auftretender Abweichungen, dokumentiert[202]. In dem Verifizierungsplan wird festgelegt, an welchen Stellen im Entwicklungsprozess solche Verifizierungsaktivitäten vorgesehen sind. Weiter ist im Verifizierungsplan zu notieren, welche Kriterien an die auszuführenden Tests zu stellen sind, üblicherweise basierend auf der Sicherheitsklassifizierung der Komponenten oder des Systems. [203]. Üblicherweise werden Modultest, Integrationstest, Systemtest und Abnahmetest unterschieden. Um Dokumente zu verifizieren wird häufig ein Review durchgeführt.

Im Integrationstest wird geprüft, ob die, bereits einzeln geprüften Komponenten, spezifikationsgemäß zusammenarbeiten.[204]. Die dazu erforderlichen Aktivitäten sind im Verifikationsplan festzuhalten. Im Systemtest wird hingegen geprüft, ob das System als Ganzes die Anforderungen an das System erfüllt[205]. Dabei müssen Prüfungen für jede Softwareanforderung festgelegt werden und der Problemlösungsprozess für gefundene Anomalien benutzt werden.[206] Die durchgeführten Prüfungen müssen aufgezeichnet werden.[207] Nach Änderungen müssen die Prüfungen wiederholt werden.[208] In vielen Fällen wird der Integrationstest zusammen mit dem Systemtest ausgeführt.

Diese Anforderungen werden in der Aktivität „Systemtest erstellen“ und „Systemtest durchführen“ umgesetzt.

6.7 Validierung

Durch die Validierung soll überprüft werden, inwieweit das Medizingerät Benutzeranforderungen erfüllt, für den beabsichtigten Gebrauch geeignet ist und inwieweit die verbleibenden Restrisiken

[202] [ISO 13485], Kapitel 7.3.5, 7.6, 8.1 (a) und 8.2.4.1.

[203] [IEC 60601-1-4], Kapitel 52.209 und 52.212 und [IEC prEN 62304], Kapitel 5.1.7, 5.2.7, 5.3.7, 5.4.4., 5.4.5, 7.3.1 und 9.9.

[204] [IEC prEN 62304], Kapitel 5.6

[205] [ISO 13485], Kapitel 8.2.4.1.

[206] [IEC prEN 62304], Kapitel 5.7.1 und 5.7.2.

[207] [IEC prEN 62304], Kapitel 5.7.5.

[208] [IEC prEN 62304], Kapitel 5.7.3.

den vorgegebenen Abnahmekriterien genügen[209]. [21CFR820] definiert Validierung als „Bestätigung durch Untersuchung und Bereitstellung von objektivem Nachweis, dass die besonderen Anforderungen für einen bestimmten beabsichtigten Gebrauch gleich bleibend erfüllt werden“[210]. Um diesen Nachweis zu erbringen ist es erforderlich, einen Validierungsplan zu erstellen[211], und die Validierung nach diesem Plan durchzuführen.[212] Der Validierungsplan muss die Abhängigkeiten zwischen der Validierung und Entwicklung beschreiben[213] und Verfahren zur Validierung der Benutzerschnittstelle müssen eingeschlossen werden.[214] Der Validierungsplan muss im Entwicklungsplan referenziert werden.[215]

Diese Anforderungen werden in der Aktivität „Validierungsplan erstellen“ und „Validierung durchführen“ umgesetzt.

6.8 Freigabe

Bevor die Software eingesetzt werden kann, muss sie freigegeben werden. Bevor die Software freigegeben werden darf, muss sichergestellt werden, dass die Verifizierung der Software vollständig ist[216], nicht gelöste Anomalien bewertet und dokumentiert wurden[217] und die Dokumentation vollständig ist[218]. Zudem muss sichergestellt sein, dass die Software-Freigabe bei Bedarf wiederholt werden kann.[219]

Der Hersteller muss geänderte Software-Systeme erneut freigeben. Änderungen können als Teil einer vollständigen erneuten Freigabe eines Software-Systems freigegeben werden, oder als Änderungs-Bausatz, der die geänderten Software-Komponenten und die Werkzeuge enthält, die erforderlich sind, um die Änderungen als Änderungen in ein bestehendes Software-System zu installieren.[220]

---

[209] [ISO 13485], Kapitel 7.3.6.

