Marc Bless

# Scrum und die IEC 62304

Medizinische Software mit agilen Methoden normkonform entwickeln

**Version 1.8**
**Januar 2014**

agilecoach.de

Scrum und die IEC 62304

Marc Bless

# Scrum und die IEC 62304

*Medizinische Software mit agilen Methoden normkonform entwickeln*

Verlag: epubli GmbH, Berlin

**Impressum**



Druck und Verlag: epubli GmbH, Berlin, www.epubli.de

ISBN 978-3-8442-7354-0

Schlagworte:

*Medizinische Software, IEC 62304, agile Methoden, Scrum, Medizintechnik, Normkonformität, Software-Lebenszyklus, Software-Life-Cycle, Softwareentwicklung, Projektmanagement*

www.scrum-und-die-iec62304.de
www.agilecoach.de

Version 1.8

Gestaltung: Marc Bless

Baden-Baden, Januar 2014

# Inhaltsverzeichnis

# Vorwort von Dr. Alexander W. Röhm

Vor einigen Jahrzehnten begann im Software Engineering ein Prozess des Umdenkens, ausgelöst durch Erkenntnisse, dass Software Projekte, wie jede ökonomische Aktivität, in der Gegenwart von Unsicherheit, Komplexität und Wandel, geplant und durchgeführt werden müssen. Aus verschiedenen Disziplinen wurde ein neues Denkmodell genährt, dass letztlich im Jahre 2001 den Namen Agile Softwareentwicklung erhielt. Scrum ist eines der Frameworks, die helfen sollen, die Agile Software Entwickung umzusetzen. Es verspricht Effektivität, Effizienz und Qualität. "Das ist zu gut, um wahr zu sein!", habe ich mehr als ein Mal gehört. Wenn mit guten und vielfach wissenschaftlichen Argumenten das erste Misstrauen ausgeräumt ist, kommt der zweite Einwand. "Das ist auf unsere Situation nicht anwendbar." Diese Behauptung ist oft nach einer einfachen Betrachtung ausgeräumt. Mit einer Ausnahme: Regulierte Umgebungen wie die Medizintechnik. Hier müssen komplexe Betrachtungen durchgeführt werden, um zu Belegen, dass ein Prozess die Anforderungen der Regulierung erfüllt. Das vorliegende Buch tut dies mit eindrucksvollem Erfolg, da es über eine semi-formale Repräsentation der Requirements der IEC 62304 die komplette Abdeckung verifiziert.

Theorie ist die eine Seite der Medaille, die Praxis das Andere. Und es ist nicht erstaunlich, dass der Autor Scrum auch erfolgreich mit Software Engineering Teams in der Medizintechnik umgesetzt hat. Ich hatte die Ehre, diesen Schritt zusammen mit ihm zu wagen. Dieser Mut hat sich gelohnt, wir haben gesehen wie Entwickler an den Kunden denken, Software in planbarer Zeit und ohne Bugs entsteht, wie sich ändernde Anforderungen transparent in die Software und

notwendigen Artefakte einfließen und last-but-not-least alle wieder das Produkt und dessen Vision im Blick behalten und mit Leib und Seele Freude an ihrer Arbeit haben.

Ein Ratschlag an den Leser ist an dieser Stelle noch angebracht. Die Vorteile von Scrum entfalten sich nicht durch die Implementation eines Prozesses alleine. Stattdessen will es von den Werten und Zusammenhängen her erfasst und umgesetzt werden und beschreibt somit eher einen Startpunkt für eine Abenteurreise. Wenn der Leser agile Methoden für medizinische Systeme nutzen will, stellt dieses Buch wichtige Wegweiser auf, die während dieser Abenteuerreise sehr hilfreich sind.

Viel Erfolg und eine erlebnisreiche Reise!

Dr. Alexander W. Röhm
Senior Manager Software Engineering
Kofax plc.

*Freiburg, im November 2013*

# Vorwort von Frank Lange

Wenn ich bei Vorträgen oder in Gesprächen mit Kunden auf das Thema Scrum in der Medizintechnik zu sprechen komme, blicke ich oft in erstaunte Gesichter. "Dürfen wir das überhaupt?" ist meist die erste Frage, noch bevor der Nutzen geprüft wird. Klar, Scrum und andere agile Methoden sind mittlerweile in aller Munde und haben in anderen Branchen bereits zu vielen erfolgreichen Projekten geführt - aber in der Medizintechnik? Das geht dann vielen doch ein wenig zu weit.

Produktentwicklungen in der Medizintechnik sind gekennzeichnet von einem hohen Qualitätsbewusstsein. Es geht um Menschenleben, und jedes potentielle Produktisiko muss daher eliminiert oder zumindest auf ein Minimum reduziert werden. Normen und Richtlinien sind notwendig um die erforderlichen Prozesse einzufordern, mit deren Hilfe diese hohe Produktqualität sichergestellt werden kann. Als Prozess hat sich in der Medizintechnik das V-Modell etabliert und in den letzten Jahrzehnten dafür gesorgt, dass Medizinprodukte immer sicherer, zuverlässiger und kostengünstiger entwickelt und hergestellt werden können.

Doch seit ein paar Jahren ändert sich der Markt für Medizintechnik rasant. Unternehmen müssen sich immer stärker spezialisieren, um Spitzenpositionen in einem bestimmten Marktsegment - und sei es auch nur eine kleine Nische - zu erlangen. Je enger die Nische, umso wichtiger ist ein weltweiter Vertrieb, denn nur so ist der Gesamtmarkt dieser Nische groß genug, um die immer größer werdenden Entwicklungskosten zu decken. Diese Anforderung zur Spezialisierung und Internationalisierung gilt auch - und speziell für

kleinere - Unternehmen. Als dritte Anforderung kommt dazu, dass Entwicklungen in der Medizintechnik immer schnelleren Zyklen unterliegen. Ursache hierfür ist, dass Kunden Medizinprodukte nicht mehr nur mit dem direkten Wettbewerb vergleichen, sondern immer häufiger auch mit Consumerprodukten wie Smartphones, Tablets u.ä. Von Medizintechnikprodukten wird die gleiche einfache Handhabung erwartet, denn Einarbeitungszeiten kann sich im Gesundheitssystem keiner mehr leisten. Zeit ist Geld!

Agile Methoden, speziell Scrum, bieten erprobte Lösungen für diese neuen Anforderungen. Scrum ist seit weit über einem Jahrzehnt etabliert, viele durchgeführt Projekte belegen den Erfolg. Aber Scrum ist so... anders! Ist man es bisher in der Medizintechnik eher gewohnt, hunderte von Seiten in den Normen durchzuarbeiten um klare Arbeitsanweisungen daraus abzuleiten, bauen agile Methoden stärker auf Werten auf. Das agile Manifest besteht im wesentlichen aus nur vier Zeilen - kann das funktionieren? Und steht nicht im agilen Manifest gar etwas davon, dass nicht mehr dokumentiert werden muss? Das wäre ja ein direkter Widerspruch zur Norm! Solche scheinbaren Widersprüche können dazu führen, dass man sich lieber gar nicht erst mit Scrum auseinandersetzt, um auf der sicheren Seite zu bleiben. Auch wenn diese "sichere Seite" alles andere als sicher ist - siehe oben.

Marc Bless hat sich mit diesen Widersprüchen auseinandergesetzt. Intensiv. Er arbeitet als agiler Coach, Trainer und Berater und hat daher Scrum "im Blut". Gleichzeitig hat er jahrelange Erfahrungen im Bereich der Medizintechnik gesammelt und kennt sowohl die Normen und Anforderungen als auch das "Mindset" der Branche. Ihm geht es nicht um ein Entweder-Oder zwischen Normen und Scrum sondern um ein Sowohl-als-auch. Und Marc Bless ist Niemand, der sich mit Kompromissen zufrieden gibt. Was Sie in diesem

Buch lesen, hat den Praxistest bereits hinter sich. Es ist die Essenz aus vielen Jahren Erfahrung mit Scrum speziell in der Medizintechnik. Und es funktioniert.

Ich wünsche Ihnen viele gute neue Gedanken bei der Lektüre dieses Buches und eine erfolgreiche Umsetzung in die Praxis!

Frank Lange
Pfinztal, den 08. Juli 2013

# Warum Agil und Medizintechnik?

## Schneller Markteinstieg

Medizintechnik und die Entwicklung entsprechender Medizinprodukte befinden sich in einem regulierten Umfeld. Gesetze, Normen und Richtlinien geben mehr oder weniger harte Rahmenbedingungen vor, in deren Grenzen ein Medizinprodukt umgesetzt werden darf. Die Einhaltung dieser Rahmenbedingungen führt im klassischen Projektmanagement oft zu langen Entwicklungszyklen, in denen wir erst sehr spät feststellen können, ob unsere anfänglichen Annahmen mit ihrer Realisierung auch tatsächlich marktreif und im Interesse der Anwender umgesetzt wurden.

Lange Entwicklungszeiten für Produktlebenszyklen sind wirtschaftlich heutzutage nicht mehr sinnvoll. Eine kurze Time-to-Market wird immer wichtiger, um den Marktbegleitern einen Schritt voraus zu sein und innovative Produktideen schnell als erster am Markt platzieren zu können. Und selbst Me-Too-Ansätze in der Produktentwicklung profitieren von einer schnellen Umsetzung der Ideen Anderer - schließlich wollen Sie Ihr Me-Too-Produkt anbieten können, lange bevor der Innovator Ihren Markt mit der nächsten, neuen umgesetzten Idee zu Ihren Ungunsten verändert.

# Kundenzufriedenheit und Qualität

Der Kunde eines Medizinproduktes wird immer mündiger und entscheidet selbst über den Erfolg einzelner Produkte und Dienstleistungen. Dies erfordert bereits bei der Entstehung neuer Produktideen eine gewisse Nähe zum Anwender oder sogar die entwicklungsbegleitende Einbindung echter Anwender. In traditionellen Vorgehensweisen kann der Anwendungserfolg oft erst ganz am Ende des Prozesses ermittelt werden. Ein entsprechendes Reagieren auf Kunden und Anwender[1] sowie deren wertvolles Feedback kann dann erst in einem nächsten Produkt, Projekt oder Release stattfinden und sich auswirken.

Der Kunde bewertet ein Produkt nicht nur nach seinen Funktionen und der einfachen Handhabung, sondern auch nach seiner Qualität und Fehlerfreiheit. Dabei spielen oft schon Kleinigkeiten eine große Rolle, die zu einer negativen Bewertung eines Produktes führen können. In sozialen Netzwerken und Produktbewertungsportalen multiplizieren sich diese Effekte mittlerweile soweit, dass wir feststellen müssen: über Sieg oder Niederlage eines Produktes kann heute die breite Masse der Anwender entscheiden, unabhängig von Marketing- und Werbemaßnahmen.

Dabei verlassen wir uns im Regelfall auf die Empfehlungen unserer (oftmals sehr weitläufigen) Freunde und Kontakte. Wir vertrauen mehr auf deren Urteilsvermögen und Erfah-

[1] Im weiteren Text wird nicht mehr strikt zwischen Kunde (derjenige, der das Produkt bezahlt) und Anwender (derjenige, des das Produkt tatsächlich benutzt) unterschieden, um die Lesbarkeit des Textes nicht unnötig zu strapazieren.

rungen als auf ausgeklügelte Werbebotschaften. Hier gilt es also, von Anfang an eine stets hohe Qualität sowie einen großen Anwendernutzen an den Markt zu bringen, um die Kundenzufriedenheit zu maximieren.

# Agile Methoden - ein Mysterium?

Der Einsatz agiler Management- und Entwicklungsmethoden liegt auf der Hand. Schnelle Entwicklungszyklen, Einbindung und Feedback echter Anwender, sowie höchste Qualitätsansprüche können durch agile Methoden angeblich nachhaltig realisiert werden. Des weiteren haben agile Methoden in den letzten Jahren ein breites Publikum erreicht und sind in aller Munde. Immer mehr Unternehmen widmen sich diesem Thema, also muss etwas dran sein. Oder handelt es sich bei Scrum und co. einfach nur um einen aktuellen Trend, dem nächstes Jahr wieder etwas anderes folgt?

Viele Versprechen, aber auch Missverständnisse kursieren bezüglich agiler Methoden:

- „Scrum ist chaotisch."
- „Jeder macht, was er will."
- „Es wird nichts mehr dokumentiert."
- „Scrum funktioniert nur bei kleinen Web-Projekten."
- „Agilität funktioniert nicht bei komplexen Medizintechnik-Projekten."
- „Agile Methoden funktionieren in unserer Organisation nicht."
- „Scrum ist die Lösung all unserer Probleme."
- „Durch Agilität werden wir effizienter."

- „Die IEC 62304 beschreibt einen konkreten Software-Life-Cycle-Prozess und muss eins-zu-eins umgesetzt werden."
- „Wir müssen ein V-Modell einsetzen, um normkonform zu arbeiten."

Die Kombination der agilen Welt und des regulierten Umfeldes der Medizinproduktentwicklung bereitet vielen Leuten Kopfzerbrechen und führt zu großer Unsicherheit. In dieser Unsicherheit lauert die Gefahr, agile Methoden in bestehende, normkonforme Prozesse zu pressen, ohne dass sich der Vorteil echter Agilität entfalten kann.

Eine vereinfachte, beschränkte oder missverstandene Anwendung agiler Methoden kann dazu führen, dass das Scheitern eines Produktes oder Projektes eben diesen agilen Methoden zugerechnet wird:

- „Agil haben wir schon probiert, das hat ja auch nicht funktioniert."
- „Agil hat bei uns alles nur schlimmer gemacht."
- „Scrum hat uns mit Problemen konfrontiert, die wir vorher gar nicht hatten."

Die Einführung von Scrum erfordert sehr viel Verständnis der zugrundeliegenden agilen Prinzipien, um nicht in die Falle zu laufen, fundamentale Fehler zu begehen. Diese Fehler mögen für den Unerfahrenen wie unwichtige Kleinigkeiten aussehen und oftmals ist es einfach bequemer, eine bestimmte Praktik oder Methode nicht einzuführen, da sie das bestehende Gefüge und die Unternehmenskultur im Innersten treffen können.

Wir[2] wissen heute aus Erfahrung, dass agile Methoden wie Scrum funktionieren, wenn sie richtig und konsequent ein- und umgesetzt werden.

# Verbindung zweier Welten

Können nun aber beide Welten effektiv nebeneinander existieren? Wie viel Agilität passt in die normativen Anforderungen? Welche agilen Praktiken eignen sich für einen regulierten Entwicklungsprozess?

Eine für uns grundlegende Aussage der Norm lautet, dass grundsätzlich jeder beliebige wasserfallartige, iterative oder evolutionäre Prozess verwendet werden kann[3].

Die grundsätzliche Frage lautet: **wie kann die Normkonformität agiler Prozesse und Methoden nachgewiesen werden?**

Das vorliegende Buch gibt Antworten auf diese Frage und beschreibt eine vollständige, IEC 62304 konforme Implementierung von Scrum als Projektmanagementmethode, die mit agilen Entwicklungspraktiken aus dem Extreme Programming angereichert ist.

---

[2] Ich selbst aus jahrelanger Erfahrung als agiler Coach und viele meiner Kollegen, die sich intensiv damit beschäftigen, Unternehmen aller Branchen erfolgreich mit agilen Methoden und Prozessen zu transformieren.

[3] IEC 62304, Annex B

# Von der Norm zum Prozess

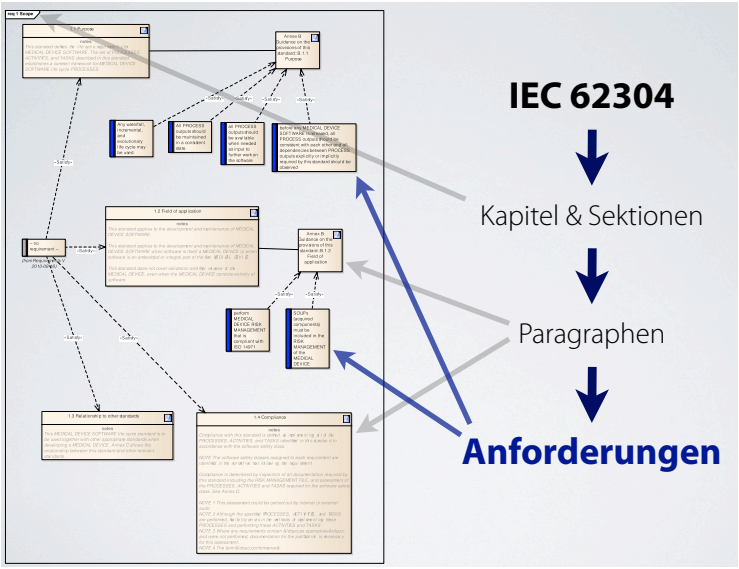
Der typische Weg für viele Qualitätsmanager und Prozessbeauftragte, eine Norm umzusetzen, ist die direkte Abbildung der Norminhalte auf einen Prozess. Damit erhalten wir im Regelfall einen vollständig normkonformen Prozess und dies ist in einer klassischen Organisation oft auch die Zielsetzung. Dort geht es nur in zweiter Linie um Effizienz der eingesetzten Methoden.

Die Abbildung der Norm auf einen eigenen, selbst definierten und normkonformen Prozess erfordert sehr viel mehr Aufwand, da dies eine detaillierte Auseinandersetzung mit der Norm voraussetzt und ein tiefes Verständnis des „Geistes" einer Norm entwickelt werden muss. Mit „Geist" ist die Meta-Ebene einer Norm gemeint, das heisst die darunter liegenden Prinzipien und die grundsätzliche Zielsetzung. Durch die Erarbeitung dieser Perspektive auf die normativen Inhalte wird es möglich, aus dem sequentiell beschriebenen Ablauf der Norm auszubrechen und zu einer neuen, teilweise kontra-intuitiven Umsetzung der Prozesse zu gelangen.

Um zu einem normkonformen, agilen Prozess zu gelangen, wurden die im Folgenden beschriebenen Transformationen der Norminhalte durchgeführt.