[210] [21CFR820] 3(z), Validation means confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use can be consistently fulfilled.

[211] [IEC 60601-1-4], Kapitel 52.210.2.

[212] [IEC 60601-1-4], Kapitel 52.210.3.

[213] [IEC 60601-1-4], Kapitel 52.210.5.

[214] [IEC 60601-1-6],Kapitel 46.202.5.

[215] [IEC prEN 62304], Kapitel 5.1.3 (b).

[216] [IEC prEN 62304], Kapitel 5.8.1.

[217] [IEC prEN 62304], Kapitel 5.8.2 und 5.8.3.

[218] [IEC prEN 62304], Kapitel 5.8.4, 5.8.5 und 5.8.6.

[219] [IEC prEN 62304], Kapitel 5.8.8.

[220] [IEC prEN 62304], Kapitel 6.3.2.

Diese Anforderungen werden in der Aktivität „System freigeben“ umgesetzt.

6.9 Management von Änderungen

Jede Änderungsanforderung muss zunächst bewertet werden, um festzustellen, in wieweit die Änderung Einfluss auf die Sicherheit hat. Jede Änderungsanforderung, die auf ein Problem zurückgeht, muss als Problembericht aufgezeichnet werden.

Eine Änderung an einem freigegebenen Software-Produkt muss als eine oder mehrere Änderungsspezifikationen dokumentiert werden. Dabei wird unter einer Änderungsspezifikation die dokumentierte Spezifikation einer Änderung verstanden[221].

Bevor die Änderung umgesetzt wird, muss sie spezifiziert werden und die Änderungsspezifikation muss hinsichtlich ihrer Auswirkungen analysiert werden[222] und anschließend freigegeben werden. Alle für die Änderung relevanten Dokumente müssen überarbeitet[223], ergänzt, bewertet und erneut freigeben werden[224]. Das geänderte Softwaresystem muss sodann erneut verifiziert[225] und freigegeben werden.

Diese Anforderungen werden in der Aktivität „Anforderungen anpassen“ sowie den planerischen Aktivitäten „Release planen“ und „Iteration planen“ umgesetzt.

6.10 Management von Problemen

Der Entwicklungsprozess muss ein Verfahren für die Problemlösung enthalten[226]. Dieses Verfahren muss über alle Phasen und Aktivitäten des Entwicklungsprozesses hinweg verwendet werden, muss innerhalb der Entwicklungsplanung festgelegt und im Entwicklungsplan dokumentiert werden[227]. Weiterhin wird gefordert, dass alle Probleme und Nicht-Konformitäten, die während der Entwicklung und während des Betriebes entdeckt werden, in ein dokumentiertes Problemlösungsverfahren für Software einzubringen sind, nachdem diese entdeckt wurden[228]. Wich-

[221] [IEC prEN 62304], Kapitel 6.2.1.4.

[222] [IEC prEN 62304], Kapitel 6.2.3 und 7.4.

[223] [IEC 60601-1-4], Kapitel 52.211.

[224] [IEC prEN 62304], Kapitel 6.2.4 und 8.2.

[225] [IEC prEN 62304], Kapitel 5.7.3.

[226] [IEC prEN 62304], Kapitel 5.1.1 und [IEC 60601-1-4], Kapitel 52.203.6.

[227] [IEC 60601-1-4]: Kapitel 52.203.6 und [IEC prEN 62304]: Kapitel 5.1.12.

[228] [IEC prEN 62304]: Kapitel 5.6.8 und 5.7.2.

tig ist dabei, dass die Komponenten eindeutig identifiziert werden können. Dies ist aber erst möglich, nachdem sie unter Konfigurationskontrolle gestellt wurden[229].

Nachdem das Medizinprodukt freigegeben wurde, darf sich der Hersteller nicht darauf verlassen, dass aufgetretene Probleme an ihn weitergeleitet werden. Der Hersteller muss vielmehr aktiv nach Rückmeldungen über aufgetretene Probleme suchen, und zwar sowohl innerhalb seiner eigenen Organisation als auch bei Anwendern.[230]

Für jedes Problem, das identifiziert wurde, muss ein Problembericht erstellt werden[231]. Problemberichte müssen nach Typ, Umfang und Kritikalität klassifiziert werden. Dies gilt auch für Anomalien, die während Verifikationsmaßnahmen wie Integrationstests gefunden wurden[232]. Problemberichte müssen wirkliche oder mögliche Schadensereignisse und wirkliche oder vermeintliche Abweichungen von Spezifikationen enthalten. Sie müssen bewertet werden, um festzustellen, ob ein wirkliches Problem besteht, in wieweit das Problem die Sicherheit betrifft, und ob eine Änderung des Software-Produktes erforderlich ist, um das Problem zu adressieren. Zur Identifizierung möglicher Sicherheitsrisiken ist der Risikomanagementprozess zu verwenden. [233].

Das Problem muss untersucht werden und, soweit möglich, die Ursachen des Problems identifiziert werden. Danach muss entschieden werden, ob das Problem gelöst werden soll. Für die Ausführung der notwendigen Änderungen ist das definierte Änderungsverfahren zu verwenden. Das Problemlösungsverfahren wird auch dann eingehalten, wenn – unter Berücksichtigung möglicher Sicherheitsbeeinträchtigungen – entschieden wird, dass das Problem nicht korrigiert wird.[234]

Der Hersteller muss den Zustand des Problemberichts, sowie ggf. die zugehörigen Änderungsspezifikationen verfolgen und dokumentieren.

Diese Anforderungen werden in der Aktivität „Probleme behandeln“, „Anforderungen anpassen“ sowie den planerischen Aktivitäten „Release planen“ und „Iteration planen“ umgesetzt.

---

[229] Daher wird auch gefordert, dass Komponenten und Systeme vor der Durchführung von Verifikationstätigkeiten unter Konfigurationskontrolle gestellt werden.

[230] [ISO 13485]: Kapitel 7.2.3 (b) und [IEC prEN 62304], Kapitel 6.2.1.1.

[231] [ISO 13485]: Kapitel 8.3 und [IEC prEN 62304]: Kapitel 6.2.1.2.

[232] [IEC prEN 62304], Kapitel 5.6.8.

[233] [IEC prEN 62304], Kapitel 6.2.1.3 und 9.4.

[234] [IEC prEN 62304], Kapitel 9.

6.11 Konfigurationsmanagement

Der Hersteller muss während der Entwicklungsplanung das Konfigurationsmanagement planen und die getroffenen Festlegungen im Softwareentwicklungsplan dokumentieren oder referenzieren.[235] Häufig wird das Konfigurationsmanagement in einem separaten KM-Plan definiert. Die folgende Liste listet alle Dokumente mit den Kapiteln auf, die wesentliche Vorgaben zum Konfigurationsmanagement enthalten.

Innerhalb des Konfigurationsmanagements muss festgehalten werden, welche Arten von Komponenten unter Konfigurationskontrolle gestellt werden sollen, beispielsweise Bibliotheken mit ausführbaren Codes (Dll), ausführbare Programme, Konfigurationsdateien usw.[236] Die Komponenten, die unter Konfigurationskontrolle stehen, können in einem separaten Konfigurationsidentifikationsdokument (KID) dokumentiert werden. Konfigurationselemente müssen vor der Verifizierung unter Konfigurationskontrolle gestellt werden. Sie dürfen nur als Reaktion auf eine Änderungsspezifikation geändert werden.[237]

Diese Anforderungen werden in der den planerischen Aktivitäten „Release planen“ und „Iteration planen“ sowie der Aktivität „Lösung entwerfen, implementieren und testen“ und der Aktivität „Release freigeben“ umgesetzt.

6.12 Wartungsprozess

Die Wartung des Softwaresystems erfolgt nach der Freigabe, während das System in Betrieb ist. Damit gehört dieser Prozess nicht mehr eigentlich in den Bereich der Softwareentwicklung des Systems. Dennoch ist es wichtig, dass die für die Wartung erforderlichen Aktivitäten geplant und umgesetzt werden.

Für die Wartung wird ein Verfahren für die Verarbeitung von Problemberichten und Änderungsanforderungen gefordert.[238] Dieses Verfahren muss Kriterien festlegen, wann ein Problembericht oder eine Änderungsanforderung als Problem aufgefasst werden muss.[239] Für die Adressierung von Gefährdungen ist der Risikomanagementprozess zu benutzen. Für die Handhabung von

[235] [ISO 13485]: Kapitel 7.5.3.1 und [IEC prEN 62304]: Kapitel 5.1.10.