## Anforderungen an den Software-Life-Cycle-Prozess

Zunächst wurde die Struktur der IEC 62304 vollständig aufgebrochen. Aus den einzelnen Kapiteln, Sektionen und jedem darin enthaltenen Paragraphen wurden die eigentlichen Anforderungen an einen Software-Life-Cycle-Prozess extrahiert.



## Beispiel 1

Aus dem Paragraphen

> ***§ 7.1.2 Identify potential causes of contribution to a hazardous situation***
>
> *The MANUFACTURER shall identify potential causes of the SOFTWARE ITEM identified above contributing to a hazardous situation.*
>
> *The MANUFACTURER shall consider potential causes including, as appropriate:*
>
> *a) incorrect or incomplete specification of functionality;*
> *b) software defects in the identified SOFTWARE ITEM functionality;*
> *c) failure or unexpected results from SOUP;*
> *d) hardware failures or other software defects that could result in unpredictable software operation; and*
> *e) reasonably foreseeable misuse.*
>
> *[Class B, C]*

wurden die folgenden, fünf Anforderungen an einen Software-Life-Cycle-Prozess extrahiert:

1. [B,C] consider potential causes including incorrect or incomplete specification of functionality
2. [B,C] consider potential causes including software defects in the identified SOFTWARE ITEM functionality
3. [B,C] consider potential causes including failure or unexpected results from SOUP

4. [B,C] consider potential causes including hardware failures or other software defects that could result in unpredictable software operation
5. [B,C] consider potential causes including reasonably foreseeable misuse

In diesem typischen Fall wurden die aufgelisteten Anforderungen aus dem Paragraphen in einzelne Anforderungen an den Software-Life-Cycle-Prozess extrahiert. Dies hat den Vorteil, dass für jede einzelne Anforderung die einfachste Möglichkeit gefunden werden kann, sie zu befriedigen.

## Beispiel 2

Aus dem Paragraphen

> ***§ 7.3.1 Verify RISK CONTROL measures***
>
> *The implementation of each RISK CONTROL measure documented in 7.2 shall be VERIFIED, and this VERIFICATION shall be documented. [Class B, C]*

wurden die folgenden beiden Anforderungen an einen Software-Life-Cycle-Prozess extrahiert:

1. [B, C] verify the implementation of each RISK CONTROL measure
2. [B, C] document the verification of implemented risk control measures

Auch hier wurden auf den ersten Blick zusammenhängende Fragmente aus dem Paragraphen aufgeteilt in einzelne Anforderungen. Die beiden Aspekte „Implementation“ und „Dokumentation“ können mit verschiedenen Praktiken um-

gesetzt werden, deswegen ist eine Trennung der beiden sinnvoll.

### Vollständige Menge aller Anforderungen

Das Ergebnis dieser Transformation von Paragraphen in einzelne Anforderungen ist eine vollständige Menge aller dieser Anforderungen an den Software-Life-Cycle-Prozess.

Die Struktur der Norm suggeriert einen sequentiellen Entwicklungsablauf, der einem agilen Ansatz im Wesen widerspricht. Gleichzeitig stecken in dieser Struktur viele redundante Anforderungen, die zu einem sequentiellen Ablauf eher orthogonalen Charakter haben, wie zum Beispiel das gesamte Thema Risikomanagement.

Aus diesem Grund wurde die Menge aller Anforderungen gruppiert und in neue, einzelne Cluster eingeordnet, welche einzelne Entwicklungsthematiken inhaltlich abgeschlossen handhabbar machen.

## Dokumente und Prozesse

Den Anforderungen in den Clustern wurden konkrete agile und klassische Praktiken und Artefakte zugeordnet. Die Zuordnungen einzelner Praktiken und Artefakte zu einzelnen Anforderungen wurden so vorgenommen, dass die Gesamtmenge aller normativen Anforderungen vollständig erfüllt wird. Jede einzelne Zuordnung ist schriftlich begründet, um einen Nachweis der Anforderungserfüllung führen zu können.

Das Ergebnis aller Praktiken und Artefakte mitsamt ihren Begründungen findet sich im Hauptteil dieses Buches.

Das wichtigste Prinzip für die Zuordnung von Praktiken und Artefakten ist Einfachheit („Simplicity“) und bedeutet, die minimale, aber vollständige Befriedigung der normativen Anforderungen zu finden.

**Praktiken** sind dabei im Wesentlichen Prozesse, Reviews, Meetings, Besprechungen und Workshops.

**Artefakte** sind sämtliche Dokumente, Ergebnisse, funktionierende Software, Pläne und Beschreibungen, welche im Entwicklungsverlauf entstehen und eine Rolle spielen.

## Nachweis der Normkonformität

Der Nachweis der Normkonformität wird für jede einzelne Praktik und jedes einzelne Artefakt in diesem Buch aufgeführt. Im Hauptteil des Buches wird dafür in jedem entsprechenden Unterkapitel auf die Paragraphen der Norm und deren im Einzelnen spezielle Anforderungen an den Software-Life-Cycle-Prozess verwiesen, sowie die Begründung aufgeführt, wie die Normkonformität zu jeder Anforderung hergestellt wird.

Der nachgelagerte Teil des Buches führt in tabellarischer Form durch die einzelnen Sektionen und Paragraphen der Norm und Listet die jeweiligen Praktiken/Artefakte auf mitsamt der Begründung der Normkonformität.

Dadurch wird für Anwender dieser Methodik sehr schnell ersichtlich, welche Praktiken und Artefakte für welche Paragraphen und Anforderungen aus der Norm eingesetzt werden können. Auditoren wird sehr leicht aufgezeigt, welche Paragraphen der Norm mit welchen Praktiken und Artefakten umgesetzt werden können.

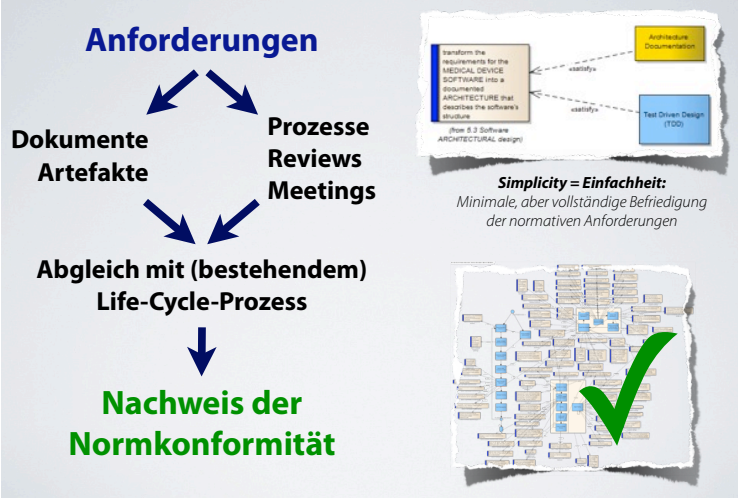
## Angepasste Vorgehensmodelle

Das in diesem Buch vorgestellte Prozessbild gibt dem Leser einen definierten, agilen Prozess vor. Dies hat zum Einen den Vorteil, dass sich der Prozess „by the book" einführen und etablieren lässt. Es birgt zum Anderen jedoch die Gefahr, dass sich dieser Prozess nach einer Einführung nicht mehr weiter entwickelt. Die Weiterentwicklung von Methoden und Prozessen auf die jeweiligen kulturellen und organisatorischen Bedürfnisse in einem Unternehmen oder Team ist aber einer der wichtigsten Grundpfeiler aller Agilität. Es ist also nicht nur wunschenswert, dass sich Prozesse regelmäßig ändern, sondern Pflicht, wenn man eine wirklich agile Organisation erzeugen möchte.

Die in den folgenden Kapiteln aufgeführten Referenzen zwischen Norm und Umsetzung können sehr leicht dafür genutzt werden, den Einfluß von anstehenden Prozessänderungen auf die Normkonformität bewerten zu können.

## Nachweis der Normkonformität anderer Prozesse

Die beschriebene Systematik, welche zu der in diesem Buch dargestellten Prozesslandschaft geführt hat, kann sowohl dafür genutzt werden, Software-Life-Cycle-Prozesses vollständig neu zu definieren, als auch die Normkonformität bestehender Software-Life-Cycle-Prozesse nachzuweisen bzw. in Frage zu stellen.




Grundsätzlich muss deutlich gesagt werden, dass beide Ansätze einen nicht unerheblichen Aufwand mit sich bringen, der im regulierten Umfeld jedoch in jedem Fall durchgeführt werden muss.

Der Nachweis der Normkonformität durch den Abgleich mit einem bestehenden oder neuen Software-Life-Cycle-Prozess ist aufwändig, mit dieser Systematik jedoch effektiv machbar.

# Überblick der Prozesse und Artefakte

Das nachfolgende Diagramm zeigt den gesamten Überblick aller Prozesse und Artefakte, welche aus normativen und agilen Aspekten benötigt werden.

## Agile Prozesse:

*Acceptance Testing*
*Clean Code / SOLID Principles*
*Continuous Integration*
*Early Integration / Early Testing*
*Exploratory and Manual Testing*
*Integration Testing*
*Product Backlog Grooming*
*Release Planning Meeting*
*Scrum*
*Sprint Planning*
*Sprint Retrospective*
*Sprint Review*
*Test Automation*
*Test-Driven Development (TDD)*
*Unit Testing*
*Zero Bug Tolerance*

Diese Prozesse bezeichne ich als „Agile Prozesse", da sie entweder aus dem Umfeld der agilen Methoden entsprungen sind oder sich in diesem Umfeld als integrale Bestandteile etabliert haben.

## Nicht-agile Prozesse:

*Problem Communication*
*Quality Management Prozess*
*Risk Analysis and Management*
*Software Maintenance Process*
*Team Kick-Off Meeting*
*Usability Process*

Diese Prozesse bezeichne ich als „Nicht-agile Prozesse", da sie dem Umfeld der klassischen Methoden entsprungen sind und nicht zwangsläufig in agilen Projekten eingesetzt werden.

## Agile Artefakte:

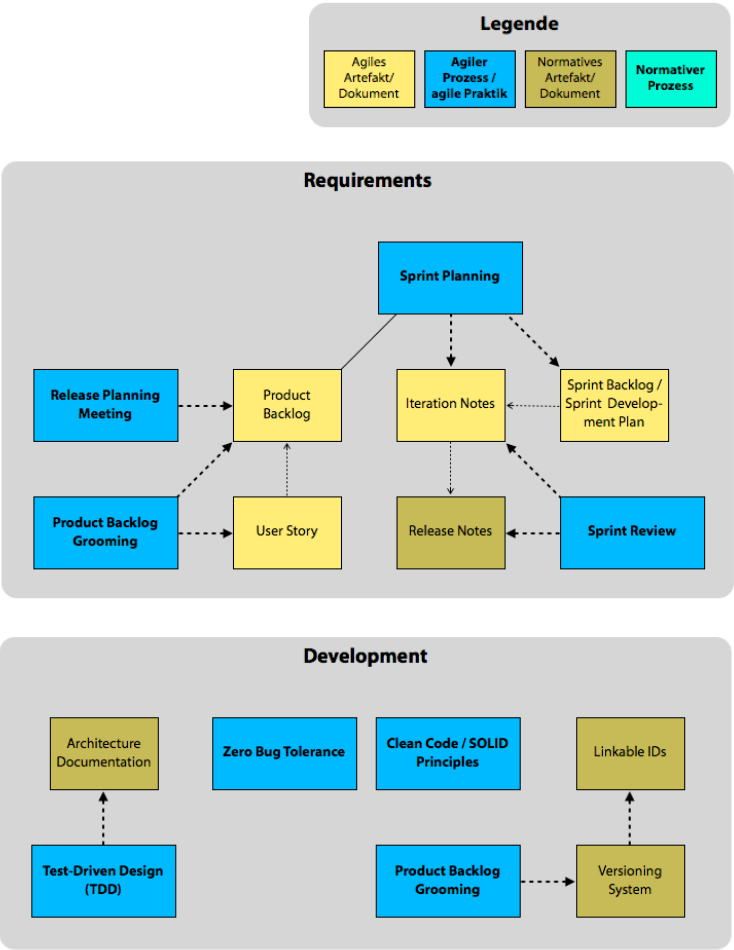
*Definition of Done*
*Definition of Ready*
*Iteration Notes (Sprint Notizen)*
*Product Backlog*
*Sprint Development Plan = Sprint Backlog*
*Team Charter / Working Agreements*
*User Story*

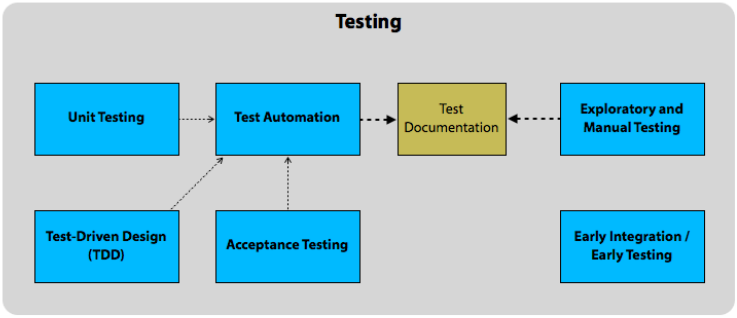
Analog zu den sogenannten „Agilen Prozessen" bezeichne ich diese Artefakte als „Agile Artefakte", da sie entweder aus dem Umfeld der agilen Methoden entsprungen sind oder sich in diesem Umfeld als integrale Bestandteile etabliert haben.

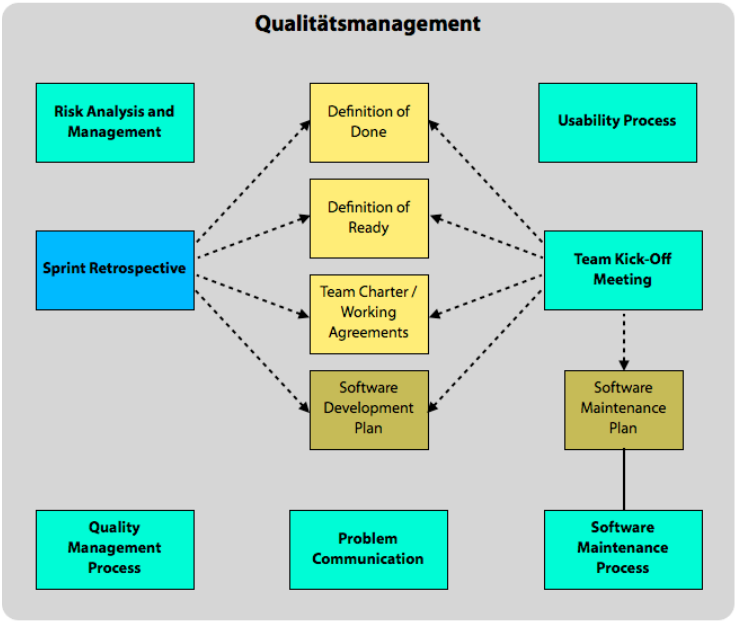
## Nicht-agile Artefakte:

*Architecture Documentation*
*Linkable IDs*
*Release Notes*
*Software Development Plan*
*Software Maintenance Plan*
*Test Documentation*
*Versioning System*

Analog zu den sogenannten „Nicht-agilen Artefakten" bezeichne ich diese Artefakte als „Nicht-agile Artefakte", da sie dem Umfeld der klassischen Methoden entsprungen sind und nicht zwangsläufig in agilen Projekten eingesetzt werden.


Legende
Agiles Artefakt/ Dokument
Agiler Prozess / agile Praktik
Normatives Artefakt/ Dokument
Normativer Prozess
Requirements
Sprint Planning
Release Planning Meeting
Product Backlog
Iteration Notes
Sprint Backlog / Sprint Develop-ment Plan
Product Backlog Grooming
User Story
Release Notes
Sprint Review
Development
Architecture Documentation
Zero Bug Tolerance
Clean Code / SOLID Principles
Linkable IDs
Test-Driven Design (TDD)
Product Backlog Grooming
Versioning System


Testing
Unit Testing
Test Automation
Test Documentation
Exploratory and Manual Testing
Test-Driven Design (TDD)
Acceptance Testing
Early Integration / Early Testing


Qualitätsmanagement
Risk Analysis and Management
Definition of Done
Usability Process
Definition of Ready
Sprint Retrospective
Team Kick-Off Meeting
Team Charter / Working Agreements
Software Development Plan
Software Maintenance Plan
Quality Management Process
Problem Communication
Software Maintenance Process

Im folgenden Hauptteil dieses Buches werden die einzelnen Praktiken und Artefakte in jeweils eigenen Kapiteln im Detail beschrieben. Die Struktur dieser einzelnen Kapitel ist folgendermaßen aufgebaut:

- **Beschreibung** - kurze Erklärung der Praktik oder des Artefaktes für das bessere Verständnis, welche Absicht dahintersteckt und welche Ergebnisse erwartet werden.

- **Verantwortliche/beteiligte Rollen** - Liste der üblicherweise beteiligten Rollen für eine bessere Einschätzung, welche Rollen in einem bestehenden Unternehmenskontext für den Einsatz der Praktik oder des Artefaktes in Frage kommen. Die in diesem Buch aufgeführten Rollen müssen nicht genau so übernommen werden, sie dienen eher dem Aufzeigen potentieller, sinnvoller Möglichkeiten.

- **Referenzen** - Liste von Büchern, Artikeln und Webseiten, welche sich mit der Praktik bzw. dem Artefakt detailliert auseinandersetzen.