[236] [IEC prEN 62304], Kapitel 8.1.

[237] [IEC prEN 62304], Kapitel 8.2.

[238] [IEC prEN 62304], Kapitel 6.

[239] [IEC prEN 62304], Kapitel 6.2.1.

Problemen ist das Problemlösungsverfahren zu verwenden.[240] Für die Handhabung von Änderungsanforderungen ist das Änderungsverfahren zu benutzen.

Für die über die Softwareentwicklung hinausgehenden Tätigkeiten, wie die aktive Beobachtung des Marktes wurden keine Aktivitäten vorgesehen, da diese Tätigkeiten nicht mehr in den Bereich der Softwareentwicklung fallen. Sofern Probleme identifiziert wurden, ist die Aktivität „Probleme behandeln“ zu verwenden.

6.13 Risikomanagement

Durch die regulatorischen Vorgaben wird ein Prozess zum Management von Risiken gefordert. Dieser Prozess muss dokumentiert werden und Elemente zur Risikoanalyse, Risikobewertung und Risikokontrolle beinhalten. [241].

Damit der Risikomanagementprozess angemessen durchgeführt werden kann, ist es erforderlich, dass die Organisation Ihre Grundsätze zur Festlegung vertretbarer Risiken unter Berücksichtigung einschlägiger Normen und Vorschriften bestimmt, die Verfügbarkeit der benötigten Mittel sicherstellt, das erforderliche, ausgebildete Personal bereitstellt und die Ergebnisse der Risikomanagementaktivitäten in regelmäßigen Abständen überprüft, um die Wirksamkeit des Risikomanagementprozesses sicherzustellen.[242]

Diese Festlegungen und die spezifische Umsetzung der drei Verfahren zur Risikoanalyse, Risikobewertung und Risikokontrolle werden üblicherweise während der Entwicklungsplanung festgelegt und in der Risikomanagementakte dokumentiert.[243]

Um mögliche Gefährdungen analysieren zu können, ist es zunächst erforderlich, den bestimmungsgemäßen Gebrauch und den vorhersehbaren Missbrauch zu ermitteln und zu dokumentieren. Basierend auf diesen Informationen wird eine Liste aller quantitativen und qualitativen Merkmale erarbeitet, die die Sicherheit des Medizinproduktes beeinflussen. Diese Daten sind in der Risikomanagementakte zu dokumentieren. Weiterhin müssen in der Risikoanalyse alle vernünftigerweise vorhersehbaren Umstände der Gefährdungen für Patienten, Anwender, Servicepersonal, Unbeteiligte sowie Umgebung und Umwelt ermittelt werden. Als Ursachen müssen sowohl menschliche Eigenschaften, Fehler sowie Umgebungsbedingungen als auslösende Ursachen betrachtet werden. Zudem müssen die vernünftigerweise vorhersehbaren Folgen von

[240] [IEC prEN 62304], Kapitel 6.2.2.

[241] [ISO 14971], Kapitel 3, [IEC 60601-1-4], Kapitel 52.203.3 und [IEC prEN 62304], Kapitel 7.

[242] [IEC 60601-1-4]: Kapitel 52.202 und [ISO 14971], Kapitel 3.

[243] [IEC prEN 62304], Kapitel 5.1.8.

Ereignissen, die in einer Gefährdung resultieren, betrachtet werden. Die in der Risikoanalyse eingesetzten Verfahren werden in der Norm nicht festgelegt, jedoch müssen diese Verfahren in der Risikomanagementdokumentation enthalten sein oder referenziert werden.[244]

Durch die Risikobewertung werden die während der Risikoanalyse ermittelten möglichen Gefährdungen bewertet. Dazu werden für jede mögliche Gefährdung die Auftretenswahrscheinlichkeit und das Schadensausmaß ermittelt. Für jede Gefährdung muss sodann – unter Anwendung der im Risikomanagementplan festgelegten Kriterien – entschieden werden, ob das eingeschätzte Risiko so gering ist, dass eine Risikominderung nicht erforderlich ist, oder ob Maßnahmen zur Risikominderung festgelegt werden müssen. Die Risikobewertung ist in der Risikomanagementakte zu dokumentieren.[245]