- **Umgesetzte Anforderungen aus der Norm** - Liste der direkt aus dem Normtext abgeleiteten Anforderungen an den Software-Life-Cycle-Prozess. Zu jeder Anforderung aus der Norm liegt eine Beschreibung bzw. Begründung vor, aus der hervorgeht, aus welchem Grund die Praktik oder das Artefakt geeignet ist, diese Anforderung umzusetzen und die Herstellung der Normkonformität herzustellen.
  Folgende Spalten sind in den Tabellen enthalten:
  - **§** - Paragraph der IEC 62304, in dem die Anforderung an den Software-Life-Cycle-Prozess zu finden ist. Dabei kann es sich sowohl um den tatsächlichen Para-

graphen in der Norm handeln, als auch um die entsprechenden Anmerkungen in Annex B der Norm.
  - **SK** - Sicherheitsklassifizierung, für welche die Anforderung aus der Norm umzusetzen ist.
  - **Normtext** - Originaltext aus dem entsprechenden Paragraphen der IEC 62304 bzw. dessen Anmerkungen in Annex B oder eine davon abgeleitete Anforderung an den Software-Life-Cycle-Prozess.
  - **Herstellung der Normkonformität** - Beschreibung bzw. Begründung, aus welchem Grund die Praktik oder das Artefakt geeignet ist, diese Anforderung umzusetzen.

- **Verknüpfungen zu anderen Praktiken und Artefakten** - Querverweise zu anderen Praktiken und Artefakten in diesem Dokument, welche einen direkten Bezug zur beschriebenen Praktik oder dem Artefakt besitzen.

# Agile Werte und Prinzipien

## Das (mißverstandene) agile Manifest

Im Jahre 2001 kamen in Snowbird, Utah (USA) 17 Menschen zusammen, die sich mit Software-Entwicklungsprojekten beschäftigt hatten und während der 90er Jahre des letzten Jahrtausends zu ähnlichen Erkenntnissen über Erfolg und Mißerfolg dieser Projekte gekommen waren. Das Ergebnis des zweitägigen Treffens wurde im „Manifest für agile Softwareentwicklung“ festgehalten.

Die Formulierungen der vier Wertgegenüberstellungen in diesem Manifest sind eine von vielen Ursachen für Mißverständnisse und falsche Interpretationen, was „agil“ tatsächlich bedeutet. Dies führt unter anderem zu den Annahmen, dass in einem agilen Projekt (a) keine Prozesse mehr beachtet werden („jeder macht, was er will“), (b) keine Dokumentation mehr geschrieben wird und (c) keinerlei Planung mehr stattfindet.

## Manifest für Agile Softwareentwicklung

Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen.
Durch diese Tätigkeit haben wir diese Werte zu schätzen
gelernt:

**Individuen und Interaktionen** mehr als Prozesse und Werkzeuge

**Funktionierende Software** mehr als umfassende Dokumentation

**Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung

**Reagieren auf Veränderung** mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.

| | | |
|---|---|---|
| Kent Beck | James Grenning | Robert C. Martin |
| Mike Beedle | Jim Highsmith | Steve Mellor |
| Arie van Bennekum | Andrew Hunt | Ken Schwaber |
| Alistair Cockburn | Ron Jeffries | Jeff Sutherland |
| Ward Cunningham | Jon Kern | Dave Thomas |
| Martin Fowler | Brian Marick | |

Es gibt durchaus Umfelder, in denen ohne Probleme genau so gearbeitet werden kann. Nehmen wir beispielsweise ein innovatives Lean Start-Up Unterfangen, bei dem wir noch nicht einmal wissen, wer der Kunde des Produktes sein wird und was ein potentieller Anwender tatsächlich benötigen könnte. In solch einem Fall stehen uns die im agilen Manifest weniger wertgeschätzten Punkte eher im Weg, als dass sie zu einem wertvollen Erkenntnisgewinn beitragen würden.

Auf der anderen Seite gibt es gerade in der Medizintechnik immer wieder Unternehmen, die auf Grund von viel zu weit gefassten Interpretationen der Normen in einer Prozesslandschaft gelandet sind, in der Werkzeuge im Detail definiert, Dokumente selbst für irrelevante Ergebnisse erstellt, Verträge zwischen Abteilungen festgezurrt, sowie die ursprünglichen Pläne in Blei gegossen werden müssen. In solchen Fällen befinden wir uns nicht mehr nur in einem regulierten Umfeld, sondern in einem überregulierten Umfeld, welches aus Furcht vor normativer Sanktionierung immer weiter mit Restriktionen versehen wird, anstatt den Prozessrahmen auf ein sinnvolles Maß zurück zu schrauben und eine effiziente und effektive Produktentwicklung zu ermöglichen.

Die vier Wertgegenüberstellungen im agilen Manifest dürfen nicht wie ein „entweder-oder" betrachtet werden. Wir müssen sie vielmehr als acht einzelne und voneinander unabhängige Regler verstehen, deren Stärke an die jeweilige Situation, das jeweilige Projekt und das entsprechende Umfeld angepasst werden müssen.

Unser Ziel dabei ist immer, die Werte auf der linken Seite maximal zu leben. Individuen und Interaktionen, funktionierende Software, Zusammenarbeit mit dem Kunden und Re-

agieren auf Veränderung sind allesamt wichtige Erfolgsfaktoren für Softwareprojekte.

In der Medizintechnik befinden wir uns in einem Umfeld, in dem wir viel mehr darauf achten müssen, dass definierte Prozesse zum Einsatz kommen, Werkzeuge auf Risiken geprüft werden, sowie mit nachvollziehbaren Absprachen und Plänen gearbeitet wird. Abhängig von der jeweiligen Produktkategorie und der Sicherheitsklassifizierung des Produktes ergeben sich mehr oder weniger hohe Notwendigkeiten für die Werte auf der rechten Seite des agilen Manifests.

# Agile Basiswerte

**Commitment (Zusage) -** Sei bereit, Dich einem Ziel zu verpflichten. Agile Methoden geben den Leuten alle Autorität, um ihre Zusagen umzusetzen und zu erreichen.

**Einfachheit** - Wähle die einfachste technische Lösung, um den größtmöglichen Nutzen und Wert für den Anwender bereit zu stellen.

**Feedback** - Zeige Projektergebnisse früh und oft, hol Dir die Rückmeldungen dazu ein und passe Deine Vorgehensweise darauf an.

**Fokus -** Erledige nichts außer Deiner Arbeit. Fokussiere all Deine Bemühungen und Fähigkeiten darauf, an Deinen Zusagen zu arbeiten. Kümmere Dich um nichts anderes.

**Kommunikation** - Alle im Team kommunizieren täglich miteinander und arbeiten zusammen von den Anforderungen bis zum Code, um die beste Lösung für unser Problem zu finden.

**Mut -** Erzähle die Wahrheit über den Projektfortschritt und unsere Schätzungen. Sei mutig, um Zusagen zu geben, fokussiert zu handeln, offen zu sein und respektvoll miteinander umzugehen.

**Offenheit -** Agile Methoden macht jeden Aspekt eines Projektes für alle sichtbar. Liefere alle Informationen zeitnah und transparent. Stelle ein Umfeld her, in dem

Wahrheit und Ehrlichkeit ein sicherer Raum gegeben wird.

**Respekt -** Leute werden durch ihren Hintergrund und ihre Erfahrungen geformt. Es ist wichtig, die unterschiedlichen Menschen zu respektieren, aus denen ein Team besteht.

# Prinzipien hinter dem agilen Manifest

In diesem Abschnitt werden alle zwölf agilen Prinzipien im Detail beschrieben und die Bedeutung in der Medizintechnik herausgearbeitet. Wir werden sämtliche Textfragmente der Prinzipien im Einzelnen untersuchen und die jeweils zugehörigen Werte, tiefer gehenden Prinzipien und abgeleiteten Praktiken bestimmen.

## Agiles Prinzip 1: Den Kunden zufrieden stellen

Das erste Prinzip unter dem agilen Manifest lautet:

> *"Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen."*

**"Unsere"** – wer sind wir? Wer ist das Team? Das Team besteht aus allen Personen, welche benötigt werden, um aus den Anforderungen eine funktionierende Software zu erzeugen, diese zu testen, die Anforderungen zu priorisieren und deren Wert zu bestimmen, und welche sich fokussiert auf ein Commitment konzentrieren können.

Werte: Kommunikation

Prinzipien: Vielfalt,
Bevollmächtigtes Team

Praktiken: Ganzes Team

Es gibt keinen besonderen Bezug zur Medizintechnik. Teams werden in allen Kontekten für Projekt- und Produktentwicklung eingesetzt, unabhängig von Medizintechnik und agilen Methoden.

**"höchste Priorität"** – das Team soll nicht abgelenkt werden, sondern seinen Fokus auf die wichtigsten Dinge legen können. Um dem Team dabei zu helfen, sich

nicht ablenken zu lassen, soll es in einem sicheren Umfeld geschützt sein, so dass keine externen Einflüsse die Aufmersamkeit des Teams auf sich ziehen können.

Werte: Commitment,
Fokus

Prinzipien: Limitierung der laufenden Arbeiten,
Ökonomie

Praktiken: Product Backlog

Auch hier existiert kein Bezug zur Medizintechnik. Ein fokussiertes und konzentriertes Arbeiten der beteiligten Teams ist ein grundsätzlich erstrebenswerter Zustand.

**"den Kunden"** – wer ist das? In einer idealen Welt entspricht dieser Kunde allen unterschiedlichen Typen von Anwendern der entstehenden Software. Da es meist nicht möglich ist, all diese Personen und Rollen in einem Team "an Bord" zu haben, kann der Kunde auch durch einen Product Owner repräsentiert werden.

Werte: Kommunikation,
Respekt

Praktiken: Aktive Einbindung der echten Anwender

Sobald wir es mit Anwendern unseres Produktes zu tun haben, entsteht die Notwendigkeit, sich mit allen Aspekten der Risikoanalyse auseinander zu setzen, wie z.B. Risiken für Anwender und Patienten. Genauso

notwendig ist die Analyse von Anforderungen und der Gebrauchstauglichkeit nach der Usability-Norm IEC 62366.

**"frühe Auslieferung"** – die ersten Ergebnisse der gewünschten Software müssen den Anwendern so früh wie möglich gegeben werden, um deren Feedback zu erhalten und weitere Einsichten darüber zu gewinnen, was in unserem Produkt tatsächlich benötigt und gewünscht wird.

Werte: Feedback,
Kommunikation

Prinzipien: Regelmäßige Auslieferung,
Inkremente Entwicklung

Praktiken: Inkrementelles Deployment,
Kleine Releases

Im Umfeld der Medizintechnik sind wir klassischerweise in der Lage, Rückmeldungen durch klinische Validierungen und Markterprobungsphasen nach der Entwicklung einzuholen. Die Phrase „nach der Entwicklung" schließt jedoch eine frühe Auslieferung beinahe aus. Oft bedeuten die Voraussetzungen, um mit unfertigen Produkten zu Anwendern und Patienten gehen zu dürfen, einen hohen Aufwand an Dokumentation und Unbedenklichkeitsnachweisen. Hier benötigen wir neue Möglichkeiten, diese Aufwände zu reduzieren und dennoch ohne Risiken eine frühe Auslieferung zu nutzen. Das in diesem Buch beschriebene Vorgehensmodell löst unter anderem genau diese Problematik auf.

**"kontinuierliche Auslieferung"** – es reicht nicht aus, nur die ersten Ergebnisse der Software zu zeigen. Es ist vielmehr notwendig, auf einer regelmäßigen Basis Zwischenergebnisse vorzustellen.

Werte: Commitment,
Feedback

Prinzipien: Flow,
Iterative Vorgehensweise

Praktiken: Produkt Demonstration,
Sprint

Es existiert kein nennenswerter Bezug zur Medizintechnik. Wichtig ist die oben beschriebene frühe Auslieferung. Diese kontinuierlich zu gestalten, ist nur eine konsequentere Fortführung der Praktik.

**"wertvolle Software"** – wer ist in der Lage, den Wert einer Software zu definieren? Auch hier ist es der Kunde bzw. der Anwender, welcher die Anforderungen nach ihrem Geschäftswert und Nutzen priorisieren kann.

Werte: Commitment,
Fokus

Prinzipien: Ökonomie,
Business Value

Praktiken: Product Backlog

Um für uns in der Medizintechnik „wertvolle" Anforderungen ausfindig zu machen, genügt es nicht, einzig

und allein den zahlenden Kunden zu adressieren. Wir müssen vielmehr alle Anwender und Patienten in Betracht ziehen. Auch hier befinden wir uns sofort wieder in den Themen Risikomanagement und Usability Als Medizinproduktehersteller ist bei gewissen Themen die Durchführung von normativ geforderten Tätigkeiten mindestens genau so wichtig wie die Entwicklung neuer Funktionalität.

**"zufrieden zu stellen"** – wie kann ein Team wissen, ob seine Ergebnisse zufriedenstellend sind? Eine Definition-of-Done und durchlaufene Akzeptanz-Tests sind notwendig für jede Anforderung an die Software.

| | |
|---|---|
| Werte: | Commitment,<br>Fokus |
| Prinzipien: | Business Value,<br>Qualität,<br>Schnelles Scheitern |
| Praktiken: | Definition of Done,<br>Aktive Einbindung der echten Anwender,<br>Akzeptanz-Tests,<br>Sprint Review |

Ein direkter Bezug zur Medizintechnik findet sich auch in diesem Punkt wieder in der Usability-Norm IEC 62366, aber auch in der Validierung unserer Produkte. Im Gegensatz zum V-Modell, in dem die Validierung des Produktes sehr spät im Entwicklungsprozess vorgesehen ist, benötigen wir eine Möglichkeit, in kurzen Zyklen unfertige Teilprodukte validieren zu lassen. Das

in diesem Buch beschriebene Vorgehensmodell löst diese Problematik mit relativ einfachen Mitteln auf.

Wenn wir alle Rollen und Praktiken zusammenfassen, die dieses erste Prinzip enthält, kommen wir zu einem voll funktionstüchtigen Team, welches mit einer iterativ-inkrementellen Herangehensweise die wichtigsten Dinge für den Anwender umsetzt.

## Agiles Prinzip 2: Änderungen willkommen heissen

Das zweite Prinzip unter dem agilen Manifest lautet:

> *"Anforderungsänderungen sind auch spät während der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden."*

**"Anforderungsänderungen"** – wir dürfen uns bezüglich der Gewissheit einer Anforderung nie sicher sein, da die Realität kein statisches System ist. Menschen, Märkte, Bedürfnisse und grundsätzlich alle Dinge im Umfeld unserer Produktentwicklung befinden sich in konstantem Fluß und Veränderung. Dies müssen wir in jedem einzelnen Aspekt unserer Arbeit akzeptieren und berücksichtigen.

| | |
|---|---|
| Werte: | Offenheit,<br>Mut |
| Prinzipien: | Das Ganze betrachten,<br>Reversible Änderungen während der Entwicklung,<br>Adaption |
| Praktiken: | Aktive Einbindung der echten Anwender,<br>Product Backlog,<br>Sprint Review |

Hier stoßen wir in unserem Umfeld oft auf sehr große Widerstände und Konflikte. Das V-Modell unterstützt die Tendenz, alle Anforderungen am Anfang eines Pro-

jektes festzulegen. Auch die inhaltlich sequentielle Struktur der IEC 62304 führt in vielen Unternehmen zu ebenso sequentiellen Entwicklungsprozessen, die in der Organisation zusätzlich mit Stage-Gate-Managementprozessen erweitert werden. In solch einem Umfeld ist es sehr aufwändig, alle Dokumente aufzusetzen und fertig zu stellen, die für eine Freigabe in einem Gate benötigt werden. Dies führt bei vielen Beteiligten unweigerlich dazu, dass Anforderungsänderungen so weit wie möglich vermieden werden. Jede Änderung würde zu entsprechenden Anpassungen in den Dokumenten führen und eine weitere Änderungsfreigabe herbeiführen.
Wir benötigen für Anforderungsänderungen also die Möglichkeit einer einfachen, risikofreien Handhabung. Das in diesem Buch beschriebene Vorgehensmodell löst unter anderem genau diese Problematik auf.

**"auch spät während der Entwicklung**" – es gibt keinen sinnvollen Zeitpunkt, um eine Menge von Anforderungen zu fixieren, selbst wenn das Projektende wenige Wochen in der Zukunft liegt. Unser Projektumfeld kann sich jederzeit ändern und dadurch die Notwendigkeit bestehen, die bislang für gültig betrachteten Anforderungen zu überarbeiten.

Werte: Offenheit,
Mut

Prinzipien: So spät wie möglich entscheiden

Praktiken: Aktive Einbindung der echten Anwender,
Product Backlog,
Sprint Review

Durch die phasenorientierten Vorgehensmodelle, die sich in der Medizintechnik bislang etabliert hatten, stehen späte Anforderungsänderungen immer vor großen Hürden. Dies kann so weit gehen, dass es ab einem bestimmten Zeitpunkt gar keine Änderungen mehr geben darf. In der Praxis bedeutet dies entweder, dass wir (a) trotz besseren Wissens darauf verzichten, das richtige Produkt zu bauen, (b) den definierten Änderungsmanagementprozess bemühen und damit zum Teil hohe Aufwände erzeugen oder (c) den definierten Prozess geschickt umgehen und damit umgesetzte Realitäten erschaffen, die nicht mehr den fixierten Anforderungen entsprechen. In allen drei Fällen handeln wir uns Probleme ein: unzufriedene Anwender, hohe Kosten und lange Dauer, sowie inkonsistente Nachvollziehbarkeit der Produktentwicklung.
Wir benötigen für Anforderungsänderungen also die Möglichkeit, diese mit wenig Aufwand jederzeit in die Produktentwicklung einfließen zu lassen. Das in diesem Buch beschriebene Vorgehensmodell setzt genau diesen Anspruch um.

**"willkommen"** – erzeugt ein positives und offenes Gefühl im Sinne einer Einladung. Es entsteht keine ablehnende Haltung gegenüber Veränderungen.

Werte: Offenheit,
Respekt

Prinzipien: Adaption

Praktiken: Sprint Planung

Grundsätzlich besteht die Notwendigkeit, eher veränderungsscheue Kulturen in Teams und Organisationen dahingehend zu öffnen, dass Veränderungen nicht mehr als Bedrohung wahrgenommen werden. Es gibt jedoch keinen besonderen Bezug zur Medizintechnik.

**"Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden."** – Wie bereits im ersten Prinzip (Den Kunden zufrieden stellen) ist es unser Ziel, dem Kunden/Anwender zu helfen und ihn zu betreuen. Der Kunde steht immer im Mittelpunkt unseres Handelns.