Wenn eine Risikominderung erforderlich ist, müssen Maßnahmen festgelegt werden, um das Risiko auf ein vertretbares Maß zu reduzieren. Eine Maßnahme kann dabei die Wahrscheinlichkeit des Auftretens oder den Schweregrad des möglichen Schadens verringern. Dazu muss zunächst versucht werden, das Risiko durch eine Änderung des Designs zu beseitigen oder zu verringern. Ist dies nicht möglich, sind Schutzvorrichtungen oder Schutzmaßnahmen vorzusehen. Ist auch dies nicht möglich, sind Sicherheitsinformationen vorzusehen. Die ausgewählten Maßnahmen zur Risikokontrolle sind in der Risikomanagementakte zu dokumentieren. Jedes Restrisiko, das nach der Durchführung der Maßnahmen zur Risikokontrolle verbleibt, muss anhand der im Risikomanagementplan festgelegten Kriterien bewertet werden. Die Bewertung ist in der Risikomanagementakte zu dokumentieren.[246]

Der Risikomanagementprozess ist, wie es verlangt wird, ein integraler Bestandteil des Entwicklungsprozesses. Daher kann auch keine einzelne Aktivität angegeben werden, durch die die Vorgaben erfüllt werden. In den planerischen Aktivitäten „Release planen“ und „Iteration planen“ findet immer eine Analyse und Bewertung möglicher Risiken statt. Weiterhin wird geprüft, ob vorgesehene Maßnahmen zur Risikoverringerung nicht vergessen werden. Bei den Fertigstellungsaktivitäten „Iteration Review Meeting“, „Release freigeben“ und „System freigeben“ wird nochmals explizit die Umsetzung dieser Maßnahmen geprüft. Nicht zuletzt wird in der „System freigeben“ das verbleibende Restrisiko bewertet.

[244] [ISO 14971], Kapitel 4, [IEC 60601-1-4], Kapitel 52.204.3 und [IEC prEN 62304], Kapitel 7.1.

[245] [ISO 14971], Kapitel 5, [IEC 60601-1-4], Kapitel 52.204.3.

[246] [ISO 14971], Kapitel 6, [IEC 60601-1-4], Kapitel 52.204.4 und [IEC prEN 62304], Kapitel 7.2.

7 Fazit

Softwareentwicklung mit agilen Praktiken hat in letzten Jahren immer mehr Verbreitung gefunden. Die Ausführungen in den vorherigen Kapiteln haben gezeigt, dass es möglich ist, agile Vorgehensmodelle so zu erweitern, dass sie auch für die Entwicklung medizinischer Software geeignet ist. Zwar ist das so entstandene Vorgehensmodell deutlich schwergewichtiger und verlangt mehr Dokumentation als dies in agilen Vorgehensmodellen, wie XP oder Scrum üblich ist. Kann man dennoch das beschriebene Vorgehensmodell noch agil nennen? Schauen wir uns die vier agilen Werte an:

Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge“

Dies gilt uneingeschränkt für das beschriebene Vorgehensmodell. Prozesse und Dokumentation sind zwar wichtig, aber nur in dem Umfang, wie die regulatorischen Vorgaben dies verlangen. Die Zusammenarbeit im Team nimmt auch in dem oben beschrieben Vorgehensmodell eine zentrale Rolle ein. Umfangreiche Planung findet immer nur für die nächste Iteration statt, und in diesem Punkt unterschiedet sich das beschriebene Vorgehensmodell nicht wesentlich von XP oder Scrum.

„Funktionierende Software ist wichtiger als umfassende Dokumentation“

Der Dokumentationsanteil ist in dem beschriebenen Vorgehensmodell deutlich größer, als dies von XP oder Scrum her bekannt ist. Es wird aber kein unnütze Dokumentation erzeugt. Zudem liegt auch der Fokus dieses Vorgehensmodells auf der Erstellung lauffähiger Software.

„Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen“

Die Rolle des Produktverantwortlichen unterscheidet sich nicht wesentlich von der des Product Owners in Scrum. Die Übernahme von Änderungen vollzieht sich etwas schwergewichtiger, da immerhin die Sicherheit des Patienten und des Benutzers Berücksichtigung finden muss. Anmerkungen und Einwände können aber dennoch spätestens in dem nächsten Release berücksichtigt werden. Rückmeldungen von Benutzern können kurzfristig berücksichtigt werden.

„Sich auf unbekannte Änderungen einzustellen ist wichtiger als einem zu folgen“

Die Entwicklung erfolgt iterativ-inkrementell. Daher ist der Entwicklungsprozess darauf ausgelegt, Änderungen angemessen zu berücksichtigen. Auch Anpassungen an dem Prozess selbst wird durch die Iteration Retrospektive berücksichtigt.

Die vier Werte des agilen Manifestes finden sich also in dem oben beschriebenen Vorgehensmodell Daher kann man mit gutem Grund das Vorgehensmodell agil nennen. Das vorgestellte Vorgehensmodell hat sicherlich den Nachteil, nicht nur die Praktiken eines Vorgehensmodells zu

verwenden, sondern Praktiken mehrerer agiler Vorgehensmodelle kombiniert wurden. Das wurde aber nicht verlangt.

Das beschriebene Vorgehensmodell organisiert sich an kleinen Teams und damit auch an kleinen bis mittleren Projekten. Gerade Scum positioniert sich immer mehr auch als agiles Vorgehensmodell für große Projekte. Wie muss das beschrieben Vorgehensmodell ergänzt werden, um auch für große Projekte geeignet zu sein? Wie können mehrere Teams in einer agilen Umgebung koordiniert zusammenarbeiten? Hier kann sicherlich in der Zukunft noch einiges getan werden.

Eine weitere Frage stellt sich hinsichtlich der Beschreibung der Vorgehensmodelle. Kann hier vielleicht ein Metamodell für die Beschreibung von agilen Vorgehensmodellen, oder agilen Vorgehensmodellen für die Entwicklung medizinischer Software gefunden werden?

8 Referenzen

[Abrahamsson]	Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta : Agile software development methods, Review and analysis, http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf, letzter Zugriff 26.04.2008
[Balzert1996]	Helmut Balzert, Lehrbuch der Software-Technik, Software-Entwicklung, Spektrum Akademischer Verlag, 1996
[Balzert1998]	Helmut Balzert, Lehrbuch der Software-Technik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung Spektrum Akademischer Verlag, 1998
[Balzert2008]	Helmut Balzert, Lehrbuch der Software-Technik, Softwaremanagement, 2. Auflage, Spektrum Akademischer Verlag, 2008
[Beck2000]	Kent Beck - extreme Programming explained - Embrace Change, Addison Wesley, 2000
[Beck2004]	Kent Beck - extreme Programming explained - Embrace Change, Addison Wesley, Second Edition, 2004
[Bleek2008]	Wolf-Gideon Bleek, Henning Wolf: Agile Softwareentwicklung – Werte, Konzepte und Methoden, dpunkt.verlag, Heidelberg, 2008
[Boehm2002]	Barry Boehm - Get Ready for Agile Methods, http://www2.umassd.edu/SWPI/xp/papers/r1064.pdf, letzter Zugriff 25.04.2008
[Boehm-Turner]	Barry Boehm, Richard Turner - Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley, 2003
[Bosch]	Tim Bosch, Increase the Efficiency and Pace of Medical Software Development by Going Agile, www. Foliage.com, letzter Zugriff 05.01.2008
[Broy2001]	Manfred Broy, Software Engineering – From Auxiliary to Key Technology, in: Manfred Broy, Ernst denert (Eds.): Software Pioneers – Contributions to Software Engineering, Springer Verlag, 2001
[CHAOS1994]	Chaos-Report der Standish Group, zitiert nach: Ingolf Giese, Warum explodierten Mariner 1, Ariane 5 ..., GSI Darmstadt, http://www-aix.gsi.de/~giese/swr/folgen1.html, letzter Zugriff, 25.04.2008