Werte: Offenheit,
Commitment,
Fokus

Prinzipien: Ökonomie

Praktiken: Regelmäßige Auslieferung,
Sprint Review,
Product Backlog,
Aktive Einbindung der echten Anwender

Es existiert kein besonderer Bezug zur Medizintechnik, da es branchenunabhängig das Ziel jeder Unternehmung ist, Wettbewerbsvorteile zu erkennen und zu nutzen.

Zusammengefasst müssen wir offen sein für jedwede Änderungen und benötigen den Mut, dem Kunden dabei zu helfen, sich dieser Veränderungen anzupassen.

## Agiles Prinzip 3: Regelmäßige Lieferung funktionierender Software

Das dritte Prinzip hinter dem agilen Manifest lautet:

> *"Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne."*

**"Liefere"** – wir benötigen ein fertiges Produkt, das für einen Anwender echten Mehrwert erzeugt und welches wir an diesen Anwender ausliefern können.

| | |
|---|---|
| Werte: | Mut,<br>Commitment |
| Prinzipien: | Schnellstmöglich liefern |
| Praktiken: | Inkrementelles Deployment,<br>Kleine Releases,<br>Sprint Review,<br>Produkt Demonstration |

In unserem medizintechnischen Umfeld stehen wir vor größeren Herausforderungen, je nach Sicherheitsklassifizierung unseres Produktes bzw. der darin enthaltenen Software. Ein sehr hoher Aufwand muss oftmals betrieben werden, um ein tatsächlich „fertiges" Produkt in einen auslieferungsfähigen Zustand zu bringen. Dieser Aufwand ist mit Sicherheit zu hoch, als dass wir ihn alle zwei Wochen durchführen könnten, um dem Anwender eine neue Version zur Verfü-

gung zu stellen.
Wir benötigen für diese Fälle also entweder eine andere Möglichkeit der Auslieferung oder eine andere Zielgruppe. Wenn wir nicht an die echten Anwender und Patienten ausliefern können, müssen wir mit entsprechenden Vertretern dieser Anwender zusammenarbeiten. Hier bieten sich verschiedene Möglichkeiten, angefangenen bei Produkt-Demonstrationen in Sprint Review Meetings mit diesen echten Anwendern, über Feedbackschleifen in klinischen Validierungen, bis hin zur vollständigen Vertretung der Anwender durch einen Product Owner.

**"funktionierende Software"** – das Ergebnis unserer Anstrengungen darf kein theoretisches Konzept oder ein Prototyp sein, sondern ein funktionierendes Software-Produkt, mit dem der Anwender effektiv arbeiten kann.

Werte: Fokus,
Commitment

Prinzipien: Qualität

Praktiken: Definition of Done,
Akzeptanz-Tests,
Sprint Review,
Produkt Demonstration

Ein besonderer Bezug zur Medizintechnik ist hier nicht gegeben. Es ist grundsätzlich wichtig, ein funktionierendes Produkt in hoher Qualität zu entwickeln.

**"regelmäßig"** – wir müssen dem Anwender ein benutzbares Produkt weit öfter geben als nur einmal am Ende des Entwicklungsprojekts.

Wert: Fokus,
Mut

Prinzipien: Regelmäßige Auslieferung

Praktiken: Inkrementelles Deployment,
Kleine Releases

Auch hier liegt die Herausforderung darin, alle notwendigen Maßnahmen vor jeder Lieferung durchzuführen.

**"innerhalb weniger Wochen oder Monate"** – die Iterationen sollen kurz getaktet sein, damit sie einfach zu handhaben und klar arrangiert sind

Wert: Fokus

Prinzipien: Schnellstmöglich liefern,
Iterative Vorgehensweise

Praktiken: Sprint

Die Schwierigkeit liegt hier in der Abwägung von Aufwand und Nutzen einer kurzen Iteration für die regelmäßige Auslieferung eines funktionierenden Produktes. Hier spielen auch Aspekte wie z.B. fachliche und technologische Komplexität eine Rolle: kennen wir den Markt, den Anwender und seine Belange? Beherrschen wir die eingesetzte Technologie? Dann

können wir eventuell eine längere Iteration für die Auslieferung wählen. Oder gibt es hohe Unsicherheiten, was die Anforderungen angeht? Dann wollen wir mit Hilfe von kurzen Iterationen schnell mit jeder Auslieferung lernen, was der Anwender überhaupt von unserem Produkt benötigt.

**"bevorzuge dabei die kürzere Zeitspanne"** – Ergebnisse müssen früh und oft erzeugt werden, damit der Anwender/Kunde darauf schnell reagieren und entscheiden kann.

Werte: Fokus,
Einfachheit,
Feedback

Prinzipien: Iterative Vorgehensweise,
Limitierung der laufenden Arbeiten,
Schnellstmöglich liefern,
Verschwendung eliminieren

Praktiken: Inkrementelles Deployment,
Kleine Releases,
Sprint Backlog,
Sprint

Wie bereits erwähnt, müssen wir, was die Länge der Iteration für die Auslieferung angeht, genau Aufwand und Nutzen abwägen.

## Agiles Prinzip 4: Bereichsübergreifende Zusammenarbeit

Das vierte Prinzip hinter dem agilen Manifest lautet:

> *"Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten."*

**"Fachexperten"** – es gibt Leute mit einem enorm umfangreichen Wissen über die Zielmärke eines Produktes, dessen Anwender und Kunden, sowie der benötigten Funktionalitäten. Diese Personen sollten nicht unbedingt die Entwickler des Produktes sein, da diese üblicherweise dazu tendieren, technikgetriebene, schicke kleine Dinge zu implementieren. "Fachexperten" sollten diejenigen sein, welche die Produktanforderungen mit dem größten Kundennutzen (oder Geschäftswert) pflegen und priorisieren können.

Werte: Respekt,
Kommunikation

Prinzipien: Ökonomie

Praktiken: Aktive Einbindung der echten Anwender,
Ganzes Team

Im medizintechnischen Umfeld gibt es sehr viele Fachexperten, die ihren notwendigen Beitrag zu einer effektiven Medizinproduktentwicklung leisten müssen. Angefangen bei unternehmensinternen Personen aus dem Produkt-Management, Vertrieb oder Außendienst bis hin zu externen Personen wie beispielswei-

se Ärzte, Medizinisch-technische Assistenten (MTAs), Wissenschaftler und Forscher des medizinischen Bereiches unseres Produktes, sowie als wichtigste Personen die Patienten, die letztlich als Anwender unseres Produktes den größtmöglichen Nutzen erfahren sollen.
Je mehr all dieser Personen wir in unsere Produktentwicklung einbinden können, desto größer wird der Nutzen unseres Produktes am Ende sein.

**"Entwickler"** – im Sinne von "Produktentwickler" bezieht sich das nicht nur auf programmierende Software-Entwickler. Ebenso beteiligt sind Tester, technische Redakteure, Interface Designer, Business Analysten und alle anderen Rollen, die dazu beitragen können, ein funktionierendes Software-Produkt zu erstellen.

Werte: Respekt,
Kommunikation

Prinzipien: Vielfalt

Praktiken: Ganzes Team

Es existiert kein besonderer Bezug zur Medizintechnik. Die Einbindung aller Beteiligter im Produktentstehungsprozess ist eine allgemeine Problematik, die branchenunabhängig gelöst werden muss.

**"müssen zusammenarbeiten"** – es ist im Regelfall nicht hilfreich, Experten für sich alleine arbeiten zu lassen und ihre Ergebnisse in einer Prozesskette dem nächsten Experten zu zu werfen. Ihre Leistung sollte viel-

mehr das Ergebnis enger Zusammenarbeit und Kommunikation sein.

Werte: Kommunikation

Prinzipien: Vielfalt

Praktiken: Ganzes Team

Die Schwierigkeit in der Medizintechnik ist oftmals, dass es für sehr spezialisierte Detailthemen nur einen einzigen Experten gibt. Auf diesen einen Wissensträger sind andere Teammitglieder immer wieder angewiesen, manchmal sogar ganze Teams und Abteilungen. Mit agilen Methoden können solche Abhängigkeiten leicht transparent gemacht werden, so dass wir in der Lage sind, sie entweder aufzulösen oder sie zumindest leichter zu koordinieren.

**"täglich"** – Zusammenarbeit und Kommunikation funktionieren nur auf einer kontinuierlichen Basis gut. Ein Product Owner muss für das Team während des ganzen Sprints verfügbar und präsent sein, nicht nur zu den Planungs- und Review-Meetings.

Werte: Fokus,
Commitment

Prinzipien: Flow,
Iterative Vorgehensweise,
Osmotische Kommunikation

Praktiken: Zusammen sitzen,
Daily Stand-up,
Ganzes Team

Es existiert kein besonderer Bezug zur Medizintechnik. Die Zusammenarbeit auf täglicher Basis ist grundsätzlich von Vorteil.

**"während des Projektes"** – Zusammenarbeit und Kommunikation dürfen nicht nur während spezifischer Phasen eines Projektes stattfinden. Sie müssen durchgängig von Anfang bis zum Ende eines Projektes gelebt werden.

Werte: Commitment,
Fokus

Prinzipien: Regelmäßige Auslieferung,
Flow,
Iterative Vorgehensweise,

Praktiken: Team-Kontinuität,
Leidenschaftliche Arbeit

Die besondere Herausforderung in der Medizintechnik ist auch hier wieder die Überwindung des klassischen Phasendenkens, wie es durch das V-Modell und andere Vorgehensmodelle gefördert wird. Mitarbeiter, Abteilung und ganze Bereiche, die bislang nur in bestimmten Projektphasen und teilweise nur einzeln beteiligt waren, sind jetzt aufgefordert, übergreifend miteinander zusammen zu arbeiten und zu kommunizieren.

## Agiles Prinzip 5: Unterstützung und Vertrauen

Das fünfte Prinzip unter dem agilen Manifest lautet:

> *"Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen, und vertraue darauf, dass sie die Aufgabe erledigen."*

**"Errichte Projekte rund um motivierte Individuen"** – suchen Sie Persönlichkeiten, die den entsprechenden Antrieb mitbringen. Stellen Sie nur Leute ein, die Ihren Job lieben und Ihrer Arbeit mit großer Leidenschaft nachgehen. Trennen Sie sich von unfähigen, unwilligen oder gar zerstörerischen Mitarbeitern, da diese einen sehr negativen Einfluss auf den Projekterfolg haben werden.

| | |
|---|---|
| Werte: | Commitment,<br>Mut,<br>Respekt |
| Prinzipien: | Lernen verstärken,<br>Mitmenschlichkeit,<br>Akzeptierte Verantwortung |
| Praktiken: | Leidenschaftliche Arbeit |

Es gibt keinen besonderen Zusammenhang mit der Medizintechnik. Der große Vorteil unserer Branche ist es jedoch, dass sehr viele Mitarbeiter die Medizintechnik zu ihrem Arbeitsgebiet gewählt haben, eben

weil sich dort sinnvollende und erfüllende Produkte realisieren lassen.

**"Gib ihnen"** – bereits in der Bibel steht geschrieben "Gebt, und es wird euch gegeben werden". Verlangen Sie nicht nach Ergebnissen von den Teams, sondern geben Sie ihnen, was sie für ihr privates und berufliches Leben benötigen.

Werte: Respekt

Prinzipien: Bevollmächtigtes Team,
Mitmenschlichkeit,
Unterstützende Kultur

Praktiken: Dienende Führung

Auch hier existiert kein besonderer Zusammenhang mit der Medizintechnik. Auch unabhängig vom gewählten Vorgehensmodell ist es langfristig hilfreicher, eine dienende Führungskultur zu etablieren.

**"das Umfeld"** – das Team benötigt ein angemessenes Arbeitsumfeld, einen geeigneten Arbeitsplatz und das beste Equipment, welches verfügbar ist, um seine Arbeit auf eine professionelle Art und Weise durchzuführen. Dabei handelt es sich auch um die Prozesse des Teams aus dem Blickwinkel der gesamten Organisation. Umgeben Sie das Team mit schlanken Strukturen und versuchen Sie, bürokratische Strukturen zu vermeiden, welche die Performance und Geschwindigkeit des Teams behindern.

Werte: Einfachheit,
Offenheit

Prinzipien: Bevollmächtigtes Team,
Unterstützende Kultur

Praktiken: Ganzes Team,
Team-Raum

Es besonderer Bezug zur Medizintechnik ist hier nicht gegeben. Es ist grundsätzlich sinnvoll, einem Team das bestmögliche Arbeitsumfeld bereit zu stellen.

**"und die Unterstützung, die sie benötigen"** – helfen Sie dem Team, sobald es nicht weiterkommt oder irgendwo stecken bleibt. Machen Sie den Weg für das Team frei und entfernen Sie jedwede Behinderung. Zeigen Sie dem Team alternative Wege, die es beschreiten kann.

Werte: Mut,
Fokus

Prinzipien: Verbesserung,
Adaption,
Unterstützende Kultur

Praktiken: Dienende Führung

Wie bereits erwähnt, ist die Etablierung einer dienenden Führungskultur von der Medizintechnikbranche unabhängig.

**"vertraue darauf, dass sie die Aufgabe erledigen"** – wenn Sie motivierte Individuen im Team haben, dann müssen sie diesen auch vertrauen. Es sind die Leute, welche genau wissen, wie sie ihre Arbeit erledigen müssen. Lassen Sie niemanden außerhalb des Teams entscheiden und planen, wie das Team seine Arbeit erledigen soll und wer an was und wann arbeitet.

| | |
|---|---|
| Werte: | Respekt,<br>Mut |
| Prinzipien: | Selbstorganisiertes Team,<br>Bevollmächtigtes Team,<br>Mitmenschlichkeit,<br>Vielfalt,<br>Unterstützende Kultur |
| Praktiken: | Ganzes Team,<br>Team-Raum,<br>Team-Kontinuität,<br>Leidenschaftliche Arbeit |

Der Bezug zur Medizintechnik liegt hier in der Frage, wie die Nachvollziehbarkeit des Entwicklungsprozesses und der Ergebnisse gewährleistet werden können, wenn ein Team selbstverantwortlich arbeitet und darauf vertraut wird, dass das Team schon das Richtige erledigt. Bei der Beantwortung dieser Frage helfen uns einfache Praktiken wie Sprint Notizen und das Sprint Review. Unser Ziel ist nicht in erster Linie, formal korrekte Nachweise über die erledigten Aufgaben anzufertigen. Diese sind vielmehr ein notwendiges Vehikel im regulierten Umfeld. Unser eigentliches, weitaus wichtigeres Ziel ist die Fähigkeit, in regelmäßigen, kurzen Abständen auf Basis dieser erledigten

Aufgaben darüber zu entscheiden, ob unser Produkt einen finalen Stand erreicht hat oder in welche Richtung wir die weitere Produktentwicklung verändern müssen.

## Agiles Prinzip 6: Konversation von Angesicht zu Angesicht

Das sechste Prinzip unter dem agilen Manifest lautet:

> *"Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht."*

**"die effizienteste und effektivste Methode, Informationen zu übermitteln"** – es ist nicht ausreichend, Informationen passiv verfügbar zu machen. Vielmehr sollten sie der Zielgruppe aktiv vermittelt werden.

Werte: Kommunikation,
Offenheit

Prinzipien: Transparenz,
Arbeit sichtbar machen,
osmotische Kommunikation,
Sagen statt Fragen

Praktiken: Informationsreicher Arbeitsplatz

In der Medizintechnik haben wir die Notwendigkeit zu erhöhter Dokumentation im Vergleich zu vielen anderen Branchen. Es ist eine Gratwanderung, hier das richtige Maß an Informationsverteilung zu finden. Zu viel kann dazu führen, dass die wichtigen Informationen in der Flut aller Mitteilungen unter gehen. Zu wenig kann dazu führen, dass wesentliche Informationen nicht gelesen werden.

**“an und innerhalb eines Entwicklungsseams”** – erwarten Sie nicht nur aktive Kommunikation von außen an ihr Team, sondern fördern Sie aktive Kommunikation mit Ihren eigenen Teammitgliedern.

Werte: Kommunikation,
Offenheit

Prinzipien: Transparenz,
Arbeit sichtbar machen,
Sagen statt Fragen

Praktiken: osmotische Kommunikation,
Informationsreicher Arbeitsplatz,
Dienende Führung

Ein besonderer Bezug zur Medizintechnik ist hier nicht gegeben.

**“ist im Gespräch von Angesicht zu Angesicht”** – öffnen Sie keine ineffizienten Kommunikationskanäle wie zum Beispiel Dokumentation auf Papier, E-Mails und Bug-Tracking Systeme. sprechen Sie mit dem Empfänger Ihrer Nachricht lieber direkt per Telefon, Video oder im besten Fall vor einem Whiteboard von Person zu Person.

Werte: Fokus,
Feedback

Prinzipien: Sagen statt Fragen

Praktiken: Zusammen sitzen,
Team-Raum

Auch hier haben wir in der Medizintechnik oft wenig Wahl darüber, was in welcher Form dokumentiert werden muss. Die Kunst dabei ist jedoch, nicht diese Dokumente als Grundlage für Kommunikation zu verwenden, sondern das echte Gespräch und die Konversation zwischen beteiligten Personen zu fördern.

## Agiles Prinzip 7: Funktionierende Software

Das siebte Prinzip unter dem Agilen Manifest lautet:

> *"Funktionierende Software ist das wichtigste Fortschrittsmaß."*

**"Funktionierende Software"** – es ist unerheblich, wie viele Module, Schnittstellen und Dokumente wir in Arbeit haben oder wie viele davon existieren. Das einzige, was zählt, sind fertiggestellte, voll funktionstüchtige Teile des benötigten Softwareprodukts.

Werte: Commitment,
Fokus

Prinzipien: Qualität

Praktiken: Definition of Done,
Akzeptanz-Tests,
aktive Einbindung der echten Anwender,
Testen durch den Anwender,
Produkt Demonstration

In der Medizintechnik haben wir die besondere Herausforderung, dass ein „funktionierendes" Produkt nicht nur aus einer Ansammlung integrierter Funktionalitäten besteht, sondern viele weitere Aspekte wie z.B. Usability (IEC 62366), Risikomanagement (ISO 14971), Dokumentation, Verifikation und Validierung. Die Umsetzung und Einhaltung dieser Aspekte führt dazu, dass wir nicht einfach Funktionen zählen kön-

nen. Auch ein Mindestmaß an nicht-funktionalen Anforderungen gehört für uns immer zum Pflichtprogramm bei der Entwicklung medizintechnischer Produkte.

**"ist das wichtigste Fortschrittsmaß"** – niemand interessiert sich für die Anzahl der Arbeitsstunden, die Anzahl der Programmzeilen oder eine andere technische Metrik. Nur fertige und auslieferbare Features sind wichtig.

| | |
|---|---|
| Werte: | Einfachheit,<br>Commitment,<br>Feedback |
| Prinzipien: | Business Value,<br>Regelmäßige Auslieferung |
| Praktiken: | Inkrementelles Deployment,<br>Kleine Releases,<br>Release Burndown,<br>Produkt Demonstration |

Es existiert kein besonderer Bezug zur Medizintechnik. Wie in vielen anderen Branchen ist die Herausforderung, die klassischen Messmethoden und KPIs[4] auf ihren Nutzen hin zu hinterfragen und moderne, motivierende Kennzahlen zu erfassen.
Ein typisches Beispiel ist die Bestimmung der Auslastung aller Projektmitarbeiter mit dem Ziel, eine 100%-Auslastung dieser Personen zu erreichen, da dies zumindest in der weit verbreiteten Meinung der maxi-

[4] KPI = Key Performance Indicator

malen Effizienz eines Teams entsprechen muss. Diese Art von „Management by Measurement" führt leider oftmals zu dem sonderbaren Verhalten, dass Mitarbeiter versuchen, ihre physische Anwesenheit zu maximieren, nicht jedoch das Ergebnis ihrer Entwicklungs- oder Denkleistung. Es gibt viele andere Beispiele für Kennzahlen, die letztendlich zu einem ungewünschten, künstlichen Verhalten der Mitarbeiter führen.

## Agiles Prinzip 8: Gleichmäßiges Tempo

Das achte Prinzip unter dem Agilen Manifest lautet:

> *"Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können."*

**"Agile Prozesse fördern nachhaltige Entwicklung."** – Dies ist kein Prinzip, sondern eine Annahme, welche durch die Anwendung agiler Praktiken bestätigt werden muss. Ich bin der Meinung, dieser Satz sollte aus dem achten Prinzip entfernt oder zumindest umformuliert werden.

**"Die Auftraggeber, Entwickler und Benutzer"** – Es handelt sich um einen bereichsübergreifenden Ansatz, welcher das gesamte Team umfasst sowie alle Personen, die in der Produktentstehungskette involviert sind.

Werte: Kommunikation

Prinzipien: Vielfalt

Praktiken: Aktive Einbindung der echten Anwender, Ganzes Team

Der große Vorteil in der Medizintechnik ist, dass wir sowieso mit den Anwendern beim Thema Usability

zusammenarbeiten müssen, genauso wie mit den Auftraggebern in Form von Management, Produktverantwortlichen und Kunden. Die Einbindung all dieser Personengruppen muss nun effektiv gestaltet werden.

**"sollten ein gleichmäßiges Tempo [...] halten können"** – Die an der Produktentstehungskette beteiligten Personen dürfen weder an Überlastung leiden, noch dürfen sie wartend rumsitzen. Eine wichtige Praktik insbesondere für das Team ist der Kollektivbesitz sowohl des Codes, als auch der Verantwortung. Ich nenne das "verteiltes Commitment".

| | |
|---|---|
| Werte: | Commitment,<br>Fokus,<br>Mut |
| Prinzipien: | Regelmäßige Auslieferung,<br>Flow |
| Praktiken: | Leidenschaftliche Arbeit,<br>Kollektivbesitz |

Die Besondersheit in der Medizintechnik liegt darin, dass abhängig von der Sicherheitsklassifizierung eines Produktes der vollständige Kollektivbesitz in einem Team untersagt ist. Ein Entwickler darf in manchen Fällen seine eigenen Ergebnisse gar nicht selbst verifizieren. Hier muss in Abhängigkeit der vorhandenen Skills und des Projektumfeldes abgewogen werden, welche Form von Kollektivbesitz angewendet werden kann.

**“auf unbegrenzte Zeit”** – Es reicht nicht, eine konstante Geschwindigkeit am Anfang eines Projektes oder nur für ein paar Iterationen zu halten. Gerade am Ende eines Projektes sollen alle beteiligten Personen Ruhe bewahren und in einem “Flow” die Arbeit zu Ende bringen.

Werte: Commitment

Prinzipien: Mitmenschlichkeit,
Flow,
Limitierung der laufenden Arbeiten

Praktiken: Leidenschaftliche Arbeit

Ein besonderer Zusammenhang mit der Medizintechnik ist hier nicht gegeben.

## Agiles Prinzip 9: Technische Exzellenz

Das neunte Prinzip unter dem Agilen Manifest lautet:

> *"Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität."*

**"Ständiges Augenmerk"** – Hüten Sie sich vor Routine im Arbeitsalltag! Halten Sie Ihre Konzentration oben und bleiben Sie fokussiert an den Dingen, an denen Sie gerade arbeiten. Dies bedeutet auch, eine Pause zu machen, wenn Sie nicht dazu in der Lage sind. Zwingen Sie sich nicht, weiter zu arbeiten, wenn Sie abgelenkt und kraftlos sind.

Werte: Fokus

Prinzipien: Akzeptierte Verantwortung, Reflektieren

Praktiken: Sprint Retrospektive

Es gibt hier keinen besonderen Bezug zur Medizintechnik.

**"technische Exzellenz"** – Versammeln Sie die richtigen Leute mit den richtigen Fähigkeiten, um ein Produkt zu erzeugen. Kultivieren Sie ein Umfeld lebenslangen Lernens, um sich neue Kenntnisse anzueignen und die eigenen Fähigkeiten zu erweitern.

Werte: Mut,
Einfachheit,
Fokus

Prinzipien: Verbesserung,
Lernen verstärken,
Akzeptierte Verantwortung,
Qualität

Praktiken: Code Retreat / Coding Dojo,
(agiles) Testen,
Sprint Retrospektive

Ein besonderer Zusammenhang mit der Medizintechnik ist hier nicht gegeben.

**"gutes Design"** – Erzeugen Sie kein Produkt, das außen zwar schön aussieht, im inneren jedoch - technisch betrachtet - „hässlich" umgesetzt ist. Packen Sie sich selbst an Ihrer Berufsehre und bauen Sie ein Software-Produkt auf eine Art und Weise, dass Sie auf jedes Detail stolz sein können.

Werte: Einfachheit

Prinzipien: Verbesserung,
Qualität,
Inkrementelle Entwicklung

Praktiken: Pair Programming,
testgetriebene Entwicklung,
Refaktorisierung

Die besondere Herausforderung in der Medizintechnik liegt darin, die notwendige Dokumentation bei regelmäßigen Veränderungen des Software-Designs schnell, einfach und ohne bürokratischen Aufwand anpassen zu können.

**"fördert Agilität"** – Wenn wir nicht die richtigen Kenntnisse, die richtigen Fähigkeiten und die richtige Attitüde besitzen, werden wir auf der agilen Reise irgendwo mittendrin stecken bleiben. Dies ist der Grund, weswegen manche Teams nach ihrem anfänglichen agilen Erfolg nicht weiterkommen.

Werte: Offenheit,
Mut

Prinzipien: Reflektieren,
Verbesserung

Praktiken: Sprint Retrospektive

Es gibt keinen besonderen Zusammenhang mit der Medizintechnik.

## Agiles Prinzip 10: Einfachheit

Das zehnte Prinzip unter dem Agilen Manifest lautet:

> *"Einfachheit — die Kunst, die Menge nicht getaner Arbeit zu maximieren — ist essenziell."*

**"Einfachheit"** – Wie in dem Prinzip bereits selbst erwähnt, handelt es sich um "die Kunst, die Menge nicht getaner Arbeit zu maximieren". Denken Sie stets über unnötige Aufgaben in Ihrer Todo-Liste nach. Denken Sie an Anforderungen, welche niemand wirklich umgesetzt haben muss, um seine Arbeit zu erledigen. Denken Sie an Methoden, Klassen und Ihr Software-Design im Allgemeinen – Sie werden immer verschiedenste Dinge finden, die einfach weggelassen werden können. Viele Aufgaben und Anforderungen tragen überhaupt nichts dazu bei, ein Ziel zu erreichen.

| | |
|---|---|
| Werte: | Fokus,<br>Einfachheit |
| Prinzipien: | Limitierung der laufenden Arbeiten,<br>So spät wie möglich entscheiden,<br>Selbstähnlichkeit,<br>Verschwendung eliminieren |
| Praktiken: | Product Backlog,<br>Refaktorisierung,<br>Verschwendung erkennen,<br>Einfaches Design |

Es existiert kein besonderer Bezug zur Medizintechnik.

**"ist essenziell"** – Das Essenzielle ist das, was tatsächlich Wert erzeugt. Alle anderen Dinge sind "nice to have", "Gold Plating" (vergolden) oder rein technischer Schnickschnack, der einfach nur umgesetzt wird, weil wir es können, nicht jedoch, weil er benötigt wird. Wir müssen das Wesentliche fokussieren, um unsere zugesagten Ziele zu erreichen.

Werte: Fokus,
Mut

Prinzipien: Verschwendung eliminieren,
So spät wie möglich entscheiden,
Ökonomie,
Business Value

Praktiken: Verschwendung erkennen

Auch hier gibt es keinen besonderen Zusammenhang mit der Medizintechnik. Es ist branchenübergreifend eine der größten Schwierigkeiten, die Entscheidung zu treffen, was in einem Projekt bzw. einer Produktentwicklung *nicht* gemacht werden soll.

## Agiles Prinzip 11: Selbstorganisation

Das elfte Prinzip unter dem Agilen Manifest lautet:

> *"Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams."*

**"Die besten Architekturen, Anforderungen und Entwürfe"** – Es geht darum, Produktideen zu entdecken, deren technische und inhaltliche Herausforderungen zu erkunden und entsprechende Lösungen zu finden. Diese Artefakte dürfen nicht einzeln und unabhängig voneinander erstellt werden. Sie gehören zusammen und bilden als Ganzes das resultierende Produkt.

Werte: Einfachheit, Fokus

Prinzipien: Vielfalt

Praktiken: Ganzes Team

In der vom V-Modell geprägten Medizintechnik mag es zunächst ungewöhnlich und fragwürdig erscheinen, Architektur, Anforderungen und Umsetzung gemeinsam zu erzeugen. Gerade diese Artefakte werden in einem klassischen Vorgehensmodell normalerweise strikt voneinander getrennt, da sie jeweils aufeinander aufbauen. Die gemeinsame, parallele Erzeugung dieser Artefakte benötigt einen ganz anderen Prozess. Antworten zu einem solchen, nicht phasen-orientier-

ten Vorgehensmodell finden sich im weiteren Verlauf dieses Buches.

**"entstehen"** – Die benötigten Artefakte können nicht von vornherein geplant werden. Sie werden während des Projektes entdeckt und entstehen auf eine evolutionäre Art und Weise. Denken Sie nicht zu lange über den richtigen Weg nach, etwas zu tun. Gehen Sie einfach iterativ-inkrementell voran und verbessern Sie diesen Weg durch regelmäßiges Reflektieren.

Werte: Einfachheit,
Feedback,
Mut

Prinzipien: So spät wie möglich entscheiden,
Adaption

Praktiken: Testgetriebene Entwicklung,
Refaktorisierung

Auch dieser Punkt erscheint auf den ersten Blick in der Medizintechnik nicht umsetzbar. In der alten, dokumentenzentrierten Vorgehensweise werden grundlegende Entscheidungen gut durchdacht und schriftlich fixiert. Es „entsteht" nicht einfach irgendeine Lösung. Hier ist die große Herausforderung, sich auf die iterativen Zyklen und Feedbackschleifen einzulassen, immer wieder neue Entscheidungen zu treffen und diese leichtgewichtig zu dokumentieren.

**"durch selbstorganisierte Teams"** – Ein Team benötigt keine einzelne, verantwortliche Person "obendrüber",

um zu entscheiden, wer was wann und wie bearbeitet. Wenn ein Team alle notwendigen Rollen und Fähigkeiten hat, um das benötigte Produkt zu erzeugen, dann ist es in der Lage, selbst zu entscheiden und sich selbst zu organisieren.

Werte: Kommunikation,
Commitment

Prinzipien: Vielfalt,
Akzeptierte Verantwortung,
Bevollmächtigtes Team,
Mitmenschlichkeit,
Osmotische Kommunikation,
Unterstützende Kultur

Praktiken: Ganzes Team,
Dienende Führung

Es existiert kein besonderer Zusammenhang mit der Medizintechnik. Der Wandel von Führungskulturen hin zu eigenverantwortlichen, selbstorganisierten Teams finden branchenübergreifend in vielen Unternehmen statt.

## Agiles Prinzip 12: Inspektion und Adaption

Das zwölfte Prinzip unter dem Agilen Manifest lautet:

> *"In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an."*

**"In regelmäßigen Abständen"** – Es ist nicht ausreichend, das Projekt nach seinem Abschluss zu betrachten, da wir nicht in der Lage sind, unsere Herangehensweise in der Projektentwicklung rückwirkend zu ändern. Auch wenn durch eine sogenannte "Post-Mortem Analyse" schwerwiegende Misstände erkannt und daraus Maßnahmen für das nächste Projekt abgeleitet werden können, ist es uns nicht möglich, das gerade durchgeführte Projekt anders umzusetzen. Aus diesem Grund müssen wir unsere aktuellen Verhaltens- und Herangehensweisen iterativ betrachten.

Werte: Commitment

Prinzipien: Iterative Vorgehensweise

Praktiken: Sprint

Es stellt für die meisten Unternehmen in der Medizintechnik kein großes Problem dar, eine iterative Vorgehensweise einzusetzen. In vielen Fällen werden bereits Reporting-Meetings, Kern-Team-Meetings oder ähnliche Veranstaltungen durchgeführt. Die Einführung

eines festen Taktes ist anfangs zwar ungewohnt, aber ohne Schwierigkeiten machbar.

**"reflektiert das Team, wie es effektiver werden kann"** – Weder der Team-Manager, noch der Scrum Master, noch ein Qualitätsmanager ist dafür Verantwortlich, die Workflows und Prozesse eines Entwicklungsteams zu definieren. Nur das Team selbst ist in der Lage, die eigenen Gewohnheiten zu inspizieren und zu entscheiden, was wie geändert werden soll. Es ist natürlich nicht verboten, einen externen Moderator für Retrospektiven heranzuziehen – dies kann dem Team helfen, sich auf die Inhalte und Erkenntnisse zu konzentrieren anstatt auf die Durchführung eines Meetings.

Werte: Offenheit, Feedback

Prinzipien: Reflektieren, Inspektion

Praktiken: Sprint Retrospektive, Verschwendung erkennen

Es gibt keinen besonderen Zusammenhang mit der Medizintechnik. In vielen Unternehmen gibt es ein institutionalisiertes Verbesserungsvorschlagswesen, an dem sich alle Mitarbeiter beteiligen können. Die eigentliche Herausforderung kommt im nächsten Punkt.

**"und passt sein Verhalten entsprechend an"** – Die schlimmste Retrospektive ist eine Reflektierung des eigenen Verhaltens ohne jegliche Einsichten und abgeleitete Maßnahmen. Es muss durchführbare Resultate für das Team geben, damit es die Möglichkeit hat, Verantwortung für diese Resultate zu übernehmen.

Werte: Commitment

Prinzipien: Verbesserung,
Verschwendung eliminieren,
Lernen verstärken,
Adaption

Praktiken: Sprint Retrospektive

Dies ist eine der größten Herausforderungen im regulierten Umfeld der Medizintechnik. Wir benötigen ein Qualitätsmanagementsystem, das es ermöglicht, den Entwicklungsprozess eines Teams von diesem Team selbst immer wieder ohne bürokratische Hürden schnell verändern zu lassen.

# Weitere Prinzipien für agile Teams

Die folgende Auflistung weiterer Prinzipien für agile Teams beinhaltet alle referenzierten Prinzipien in den vorgehend im Detail beschriebenen zwölf agilen Prinzipien hinter dem agilen Manifest.

**Adaption** - Die Fähigkeit, auf Grund von äußeren Umständen und inneren Erkenntnissen die eigene Planung und Vorgehensweise anzupassen, ist einer der Kernschlüssel zu erfolgreicher, agiler Methodik.

**Akzeptierte Verantwortung** - Echte Verantwortung im Team entsteht durch motivierte Mitarbeiter, die den Zweck und den Sinn eines Projektes oder Produktes nicht nur kennen, sondern auch dazu beitragen wollen. Verantwortung kann niemandem gegeben werden, sie wird dann eher zur Pflicht, nicht jedoch zu selbstmotivierter Verantwortung.

**Arbeit sichtbar machen** - Die Darstellung des Entwicklungsprozesses und seiner einzelnen Workflows in einem Team hilft dabei, die Arbeitsabläufe zu hinterfragen und zu verbessern. Eine funktionierende Koordination im Team wird durch diese Sichtbarkeit der Arbeit erst ermöglicht.

**Bevollmächtigtes Team** - Die akzeptierte Verantwortung eines Teams steigt mit dessen Bevollmächtigung, eigene Entscheidungen treffen zu dürfen. Diese Bevollmächtigung kann nur von oben gegeben werden.

Aus diesem Grunde ist das Management bei der Einführung agiler Methoden immer stark mit einzubeziehen.