[CHAOS2006]	Chaos-Report der Standish Group, zitiert nach: Jedes fünfte Projekt ist ein Totalausfall, Computerwoche vom 12.03.2007, http://www computer-woche.de/produkte_technik/589879/, letzter Zugriff 25.04.2008
[cLab]	cLab Papers, http://clabs.org/xpprac.htm, letzter Zugriff 25.04.2008
[deLuca-Pod]	Software Engineering Radio Episode 83: Jeff DeLuca on Feature Driven Development, http://www.se-radio.net/podcast/2008-01/episode-83-jeff-deluca-feature-driven-development, letzter Zugriff, 22.04.2008
[Dijkstra1972]	Edsger W. Dijkstra, The Humble Programmer, http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.pdf, letzter Zugriff 08.04.2008, zuerst veröffentlich in Commun. ACM 15, (1972), 10:859-866
[Dogs]	Carsten Dogs, Timo Klimmer – Agile Softwareentwicklung kompakt, mitp-Verlag, 2005
[Dogs-Timmer]	Carsten Dogs, Timo Klimmer, An Evaluation of the Usage of Agile Core Practices, Master Thesis Software Engineering Thesis no: MSE-2004:07 June 2004, http://www.bth.se/fou/cuppsats.nsf/all/807abbdae15662afc1256eb500403a8f /$file/Master%20Thesis.pdf, letzter Zugriff 07.05.2008
[Ebert]	Christof Ebert - Systematisches Requirements Management, dpunkt Verlag, 2005
[Fischer]	Bernhard Fischer, Gestaltung regulatorischer Vorgaben bei der Entwicklung medizinischer Software, Projektarbeit, http://www.johner-institut.de/fileadmin/institute/ProjektMasterArbeiten/Projektarbeit-Bernhard-Fischer.pdf, letzter Zugriff 02.05.2008.
[Hamilton]	Patrick Hamilton: Dynaxity – Management von Dynamik und Komplexität im Softwarebau, Springer, 2007
[Highsmith]	Jim Highsmith - Agile Software Development Ecosystems, Addison-Wesley, 2002
[Hüttermann]	Michael Hüttermann, Agile Java-Entwicklung in der Praxis, O'Reilly, 2008
[IEC 60601-1-4]	DIN EN IEC 60601-1-4:1996+A1:1999, erhältlich über den Beuth Verlag
[IEC 60601-1-6]	DIN EN 60601-1-6 (VDE 0750-1-6) Medizinische elektrische Geräte – Teil 1-6: Allgemeine Festlegung für die Sicherheit – Ergänzungsnorm: Gebrauchstauglichkeit (IEC 60601-1-6:2004), erhältlich über den Beuth

	Verlag
[IEC prEN 62304]	Medizingeräte-Software – Software-Lebenszyklus-Prozesse (IEC 62A / 474 /CDV:2004) Deutsche Fassung EN IEC prEN 62304:2004, Entwurf April 2005, erhältlich über den Beuth Verlag
[ISO 13495]	DIN EN ISO 13485:2003, erhältlich über den Beuth Verlag
[ISO 14971]	DIN EN ISO 14971 (ISO 14971:2000) Anwendung des Risikomanagement auf Medizinprodukte, Deutsche Fassung EN ISO 14971:2001, erhältlich über den Beuth Verlag
[Jeffries]	Ron Jeffries, Ann Anderson, Chet Hendrickson, Extreme Programming installed, Addison Wesley, 2001
[Jeffries2001]	What is Extreme Programming? written by Ron Jeffries, 11/08/2001, http://www.xprogramming.com/xpmag/whatisxp.htm, letzter Zugriff 22.04.2008
[Khramtchenko]	Serguei Khramtchenko, Comparing eXtreme Programming and Feature Driven Development in academic and regulated environments, http://www.featuredrivendevelopment.com/files/FDD_vs_XP.pdf, letzter Zugriff 01.05.2008
[Koskela]	Juha Koskela - Software configuration management in agile methods, http://www.vtt.fi/inf/pdf/publications/2003/P514.pdf, letzter Zugriff 24.04.2008
[Larman2004]	Craig Larman: Agile & Iterative Development – A Manager's Guide, Addison Wesley, 2004
[NATO1968]	Software Engineering, Report on a conference sponsored by the NATO Science committee, Garmisch, Germany, 7th to 11th October 1968, http://homepages.cs.ncl.uk/brian.randell/NATO/nato1968.PDF, letzter Zugriff 08.04.2008
[NATO1969]	Software Engineering Techniques, Report on a conference sponsored by the NATO Science committee, Rome, Italy, 27th to 31th October 1969, http://homepages.cs.ncl.uk/brian.randell/NATO/nato1969.PDF, letzter Zugriff 08.04.2008
[Oestereich2007]	B. Oestereich, C. Schröder, M. Klink, G. Zockoll, OEP . OOSE Engineering Process, dpunkt.verlag, Heidelberg, 2007
[Oestereich2008]	Bernd Oestereich, Christian Weiss, APM - Agiles Projektmanagement,