**Business Value** - Der Zuordnung von echtem Wert zu Anforderungen ist einer der wesentlichen Erfolgsfaktoren für die Produktentwicklung, aber auch für die meisten Organisationen eine der schwierigsten Aufgaben überhaupt. Erst durch die Kenntnis der erwarteten Werte in einem Backlog entsteht die Möglichkeit, nachvollziehbar und sinnvoll zu priorisieren und vor allen Dingen zu entscheiden, was nicht umgesetzt werden soll. Auch eine sinnvolle Aussage darüber, was bereits in einer Produktentwicklung umgesetzt wurde, kann erst durch die Kenntnis der Werte getroffen werden.

**Das Ganze betrachten** - Verantwortung darf nicht nur lokal verteilt werden, da wir sonst ein System der lokalen Optimierung erschaffen. In solch einem System kämpfen Mitarbeiter, Teams oder sogar ganze Abteilungen und Bereiche gegeneinander, da jeder für sich versucht, das Optimale Ergebnis zu erreichen. Viele lokale Optima erreichen in der Summe jedoch nie das Ergebnis, das durch eine globale Betrachtung hätte erreicht werden können. Aus diesem Grund muss Verantwortung immer soweit wie möglich gefasst und gemeinschaftlich erreicht werden.

**Flow** - Für die kontinuierliche Auslieferung an Anwender wird eine Vorgehensweise benötigt, die das Team darin unterstützt, in einen stetigen Fluss mit der Produktentwicklung zu kommen. Auch eine gleichmäßige Auslastung der Mitarbeiter kann durch einen stetigen Fluss erreicht werden.

**Inkrementelle Entwicklung** - Um frühes Feedback und empirische Produktsteuerung zu ermöglichen, muss regelmäßig auf das funktionierende Produkt zugegriffen und dieses bewertet werden können. Dies ist nicht möglich, wenn das Produkt erst am Ende der gesamten Entwicklung verfügbar ist. Aus diesem Grunde wird inkrementelle Entwicklung benötigt, bei der funktionierende Zwischenprodukte immer wieder gezeigt werden können.

**inspektion** - Die bewusste Einsichtnahme in den bestehenden Prozess ist einer der wichtigsten Pfeiler agiler Vorgehensweisen. Nur wenn die Schwierigkeiten, Fallstricke und Engpässe eines Prozesses erkannt werden, ist eine Veränderung des Verhaltens und damit eine zukünftige Verbesserung einleitbar.

**Iterative Vorgehensweise** - Eine kontinuierliche Auslieferung wird durch eine iterative Vorgehensweise unterstützt. Iterationen helfen durch ihre zyklische Natur dabei, kontinuierliche Aktivitäten durch Wiederholung zu verinnerlichen und sie so zum Normalzustand werden zu lassen.

**Lernen verstärken** - Nur durch kontinuierliche Weiterentwicklung und Weiterbildung der Teammitglieder kann nachhaltige Innovationskraft aufrecht erhalten werden. Das Lernen betrifft aber nicht nur die Aneignung neuer technischer Fähigkeiten, sondern gerade auch das Bewerten der bestehenden Prozesse und der funktionalen Inhalte des zu entwickelnden Produktes. Empirische Kontrolle auf diesen Ebenen wird nur durch solch einen konstanten Lernwillen ermöglicht.

**Limitierung der laufenden Arbeiten** - Die Beschränkung der aktuellen Arbeit auf die gerade wichtigste und wertschöpfendste Tätigkeit beugt zum einen schlechtem Multitasking vor und schützt zum anderen vor Verzettelung. Es sorgt dafür, dass das Team an der höchsten Priorität arbeitet und diese so schnell wie möglich fertig stellt.

**Mitmenschlichkeit** - Mitarbeiter dürfen nicht nur als funktionierende Resourcen betrachtet werden. Moderne Führung bedeutet auch, den Mitarbeiter und Kollegen als Menschen wahrzunehmen und auf seine diesbezüglichen Bedürfnisse einzugehen. Er darf auch weder überlastet werden, noch nutzlos und gelangweilt seine Zeit absitzen.

**Ökonomie** - Die wirtschaftliche Betrachtung aller einzelnen, kleinen Anforderungen führt durch die Abwägung von Business Value, Aufwand und Risiko zur Vergabe von bewerteten Prioritäten. Dies macht für alle Beteiligten den Wert einer Anforderung transparent. Um alle wirtschaftlichen Aspekte abzuwägen, wird die Zusammenarbeit und der Input von Anwendern, Kunden, Product Ownern, Management und Entwicklern benötigt.

**Osmotische Kommunikation** - Das Modell der osmotischen Kommunikation geht davon aus, dass der kommunikative Austausch zwischen Personen am besten direkt stattfinden kann und jedes Hindernis zwischen den Beteiligten überwunden werden muss, ähnlich wie eine Membran beim osmotischen Austausch von Stoffen ein Hindernis darstellt. Je länger der Weg zwischen zwei Personen und je mehr Hin-

dernisse sich zwischen diesen befinden, desto schwieriger wird der kommunikative Austausch.

**Qualität** - Was Menschen immer begeistert, ist hohe Qualität. Was dieser Begriff tatsächlich bedeutet und wie er sich in einem Produkt oder einem Feature ausprägt, muss jedes Team in jedem Umfeld neu bewerten und festlegen. Qualität muss in einem angemessenen Maß definiert sein. In jedem Fall muss zwischen äußerer (vom Anwender wahrnehmbar) und innerer (vom Entwickler wahrnehmbar) Qualität unterschieden werden.

**Reflektieren** - Eines der wesentlichen Grundprinzipien aller agilen Methoden ist das kritische Hinterfragen des Status Quo mit dem Ziel, aus den Erfahrungen Rückschlüsse für den weiteren Weg zu ziehen. Jeder Mitarbeiter sollte dieses Prinzip jederzeit für sich anwenden, um Schwierigkeiten, Probleme, aber auch Vorteile schnell zu erkennen. Auch Teams und ganze Organisationen sollten in regelmäßigen Abständen reflektieren und ihr zukünftiges Verhalten entsprechend anpassen.

**Regelmäßige Auslieferung** - Empirische Kontrolle gelingt nur durch regelmäßiges Inspizieren der erreichen Ergebnisse. Aus diesem Grunde ist es notwendig, in jeder Iteration das Feedback zum aktuellen Produktstand einzuholen. Die regelmäßige Auslieferung von Produktinkrementen hilft auch dabei, ein gleichmäßiges Tempo in der Entwicklung aufrecht zu erhalten.

**Reversible Änderungen während der Entwicklung** - Das richtige Produkt kann nur erschaffen werden, wenn die Bereitschaft besteht, bereits erzeugte Zwischenlö-

sungen wegzuwerfen und neu zu entwickeln. An bereits umgesetzten Architekturen und Designs festzuhalten, kann zu unsauberen Lösungen führen.

**Sagen statt Fragen** - Es ist wenig hilfreich, aber leider häufige Praxis, wichtige Informationen an Mitarbeiter und Teams stillschweigend an einer zentralen Stelle zu dokumentieren in der Annahme, dass die Leute, welche die Informationen lesen sollten, diese Stelle regelmäßig nach Neuigkeiten durchsuchen würden. In der Realität sind dann oftmals Sätze zu hören wie: "Das Dokument liegt aber seit langem auf unserem Laufwerk." Viel effektiver ist es, die wirklich wichtigen Informationen gering zu halten und aktiv an die notwendige Leserschaft zu verteilen.

**Schnelles Scheitern** - Das Konzept des "schnellen Scheiterns" geht bei vielen Leuten gegen ihre Intuition, da es impliziert, dass das Scheitern als wertvoll betrachtet wird. Das schnelle Scheitern ist in allen Fällen dem langsamen Scheitern vorzuziehen. Wir möchten nicht in 18 Monaten wissen, dass unser Projekt kein Erfolg ist, sondern regelmäßig nach 2 Wochen. Dann kann adäquat auf die Erfolgsfaktoren Einfluss genommen werden, ohne dass es schon zu spät ist.

**Schnellstmöglich liefern** - Eine schnelle Lieferung von funktionierenden Zwischenergebnissen ermöglicht den gegenseitigen Austausch von schnellem Feedback zum aktuellen Produkt und die Wünsche an das zukünftige Produkt.

**Selbstähnlichkeit** - Einfachheit findet sich in der Natur oftmals durch fraktale Funktionen und Selbstähnlichkeit. Sie lassen sich verblüffend einfach beschreiben, er-

zeugen in ihrer Wirkung aber hochkomplexe, dynamische Systeme. Diese Prinzipien lassen sich in der Software- und Produkt-Entwicklung genau so nutzen, wie in der Definition von Vorgehensmodellen, Prozessen und ganzen Organisationsstrukturen.

**Selbstorganisiertes Team** - In einem Team von qualifizierten Kopfarbeitern werden die besten Entscheidungen vom Team selbst darüber getroffen, wer wann welche Aufgabe wie durchführt. Jede Form von Micromanagement bewirkt den Verlust von Selbstverantwortung und Motivation. Dies ist in der agilen Welt der Grund für solche Aussagen wie "Projektleiter werden nicht mehr benötigt", welche leider zu großen Missverständnissen führen. Gemeint ist nicht der Verzicht auf Führung (Leadership), sondern der Verzicht auf kleinteilige Planung von Aufgaben, bei denen die durchführenden Teammitglieder selbst am besten wissen, wie sie zu organisieren sind.

**So spät wie möglich entscheiden** - Je früher Entscheidungen getroffen werden, desto unsicherer und risikoreicher sind diese Entscheidungen. Bei jeder Entscheidung sollte genau abgewogen werden, wann der spätestmögliche Zeitpunkt gegeben ist, um die Entscheidung endgültig zu treffen.

**Transparenz** - Der wichtigste Aspekt, um ein System, ein Produkt oder ein Prozess beurteilen zu können, ist die volle Transparenz über alle Zustände und Ergebnisse. Transparenz ist die grundlegende Basis für die weitere Inspektion und die abgeleitete Adaption bzw. Verbesserung.

**Unterstützende Kultur** - Ein Umfeld, in dem sich die einzelnen Mitarbeiter gegenseitig helfen und voran bringen, ist der ideale Nährboden für gutes Teamwork. Da Kulturen in erster Linie durch das Verhalten jedes Einzelnen geprägt werden, liegt hier sehr viel Verantwortung bei Führungskräften, das entsprechende Verhalten von Hilfe und Unterstützung an den Tag zu legen.

**Verbesserung** - Der Wille zur kontinuierlichen Verbesserung bringt jeden Einzelnen, aber auch ganze Teams und Organisationen voran. Dazu gehört vor allen Dingen auch die unabdingbare Bereitschaft, bestehende Gegebenheiten zu hinterfragen und sich selbst immer wieder auf den Prüfstand zu stellen.

**Verschwendung eliminieren** - Das aktive Weglassen von Anforderungen und Funktionalitäten führt zu einer Fokussierung auf die tatsächlich benötigten und wichtigen Features in einem Produkt. Auch der Verzicht auf die technisch "schönste" Lösung kann oftmals zu einer Beschleunigung und Entschlackung der Entwicklung kommen. Überflüssige Arbeit und Features sollen unbedingt vermieden werden.

**Vielfalt** - Im Gegensatz zu homogenen Teams mit ganz ähnlichen Charakteren und Fähigkeiten sind heterogene Teams in der Lage, kreativere Lösungen zu finden. Diesen Vorteil möchten wir nutzen durch eine hohe Vielfalt der einzelnen Teammitglieder.

# Abgeleitete Praktiken für agile Teams

Die folgende Auflistung abgeleiteter Praktiken für agile Teams beinhaltet alle referenzierten Praktiken aus den vorgehend im Detail beschriebenen zwölf agilen Prinzipien hinter dem agilen Manifest. Praktiken und Artefakte, die im weiteren Verlauf des Buches im Detail beschrieben werden, sind in folgender Auflistung nur mit einem entsprechenden Verweis enthalten.

**Aktive Einbindung der echten Anwender** - Niemand kann so gut über die Qualität und die umgesetzten Anforderungen befinden wie die echten Anwender des Produktes. Durch die regelmäßige Einbindung dieser Anwender in alle Aspekte der Produktentwicklung erhält das Team sehr wertvolles und direkt verwertbares Feedback, so dass sowohl die nächsten Anforderungen, als auch das eigene Vorgehensmodell auf den größtmöglichen Nutzen hin verbessert werden können.

**Akzeptanz-Tests** - *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Code Retreat / Coding Dojo** - In der täglichen Arbeit nimmt sich ein Entwickler selten Zeit, sich mit seinem Handwerk auseinander zu setzen, sich neue Techniken anzueignen oder zu prüfen, welche anderen Perspektiven und Entwicklungsmöglichkeiten es bezüglich sei-

ner Arbeit gibt. Formate wie Code Retreats und Coding Dojos geben den Entwicklern den expliziten Freiraum, sich abseits der echten Produktentwicklung mit anderen Entwicklungsmethoden auseinander zu setzen, neue Perspektiven einzunehmen und sich neue Techniken anzueignen.

**Daily Stand-up (Daily Scrum)** - Ein Team muss sich täglich koordinieren, um schnell die notwendigen Entscheidungen über das weitere Vorgehen zur Erreichung seiner Ziele treffen zu können. Diese kurze Besprechung dient nicht zum Statusbericht, sondern ausschließlich zur Koordination des Teams.

**Definition of Done** - *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Dienende Führung (Servant Leadership)** - Das Model der dienenden Führungskraft stellt die klassische Führungshierarchie auf den Kopf. Nicht die Mitarbeiter sind dafür da, der Führungskraft zu dienen, sondern genau anders herum: die wichtigen Personen sind die Mitarbeiter und die Führungskraft hat alles in ihrer Macht stehende zu tun, um ihre Mitarbeiter bestmöglich zu unterstützen.

**Einfaches Design** - Das Software-Design, mit dem die Entwicklungsteams arbeiten, muss so einfach wie möglich gehalten werden, damit es verständlich bleibt und die Wahrscheinlichkeit für langfristige Wartbarkeit und Erweiterbarkeit maximiert wird. Bei neuer Entwicklung muss von Anfang an auf einfaches Design Wert gelegt werden, bei der Weiterentwicklung alter Systeme besteht die Notwendig, das bestehende Design nach und nach zu vereinfachen.

**Ganzes Team** - Ein Team muss zusammenwachsen und Vertrauen aufbauen können, um zu einem guten, schlagkräftigen Team zu werden. Dies benötigt nicht nur Zeit, sondern auch Stabilität. Im Team müssen deshalb alle notwendigen Fähigkeiten und Rollen langfristig besetzt sein.

**Informationsreicher Arbeitsplatz** - Alle wichtigen Ergebnisse, laufenden Arbeiten, Prozessschritte, Anforderungen, Verabredungen und mehr befinden sich idealerweise direkt einsehbar im Team-Raum. Das Ziel ist dabei, bei der Suche nach relevanten Informationen keine Zeit zu verlieren und alles im direkten Zugriff zu haben.

**Inkrementelles Deployment** - Um frühes Feedback zu ermöglichen, muss das Produkt immer wieder zum Anwender gebracht werden können. Durch die Anwendung von inkrementellem Deployment wird das Produkt auf entsprechenden Systemen verfügbar gemacht.

**Kleine Releases** - Durch das Aufbrechen von großen Release-Paketen in kleine Releasezyklen beschleunigt sich das Feedback, das von Anwendern geliefert wird.

**Kollektivbesitz** - Ein effektives Team besteht nicht mehr aus Einzelexperten, denen jeweils ein bestimmter Teilbereich des Systems "gehört", sondern gibt sich Arbeitsvereinbarungen, damit jedes Teammitglied in jedem Teilbereich des Systems arbeiten und Veränderungen vornehmen "darf".

**Leidenschaftliche Arbeit** - Menschen werden dadurch motiviert, dass sie Aufgaben finden, die ihrer technischen

und/oder fachlichen Leidenschaften sehr nahe kommen. Im Gegensatz dazu werden Menschen sehr leicht demotiviert, wenn sie Aufgaben erledigen sollen, die weder ihrer Leidenschaft entsprechen, noch irgend einen daraus ableitbaren Sinn ergeben. Für die Personalpolitik in einem Unternehmen bedeutet dies, dass nicht nur die technisch brillantesten Köpfe angeheuert werden müssen, sondern in erster Linie auch diejenigen Mitarbeiter, die eine sehr große Begeisterung für das Produkt oder die Dienstleistung mitbringen.

**Pair Programming** - Die beste und schnellste Art der inneren Qualitätssicherung kann erreicht werden, wenn während der Entwicklung zwei Teammitglieder gemeinsam versuchen, die richtige Lösung in der richtigen Art und Weise zu bauen. Technische Entscheidungen müssen in solchen Paar-Konstellationen gerechtfertigt werden, konzeptionelle Fehler können bereits im Entstehungsprozess gefunden und vermieden werden.

**Product Backlog -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Produkt Demonstration** - Mit regelmäßigen Demonstrationen von Produktständen bereits während der Entwicklung kann sehr wertvolles Feedback von Product Ownern und Anwendern eingeholt werden. Diese Demonstrationen sind nicht nur auf Meetings wie das Sprint Review Meeting beschränkt, sondern sollen idealerweise täglich vom Team durchgeführt werden.

**Refaktorisierung** - Die kontinuierliche Verbesserung des Software-Designs in kleinen Teilen des Systems, aber

auch in größerem Umfang, ist ein wichtiger Erfolgsfaktor, um zu einem langfristig wartbaren und erweiterbaren Produkt zu gelangen.

**Regelmäßige Auslieferung** - Durch die regelmäßige Auslieferung eines funktionierenden Produktes erhalten die Anwender schnellstmöglich neue Funktionalitäten, die ihre Wertschöpfung erhöhen und den Anwendernutzen optimieren.

**Release Burndown** - Durch das iterative Abtragen der bereits umgesetzten Features (Business Value) in Form eines graphischen Charts wird sehr leicht transparent gemacht, wie viel Wert schon erreicht ist und wie viel Wert noch erreicht werden kann.

**Sprint** - Ein Sprint in Scrum entspricht dem Konzept der iterativen Vorgehensweise und umfasst den gesamten Zyklus von Planning, tägliche Entwicklung, Review und Retrospektive.

**Sprint Backlog -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Sprint Planung -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Sprint Retrospektive -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Sprint Review -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Team-Kontinuität** - Ein Team entfaltet sein gesamtes Potential erst nach und nach, es entwickelt sich mit der Zeit

in einem kontinuierlichen Prozess. Dies wird durch Team-Kontinuität unterstützt. Ein wesentlicher Erfolgsfaktor für Teams ist Vertrauen unter den Teammitglieder und der Aufbau von Vertrauen ist eine langfristige Investition. Veränderungen in der Teamstruktur führen dazu, dass sich die sozialen Aspekte im Team neu ordnen müssen und damit auch das Vertrauen zwischen bestehenden und neuen Teammitgliedern neu gewonnen werden muss.

**Team-Raum** - Ein Team und seine Kommunikation können sich erst richtig entfalten, wenn es einen eigenen Team-Raum gibt, in dem sich die Teammitglieder alle gleichzeitig gemeinsam aufhalten und arbeiten können.

**(agiles) Testen -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Testen durch den Anwender** - Die effektivste Möglichkeit, eine Einschätzung der entwickelten Funktionalitäten eines Produktes zu erhalten, ist die Einbindung der echten Anwender in dem Maße, dass diese Anwender das System funktional und explorativ testen. So werden sehr schnell Erkenntnisse gewonnen über Usability, fehlende bzw. überflüssige Funktionen, sowie die tatsächlichen Workflows der Anwender.

**Testgetriebene Entwicklung -** *siehe Detailbeschreibung im Kapitel „Praktiken"*

**Verschwendung erkennen** - In einem transparenten Vorgehensmodell können überflüssige Arbeitsschritte leicht erkannt werden. In gutem Software-Design können überflüssige Schnittstellen und Teilsysteme

ebenso leicht erkannt werden. Diese Erkenntnisse sollten unbedingt dazu genutzt werden, alle unnötigen Verschwendungen in Prozess und Produkt zu entfernen.

**Zusammen sitzen** - Die effizienteste und effektivste Kommunikation in einem Team geschieht durch die räumliche, physische Nähe der einzelnen Teammitglieder untereinander. In der idealen Konstellation sitzt das gesamte Team in einem großen Teamraum beieinander. Alles, was dieses enge Zusammensitzen beeinträchtigt, führt zu Reibungsverlusten in der Kommunikation. Bereits die Wand zwischen zwei benachbarten Teamräumen kann dazu führen, dass einzelne Teammitglieder nicht mehr gut miteinander kommunizieren. Der verlängerte Weg zum anderen Teammitglied stellt schon eine Hürde für die Kommunikation dar, die erst aktiv überwunden werden muss. Es ist nur menschlich, solche Hürden zu vermeiden. Zusammensitzen in einem Teamraum sorgt dafür, dass diese Hürden minimiert werden.

# Rollen in einem agilen Team

Viele Unternehmen stellen sich bei einer agilen Einführung anfänglich die Frage, welche im medizintechnischen Umfeld vorhandenen Rollen gehören denn eigentlich in ein agiles Team?

Liest man nicht immer wieder, dass in einem Scrum-Team jeder alles können muss und dass es keine Spezialisten mehr geben darf?

Diese Interpretation ist natürlich völliger Unsinn und geht ganz klar an unseren Realitäten in der Medizintechnik vorbei. Wir müssen bei der Fragestellung zwei Aspekte unterscheiden: die fachliche, also medizinische Expertise und die technische, also die umsetzende Expertise. Natürlich würde unser Team auf einem idealen Planeten aus Mitgliedern bestehen, die auf allen Ebenen in der Lage sind, sowohl das Produkt, als auch die Technologien zu durchdringen.

In der Realität haben wir es jedoch beispielsweise mit einem Entwickler zu tun, der die perfekten Filteralgorithmen für Bio-Impedanzsignale in embedded Code gießen kann. Von diesem würden wir aber nie erwarten, dass er sich in der gleichen Qualität mit den Themen Usability für den Anwender oder Prozess-Qualitätsmanagement auseinander setzen kann. Ja, er sollte sich bis zu einem gewissen Grad auch damit auskennen und mit diesen Themen etwas vertraut sein. Und nein, er soll seine Expertise dennoch behalten können.

In einem agilen Team haben wir an die einzelnen Teammitglieder genau diesen Anspruch, sich über die jeweilige tief gehende Spezialisierung hinaus auch mit allen anderen Themen im Projekt etwas auszukennen. Wir sprechen hier vom sogenannten "T-shaped" Entwickler, der sich in der Breite mit vielen Bereichen grundlegend auskennt und in der Tiefe seine Spezialisierung hat.

Welche Rollen gehören nun alle in ein agiles Team?

Idealerweise befinden sich in einem Team alle Rollen, welche benötigt werden, um das zu entwickelnde Produkt an die Anwender ausliefern zu können. Wenn wir eine Wertstromanalyse für eine Produktentwicklung durchführen, dann würden wir also aus jedem der dort identifizierten Bereiche mindestens eine Person in ein agiles Team stecken, damit diese dort gemeinsam dafür sorgen können, dass das richtige Produkt in der richtigen Qualität gebaut und geliefert wird.

In einem überschaubaren Umfeld kann sich dies beispielsweise in einem Team mit folgenden Rollen manifestieren: zwei Tester, drei Software-Entwickler, ein QM-Beauftragter, ein Usability-Experte und zusätzlich ein Product Owner, sowie ein ScrumMaster.

Aber auch hier müssen wir die Realitäten in den Unternehmen betrachten. Der konsequente Ansatz würde in vielen Fällen bedeuten, mehrere Dutzend oder gar Hunderte von Personen in ein Team zu stecken. Wir wissen jedoch genau, dass Teams ab einer Größe von mehr als 7 Mitgliedern signifikant an Geschwindigkeit und Schlagkraft verlieren. Damit sind wir dann in der Fragestellung der agilen Skalierung.

Bleiben wir noch kurz im überschaubaren Umfeld mit einem oder wenigen Teams. Es gibt keine allgemeingültige Antwort auf die Frage, ob beispielsweise ein QM-Beauftragter in ein agiles Team gehört. Es kann die richtige Maßnahme sein, wenn die restlichen Teammitglieder keinerlei Wissen und Erfahrung darin haben, eine richtige Risikoanalyse durchzuführen. Es kann je nach Projektumfeld auch die richtige Maßnahme sein, das Thema QM als Team-externe Dienstleistung zu betrachten, auf die das Team jeweils im Bedarfsfall zugreifen kann. Die gleichen Fragestellungen müssen für jeden Teamkontext neu beantwortet werden, z.B. für die Themen klinische Validierung, Vertrieb, Marketing, Marktanalyse, Datenbank-Administration und viele mehr.

Die Kernfrage hierbei lautet: wie viele Beteiligte aus der Kette **vor** und **nach** der reinen Entwicklung können sinnvoll in einem agilen Team integriert werden?

Die ideale Antwort darauf lautet: so viele wie möglich.

Versuchen Sie am Anfang, in einem einzelnen Team alle notwendigen Personen zu vereinen. Experimentieren Sie mit diesem Team, bis Sie weitere Erkenntnisse darüber gewonnen haben, ob diese Konstellation eine funktionierende ist oder an welcher Stelle es Änderungsbedarf gibt.

Bei größeren Vorhaben mit mehreren Teams müssen Sie die Frage beantworten, ob Sie Feature-Teams oder Komponenten-Teams erzeugen möchten. In den meisten nicht-agilen Unternehmen finden wir Komponenten-Teams, die im Regelfall die Verantwortung für eine bestimmte Komponente oder ein bestimmtes Modul im Gesamtsystem haben. Die Schnittstellen zwischen den Teams entsprechen im Wesent-

lichen den Schnitstellen in diesem Gesamtsystem[5]. Um im Gesamtsystem eine neue Funktionalität bereitstellen zu können, muss in so einem Umfeld normalerweise mit vielen Abhängigkeiten umgegangen werden. Das Herunterbrechen von Feature-Ideen auf einzelne Komponenten und damit auf Teams erfordert ein großes Maß an Vorarbeit. Eine weitere Schwierigkeit besteht in der Integration der einzelnen Team-Ergebnisse, welche auf Grund der hohen Abhängigkeiten oft erst spät durchgeführt wird und damit sehr risikobehaftet ist.

Von genau diesen Hürden wollen wir uns mit agilen Methoden verabschieden. Deswegen ist die ideale Teamaufstellung nicht Komponenten-orientiert, sondern Feature-orientiert. Das bedeutet, dass jedes einzelne Team in der Lage sein soll, ein vollständiges Feature durch alle Komponenten und Schichten eines Gesamtsystems umzusetzen und in einer lieferbaren Zustand zu bringen. Dies mag in großen Organisationen und System unrealistisch klingen. Es gibt jedoch Möglichkeiten, diesen Idealzustand zu weit wie Möglich zu erreichen.

Versuchen Sie herauszufinden, welche Ihrer Systemkomponenten von zentraler Bedeutung sind und welche Fähigkeiten all Ihre Mitarbeiter mitbringen. Ihr Ziel sollte sein, so wenige wie möglich Komponenten-Teams aufzustellen und so viele wie möglich Feature-Teams[6].

---

[5] Dieses Phänomen ist in allen Organisationen beobachtbar und bekannt unter dem Namen „Conway's Gesetz".

[6] Holen Sie sich im Zweifelsfall externe Hilfe durch einen agilen Coach oder Berater. Ein agiler Experte mit neutralem Außenblick kann Ihnen dabei helfen, eine gute Teamaufstellung für die ersten Schritte zu finden.

# Prozesse, Meetings, Reviews

In diesem Abschnitt folgt die detaillierte Beschreibung der benötigten Prozesse, Meetings, Reviews, Interaktionen und Zeremonien in alphabetischer Reihenfolge.

# Acceptance Testing

## Beschreibung

Akzeptanz-Tests sind ein grundlegender Bestandteil von (agilen) Anforderungen. Durch ihre Einhaltung wird sichergestellt, dass (a) die gewünschten Anwenderbedürfnisse befriedigt werden, (b) die umgesetzte Funktionalität durch geprüfte Schnittstellen integriert werden kann und (c) die Anforderung als "fertig" (done) betrachtet werden kann.

Die Basis der (idealerweise automatisierten) Akzeptanz-Tests sind definierte Akzeptanz-Kriterien, welche jeweils zu einzelnen Anforderungen und User Stories gehören. Akzeptanz-Kriterien sind nicht wie klassische Requirements hart beschrieben, sondern formulieren umgangssprachlich die Intention des Anwendernutzens[7].

Akzeptanz-Kriterien definieren den Rahmen einer User Story und werden nach Fertigstellung einer Story dazu benötigt, den tatsächlichen Fertigstellungsgrad[8] fest zu stellen hinsichtlich der ursprünglich formulierten Intention. Sie entstehen in Zusammenarbeit zwischen Product Owner und

---

[7] Im medizintechnischen Umfeld kann es durchaus Fälle geben, in denen technische Schnittstellen zu Geräten und anderer Software z.B. mit harten Performance-Anforderungen spezifiziert werden müssen. Es spricht nichts dagegen, solche Abhängigkeiten mit Hilfe von Akzeptanz-Kriterien zu beschreiben.

[8] Level of Done.

Entwicklungs-Team, sowie idealerweise mit Beteiligung der Kunden bzw. Anwender des Produktes.

Als Hilfsmittel in der agilen Produktentwicklung sind Akzeptanz-Kriterien das geeignete Mittel, um in kurzen Feedbackzyklen[9] fest zu legen, in welche Richtung das gewünschte und benötigte Produkt entwickelt werden soll. Damit hat das Entwicklungs-Team eine Methodik an der Hand, um für sich selbst jederzeit beurteilen zu können, ob es das richtige Produkt entwickelt[10].

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Anwender

## Referenzen

- Lisa Crispin - Agile Testing
- Markus Gärtner - ATDD in der Praxis

---

[9] Iterationen bzw. Sprints.

[10] Die Beurteilung durch das Entwicklungs-Team kann sowohl manuell als auch maschinell durch Entwicklung entsprechender Test-Automation durchgeführt werden. Die Investition in eine gute Test-Automation zahlt sich bereits bei der Entwicklung kleiner Software-Systeme schnell aus, da der Aufwand manueller Akzeptanz-Tests kontinuierlich anwächst und damit nicht mehr vollständig während einer Iteration durchgeführt werden kann.

- Wikipedia - Acceptance Testing
  ➪ `en.wikipedia.org/wiki/Acceptance_testing`
- Johnston, Courtney - User Stories: A Beginner's Guide to Acceptance Criteria
  ➪ `www.boost.co.nz/blog/agile/acceptance-criteria/`
- Agile Alliance - Acceptance Testing
  ➪ `guide.agilealliance.org/guide/acceptance.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Mit Hilfe von definierten Akzeptanz-Tests wird sichergestellt, dass das erstellte Software-Produkt die Bedürfnisse der Anwender befriedigt und die Anforderungen korrekt umgesetzt wurden. |
| § 5.5 | BC | Durch Akzeptanztests wird sichergestellt, dass die entwickelte Software entsprechende Akzeptanzkriterien befriedigt. |
| § 5.5 | BC | Mit Hilfe von Akzeptanz-Tests wird sichergestellt, dass eine in Software umgesetzte Anforderung abgekapselt funktioniert und damit für eine Integration in andere Strukturen bereit ist. |
| § 5.6 | ABC | Akzeptanztests entsprechen Black-Box-Tests. |

# Clean Code / SOLID Principles

## Beschreibung

Clean Code ist ein Wertesystem zur Professionalisierung der Software-Entwicklung. In ihm sind viele Good Practices[11] und grundlegende Prinzipien enthalten, welche durch die Erfahrungen in den letzten Jahrzehnten als gültig und korrekt betrachtet werden können.

Bekannter sind in erster Linie die fünf SOLID-Prinzipien:

- **SRP** - Single Responsibility Principle
  Eine Klasse darf nur aus einem Grund geändert werden.

- **OCP** - Open Closed Principle
  Eine Klasse muss offen sein für Erweiterungen, aber geschlossen gegen Modifikationen.

---

[11] Ich höre immer wieder den Wunsch aus Teams und Organisationen, sie hätten gerne eine Liste mit „Best Practices", die sie befolgen müssen, damit alles wie gewünscht gut wird. Diesem Wunsch möchte ich nicht nachkommen, da es den Lean-Prinzipien widerspricht, einen definierten Regelsatz vorzugeben, der die Lösung aller Probleme darstellt. Jedes Team und jede Organisation muss in ihrem ganz spezifischen Umfeld ihre eigenen „Best Practices" entwickeln. Sie sind jedoch nicht ohne weiteres auf andere Teams und Organisationen transferierbar. In diesem Buch spreche ich deswegen nur von „Good Practices", aus denen sich im Einzelfall „Best Practices" empirisch ableiten können.

- **LSP** - Liskov Substitution Principle
  Eine abgeleitete Klasse verhält sich immer so wie ihre Basisklasse.

- **ISP** - Interface Segregation Principle
  Clients werden nicht mit Details versehen, die sie nicht benötigen.

- **DIP** - Dependency Inversion Principle
  Klassen hoher Ebenen sollen nicht von Klassen niedriger Ebenen abhängig sein, sondern beide sollen von Interfaces/Abstraktionen abhängen. Interfaces sollen nicht von Details abhängen, sondern Details von Interfaces.

Der Nutzen von Clean Code zeigt sich im Software-Life-Cycle-Prozess ab dem Zeitpunkt, an dem die Maintenance-Phase des Produktes beginnt. Dies ist erfahrungsgemäß der längste Zeitraum im gesamten Life-Cycle. Wird das Clean Code-Ziel der intuitiven Verständlichkeit von Quellcode erreicht, kann in der Maintenance-Phase sehr viel Aufwand eingespart werden, in dem das Entwicklungs-Team ansonsten versuchen würde, sich den Quellcode, die darin verborgenen Strukturen und Zusammenhänge zu erarbeiten und nachzuvollziehen.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Robert C. Martin - Clean Code

- Wikipedia - Clean Code
  ➪ `de.wikipedia.org/wiki/Clean_Code`
- Westphal, Ralf; Lieser, Stefan - SOLID
  ➪ `www.clean-code-developer.de/SOLID.ashx`
- Clean Coders - Training Videos. With Personality. For Software Professionals
  ➪ `www.cleancoders.com`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 4 | ABC | Modernes Software-Engineering wird umgesetzt mit der Anwendung von Clean Code und SOLID-Prinzipien, TDD, Unit Tests, Design Patterns und vielen anderen Methoden, die hier nicht vollständig aufgeführt werden können. |
| § 4 | ABC | Durch die korrekte Anwendung objekt-orientierter Prinzipien wie "Separation of Concerns" und "Single Responsibility Principle" entstehen lose gekoppelte Architekturen. |
| § 5.4 | ABC | Durch die Anwendung objekt-orientierter Prinzipien wird die Aufteilung von Software-Einheiten in weitere Einheiten getrieben. |

# Continuous Integration

## Beschreibung

Kontinuierliche Integration beschreibt den Prozess des regelmäßigen, vollständigen Neubildens und Testens einer Anwendung.

Jeder Entwickler checkt seine Änderungen früh und regelmäßig (mindestens täglich) in eine Versionsverwaltung ein. Von dort aus wird das gesamte System neu gebaut und automatisch getestet, damit die Entwickler schnellstmöglich auf vorhandene Fehler im System aufmerksam gemacht werden können.