	dpunkt.verlag, 2008
[Palmer]	Stephen Palmer, John Felsing, A Practical Guide to Fetature-Driven Development, Prentice Hall, 2002
[Partsch]	Helmuth Partsch - Requirements-Engineering systematisch, Springer Verlag, 1998
[Pichler]	Roman Pichler, Scrum – Agiles Projektmanagement erfolgreich einsetzen, dpunkt.verlag, Heidelberg, 2008
[Pichler-Pod]	Software Engineering Radio Episode 60: Roman Pichler on Scrum, http://www.se-radio.net/podcast/2007-06/episode-60-roman-pichler-scrum, letzter Zugriff, 22.04.2008
[Pomberger],	G. Pomberger, W. Pree, Software-Engineering, Carl Hanser Verlag, München, Wien 2004
[Poppendieck]	Mary & Tom Poppendieck: Lean Software Development – An Agile Toolkit, Addison Wesley, 2003.
[Roock-Wolf-2]	Stefan Roock, Henning Wolf - Agile Feature Driven Development, Teil 2: Featurelisten, in: Java magazin, Ausgabe 02/08, Seite 126ff.
[Roock-Wolf-3]	Stefan Roock, Henning Wolf - Agile Feature Driven Development, Teil 3: Entwurf und Konstruktion der Features, in: Java magazin, Ausgabe 04/08, Seite 126ff.
[Roock-Wolf-4]	Stefan Roock, Henning Wolf - Agile Feature Driven Development, Teil 4: Projektmanagement, in: Java magazin, Ausgabe 05/08, Seite 126ff
[Royce]	Winston W. Royce, Managing the Development of Large Software Systems, http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf, letzter Zugriff 25.04.2008 erstmalig veröffentlich in: Proceedings, IEEE WESCON, August 1970, pages 1-9. Copyright © 1970 by The Institute of Electrical and Electronics Et)gineers.
[Rupp]	Chris Rupp, Requirements-Engineering und –Management, 3. Auflage, Hanser Verlag, 2004
[Salo1]	Outi Salo - Enabling Software Process Improvement in Agile Software Development Teams and Organisations, http://www.vtt.fi/inf/pdf/publications/2006/P618.pdf, letzter Zugriff

	24.04.2008
[Salo2]	Outi Salo - Improving Software Process in Agile Software Development Projects: Results from Two XP Case Studies, http://virtual.vtt.fi/virtual/agile/docs/publications/2004/2004_improving_software_process_in_sgile_software_development.pdf, letzter Zugriff 24.04.2008
[Schwaber]	Ken Schwaber, Agiles Projektmanagement mit Scrum, Microsoft Press, 2007, deutsche Übersetzung von: Ken Schwaber, Agile Project Management with Scrum, Microsoft Press, Redmond, 2007
[Stephens]	Matt Stephens, Doug Rosenberg – Extreme Programming Refactored: The Case against XP, Apress, 2003
[Sutherland2004]	Jeff Sutherland, Agile Development: Lessons learned form the First Scrum, http://jeffsutherland.com/scrum/FirstScrum2004.pdf, letzter Zugriff 25.04.2008
[SWEBOK]	http://www.swebox.org, letzter Zugriff 21.04.2008
[Versteegen2000]	Gerhard Versteegen – Projektmanagement mit dem Rational Unified Process, Springer Verlag, 2000
[Versteegen2004]	Gerhard Versteegen (Hrsg.) – Anforderungsmanagement, Springer Verlag, 2004
[Wallmüller2007]	Ernest Wallmüller, Software Process Improvement mit CMMI, PSP/TSP und ISO 15504, Hanser Verlag, 2007
[Winter]	Winter, Methodische objektorientierte Softwareentwicklung, dpunkt.verlag, 2005
[Wolf2005]	Henning Wolf, Stefan Roock, Martin Lippert, eXtreme Programming, Eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis, 2. Auflage, dpunkt.verlag, 2005