Die beiden großen Vorteile von Continuous Integration sind zum einen die Vermeidung hoher und teurer Integrationsaufwände am Ende eines Projektes und zum anderen die Möglichkeit, zu jedem beliebigen Zeitpunkt ein lauffähiges Produkt aus der Versionsverwaltung auschecken zu können.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Duvall, Paul M.; Matyas, Steve; Glover, Andrew - Continuous Integration: Improving Software Quality and Reducing Risk

- Wikipedia - Continuous Integration
  ➪ `en.wikipedia.org/wiki/Continuous_integration`
- Fowler, Martin - Continuous Integration
  ➪ `www.martinfowler.com/articles/continuousIntegration.html`
- Agile Alliance - Continuous Integration
  ➪ `guide.agilealliance.org/guide/ci.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.1 | BC | Konsequent angewandte Continuous Integration ist die Aktivität, durch welche eine frühzeitige Integration aller beteiligten Komponenten inklusive der dazugehörigen Integrationstests sichergestellt wird. |
| § 5.6 | BC | Die einzelnen Software-Einheiten werden kontinuierlich integriert. |

## Verknüpfungen zu anderen Praktiken und Artefakten:

- **Versioning System** - Kontinuierliche Integration bedient sich eines Versionierungssystems, um jederzeit kleine Änderungen am System integrieren zu können.
- **Test Automation** - Kontinuierliche Integration benötigt eine funktionierende Test-Automatisierung, um ihren Nutzen entfalten zu können.

# Early Integration / Early Testing

## Beschreibung

Frühzeitige Integration und frühzeitiges Testen sind zum Einen Konzepte der kontinuierlichen Integration mit Test-Automation, zum Anderen können sie auch entwicklungsbegleitend manuell durchgeführt werden.

Im Extremfall können Anforderungen vor ihrer eigentlichen Umsetzung bereits getestet werden (Test-First Development).

Das darunterliegende Prinzip heißt "Schnelles Scheitern" (Fail Fast). Wir möchten es so schnell wie möglich wissen, falls unser Vorhaben nicht erfolgreich sein kann.

Durch die frühe und regelmäßige Integration mit externen Bestandteilen des Produktes aus anderen Teams erhalten wir eine schnelle Feedbackschleife, die uns dabei hilft, die Qualität, Architektur und Machbarkeit des Gesamtproduktes zu beurteilen. Wir können somit vermeiden, am Ende des Projektes hohe Integrationsaufwände investieren zu müssen.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- James Shore - The Art of Agile Development (Kapitel 13.2)
- Luke, Monica - How Early Integration Testing Enables Agile Development
  - ➪ `www.ibm.com/developerworks/rational/library/early-integration-testing-enables-agile-development/`
- Agile Alliance - Integration
  - ➪ `guide.agilealliance.org/guide/integration.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.7 | BC | Anforderungen können bereits vor ihrer endgültigen Umsetzung getestet und integriert werden. |

## Verknüpfungen zu anderen Praktiken und Artefakten

- **Versioning System** - Frühe Integration bedient sich eines Versionierungssystems, um Änderungen am System sofort integrieren zu können.

# Exploratory and Manual Testing

## Beschreibung

Neben einer funktionierenden Test-Automation beschäftigen sich agile, entwicklungsbegleitende Tester mit explorativem und manuellem Testen des Systems.

Beim explorativem Testen wird das System in bestimmten Zielaspekten betrachtet. Es existiert kein vorher festgelegter Testplan, sondern dieser entsteht durch den Tester bei der eigentlichen Testarbeit, sich diesen Zielaspekten im System zu nähern und diese herauszufordern. Dabei entsteht ein Testplan, ein Testentwurf und die tatsächliche Testdurchführung.

Beim manuellen Testen werden vorher festgelegte Testpläne durchlaufen, die entweder aus klassischen Testentwürfen entstehen oder als Ergebnis des explorativen Testens als neue Teiltestpläne in den Gesamttestplan einfließen.

Der Vorteil des explorativen Testens gegenüber dem manuellen Testen ist die Flexibilität des Testers, sich unbekannten, neuen oder veränderten Teilen des System intuitiv zu nähern und diese bezüglich ihrer Qualität zu „erforschen".

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen


- Lisa Crispin - Agile Testing
- Hendrickson, Elisabeth - Exploratory Testing in an Agile Context
  ➪ de.slideshare.net/codecentric/exploratory-testing-inagileoverviewmeettheexpertselisabethhendrickson
- Bach, James - Exploratory Testing Explained
  ➪ www.satisfice.com/articles/et-article.pdf
- Wikipedia - Exploratory Testing
  ➪ en.wikipedia.org/wiki/Exploratory_testing
- Agile Alliance - Exploratory Testing
  ➪ guide.agilealliance.org/guide/exploratory.html


## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.6 | ABC | Exploratives Testen entspricht Black-Box-Tests. |

## Verknüpfung zu anderen Praktiken und Artefakten

- **Test Documentation** - Die benötigte Test Dokumentation wird mit den Ergebnissen der explorativen und manuellen Tests angereichert.

# Integration Testing

## Beschreibung

Integrations-Tests stellen sicher, dass sich die einzelnen Teile eines entwickelten System miteinander verbinden lassen und ein funktionierendes Ganzes entsteht.

Das Integrations-Testen befindet sich zwischen den Unit-Tests und der Validierung des Systems. Im agilen Context wollen wir das System innerhalb einer Iteration vollständig integriert haben mit allen Systemteilen, die in dieser Iteration entstanden sind. Dies führt zu der Notwendigkeit, Integrations-Tests zu jedem beliebigen Zeitpunkt durchführen zu können, was nichts anderes bedeutet, dass auch Integrations-Tests automatisiert werden müssen.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Lisa Crispin - Agile Testing
- Cunningham, Ward - Integration Tests
  ➪ `c2.com/cgi/wiki?IntegrationTest`
- Wikipedia - Integration Testing
  ➪ `en.wikipedia.org/wiki/Integration_testing`

- Agile Alliance - Integration
  ➪ `guide.agilealliance.org/guide/integration.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.6 | BC | Die integrierten Software-Einheiten werden durch Integrationstests verifiziert. Durch Test-Automation kann eine Dokumentation der Testergebnisse erzeugt werden. |
| § 5.6 | BC | Die Software-Integration wird durch Integrationstests verifiziert. |
| § 5.6 | ABC | Integrations-Tests entsprechen Black-Box-Tests bezüglich der Schnittstellen einzelner Systemkomponenten und White-Box-Tests bezüglich der System-Architektur. |
| § 5.6 | ABC | Integrations- sowie System-Tests können mit Simulationen, Prototypen und Mocks durchgeführt werden. |

# Problem Communication

## Beschreibung

Ein Problem-Kommunikationsprozess wird dafür eingesetzt, Stakeholder, Anwender, Kunden und allgemein alle am Produkt interessierten Personen darüber zu informieren, welche Fehler und Probleme sich in dem Produkt befinden und behoben bzw. nicht länger benutzt werden sollten.

In diesem Buch wird nicht weiter auf die Möglichkeiten eingegangen, einen Problem-Kommunikationsprozess zu definieren und zu etablieren, da es sich hierbei um klassische Projektmanagement-Themen handelt, zu denen umfangreiche Referenzen und etablierte Methoden existieren wie z.B. Projektmarketing, Projektkommunikation oder Stakeholder-Management.

## Verantwortliche/beteiligte Rollen

- Product Owner

## Referenzen

- Schelle, Ottmann, Pfeiffer - ProjektManager (beinhaltet das gesamte Projektmanagement-Kompendium der GPM/IPMA)

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.2 | ABC | Probleme und deren Konsequenzen werden über einen "Problem Communication"-Prozess an Anwender kommuniziert. |
| § 9 | ABC | Einzubindende Stakeholder werden durch einen "Problem Communication"-Prozess über die Existenz eines Problems informiert. |

# Product Backlog Grooming

## Beschreibung

Das Product Backlog Grooming Meeting dient dazu, dem Team ein sauberes, verständliches und bereites Product Backlog für mindestens die kommende Iteration verfügbar zu machen.

Die Inhalte des Product Backlogs werden auf Aktualität, Wichtigkeit, Verständnis, Schätzbarkeit und Nutzen geprüft und bei Bedarf entsprechend gesplittet, verfeinert, geändert oder entfernt. Es können auch neue Anforderungen bzw. User Stories entstehen. Grundsätzlich wird die Frage gestellt, ob das Product Backlog und seine enthaltenen Stories der Produktvision entsprechen und den richtigen Weg darstellen, diese Vision umzusetzen. Dabei legen wir den Detailfokus auf die nächsten 3 Sprints. Alle Backlog-Einträge, die sich weiter in der Zukunft befinden, sollen inhaltlich größer, allgemeiner bzw. abstrakter werden[12].

---

[12] Es ist wichtig, die Einträge des Backlogs nach unten hin immer größer, allgemeiner und abstrakter werden zu lassen. Zum Einen führt es dazu, dass das Team einen schnellen Überblick über das Backlog aufrecht erhalten kann und grob abschätzen kann, welche Themen noch umgesetzt werden sollen. Zum Anderen vermeiden wir dadurch, ein bei Projektbeginn bereits durchspezifiziertes Backlog zu erstellen, von dem wir dann nur noch schwer wegkommen. Die empirische Prozesskontrolle agiler Methoden macht sich genau das zum Vorteil, dass wir zu jedem zukünftigen Zeitpunkt flexibel agieren und reagieren können und nicht auf ein damals gültiges, heute jedoch veraltetes Backlog festgelegt sind.

Im Product Backlog Grooming Meeting wird von Product Owner und Team bewertet, welche Änderungen sich am Produkt durch Anforderungen und Feedback aus dem Sprint Review (z.B. Änderungsanträge und User Stories) ergeben. Die Entscheidung darüber, welche Anforderungen umgesetzt werden sollen, liegt einzig und alleine beim Product Owner.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Pichler, Roman - Grooming the Product Backlog
  ➪ `www.romanpichler.com/blog/product-backlog/grooming-the-product-backlog/`
- Druckman, Angela - How to Hold an Effective Backlog Grooming Session
  ➪ `www.scrumalliance.org/community/articles/2011/march/how-to-hold-an-effective-backlog-grooming-session`
- Agile Alliance - Backlog Grooming
  ➪ `guide.agilealliance.org/guide/grooming.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Im Product Backlog Grooming Meeting werden bestehende Anforderungen und User Stories im Product Backlog neu bewertet und aktualisiert. |
| § 5.3 | BC | Im Backlog Grooming werden die Anforderungen an Hardware und Software identifiziert, welche durch den Einsatz von SOUP notwendig werden. |
| § 5.3 | BC | Im Product Backlog Grooming werden die Funktionalitäten und Performance-Anforderungen an einzusetzende SOUP-Komponenten definiert. |
| § 6.1 | ABC | Im Product Backlog Grooming werden Probleme und Fehler von bereits releaster Software besprochen. |
| § 6.2 | ABC | Im Product Backlog Grooming Meeting werden Fehler- und Problem-Berichte bewertet und ihr Einfluß auf die Sicherheit bestehender Produkte und Systeme festgestellt, um notwendige Anforderungsänderungen im Product Backlog abzuleiten. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.2 | ABC | Der Product Owner läßt im Product Backlog Grooming Meeting neue und geänderte Anforderungen vom Team bewerten und erteilt deren Freigabe durch Aufnahme und Einsortierung in sein Product Backlog. |
| § 6.2 | BC | Im Product Backlog Grooming Meeting werden neue und geänderte Anforderungen auf ihren Einfluß auf bestehende Produkte und Systeme untersucht. |
| § 6.3 | ABC | Im Product Backlog Grooming werden Änderungen an der Software analysiert, bewertet und ins Product Backlog einsortiert. Damit durchlaufen Änderungen den gleichen Entwicklungsprozess wie Anforderungen und User Stories. |
| § 7.4 | BC | Im Product Backlog Grooming Meeting durchlaufen neue und geänderte Anforderungen den Risikomanagement-Prozess, um sie mit bestehenden Risiko-Maßnahmen abzugleichen. |
| § 7.4 | ABC | Im Product Backlog Grooming Meeting durchlaufen neue und geänderte Anforderungen den Risikomanagement-Prozess, um neue Risiko-Maßnahmen abzuleiten. |
| § 9 | ABC | Im Product Backlog Grooming werden notwendige Änderungen am System diskutiert und für die Entwicklung freigegeben. |

## Verknüpfungen zu anderen Praktiken und Artefakten

- **Product Backlog** - Im Product Backlog Grooming wird das Product Backlog kontinuierlich überarbeitet und angepasst.
- **User Story** - Im Product Backlog Grooming werden User Stories erstellt, verfeinert, abgeschätzt, verworfen und in einzelne User Stories aufgesplittet.

# Quality Management Prozess

## Beschreibung

Ein zertifiziertes Qualitätsmanagementsystem nach EN-ISO 13485 stellt nach MDD 93/42/EWG ein Verfahren dar, eine CE-Kennzeichnung für Medizinprodukte zu erlangen und damit deren Marktfreigabe zu erwirken. Es dient normalerweise nicht dazu, die Normkonformität nachzuweisen, sondern die Qualitätsmanagementprozesse zu definieren. Produktrealisierungs-, Risikomanagement- und Kommunikationsprozesse sind die integralen Bestandteile dieses Qualitätsmanagementsystems.

Die IEC 62304 fordert nicht explizit ein solches Qualitätsmanagementsystem, es wird aber empfohlen, die Anforderungen der EN-ISO 13485 zu berücksichtigen, da für die Erfüllung der grundlegenden Anforderungen nach MDD 93/42/EWG Anh I nicht alle Anforderungen durch die IEC 62304 abgedeckt werden.

In diesem Buch wird das Thema Qualitätsmanagement nicht weiter vertieft, da es sich auf einer anderen Ebene zum Software-Life-Cycle befindet und unabhängig von der jeweiligen Vorgehensweise umgesetzt werden sollte.

Es gibt jedoch einen ganz wesentlichen Aspekt beim Einsatz agiler Methoden im Zusammenhang mit einem etablierten Qualitätsmanagementsystem. Das grundlegende Prinzip der kontinuierlichen Prozessverbesserung - und damit der kontinuierlichen Prozessveränderung - darf von einem Qualitätsmanagementsystem nicht mit unnötigem, bürokratischen Aufwand verkompliziert und verzögert werden. Beim

Einsatz agiler Methoden benötigen wir ein Qualitätsmanagementsystem, in dem die Kernprozesse so definiert sind, dass sie sich selbst jederzeit schnell und einfach verändern lassen.

## Verantwortliche/beteiligte Rollen

- Qualitätsmanager
- Scrum Master

## Referenzen

- ISO 13485:2003
  Medical devices - Quality management systems - Requirements for regulatory purposes

- Wikipedia - ISO 13485
  ➪ `de.wikipedia.org/wiki/ISO_13485`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 4 | ABC | Ein Qualitäts-Management-Prozess ist definiert und wird eingesetzt. |

# Release Planning Meeting

## Beschreibung

Im Release Planning wird darüber entschieden, welcher Inhalt/Scope in die Produktentwicklung einfließen soll und welche Zeiträume und Meilensteine sich daraus ergeben. Ergebnis des Release Plannings ist ein initiales Product Backlog, mit dem das Team mit der Umsetzung beginnen kann.

Das Release Planning Meeting findet idealerweise vor oder nach einem Team-Kick-Off-Meeting statt.

Gibt es die Notwendigkeit, dieses Meeting regelmäßig durchzuführen[13], dann sollte es mit dem Product Backlog Grooming verschmolzen werden, um nicht ein weiteres, regelmäßiges Meeting einzuführen, welches die Teamkapazität reduzieren würde.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

---

[13] Die Ursachen für Notwendigkeiten dieser Art müssen vom ScrumMaster dringend untersucht und möglicherweise beseitigt werden. Es stecken oft organisatorische Unzulänglichkeiten hinter ständigen, negativen Veränderungen im Produktportfolio. Fehlende Vision, Aktionismus bei Problemen oder unorganisierte Wartungstaktik können grundlegende Ursachen für ständige, negative Veränderungen sein.

## Referenzen

- Baker, Simon - Release Planning
  ➪ `www.energizedwork.com/weblog/2006/04/release-planning.html`
- Wells, Don - Release Planning
  ➪ `www.extremeprogramming.org/rules/planninggame.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.8 | BC | Im Release Planning wird entschieden, welche Maßnahmen und Aktivitäten durchzuführen sind, um eine verläßliche Auslieferung der Software zu gewährleisten. |
| § 6.1 | ABC | Im Release Planning findet die Vorbewertung und Priorisierung statt für Probleme und Fehler von bereits releaster Software. |
| § 6.3 | ABC | Das Release Planning unterscheidet nicht, ob ein Release aus neuen Anforderungen besteht oder aus Änderungen an einem bestehenden System. |
| § 6.3 | ABC | Im Release Planning wird festgelegt, ob ein Release als vollständiges System-Release erstellt wird oder als Patch zu einem bestehenden System-Release. |

## Verknüpfung mit anderen Praktiken und Artefakten

- **Product Backlog** - Im Release Planning wird das Product Backlog initial erstellt bzw. in späteren Release Planning Meetings kontinuierlich überarbeitet und angepasst.

# Risk Analysis and Management

## Beschreibung

In einem Risiko-Management-Prozess werden Gefährdungen für Menschen und Umwelt durch Gebrauch eines Medizinproduktes analysiert, bewertet, durch Maßnahmen reduziert und entsprechend kontrolliert.

In der IEC 62304 nimmt das Thema Risiko-Management eine orthogonale Stellung zum gesamten Software-Life-Cycle-Prozess ein. Das bedeutet, dass sämtliche Aspekte des Life-Cycle-Prozesses vom Risiko-Management umgeben sind. Oder anders ausgedruckt: der Risiko-Management-Prozess findet während aller Entwicklungsaktivitäten und -phasen statt.

In diesem Buch wird nicht näher auf den eigentlichen Risiko-Management-Prozess und seine Umsetzung im Detail eingegangen. Die Anforderungen an den Risiko-Management-Prozess sind in der Norm ISO 14971 beschrieben und von dem jeweils eingesetzten Vorgehensmodell im Software-Life-Cycle-Prozess unabhängig.

## Verantwortliche/beteiligte Rollen

- Qualitätsmanager
- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- ISO 14971:2007
  Medical devices - Application of risk management to medical devices

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 1 | ABC | Risiko-Management wird in erster Linie im Sprint Planning durchgeführt. |
| § 4 | ABC | Risiko-Management wird im Sprint Planning Meeting durchgeführt. |
| § 4 | ABC | Im Sprint Planning wird jeder Anforderung und User Story (auf jeder Ebene) eine Sicherheitsklassifizierung zugeordnet durch die Anwendung der Risiko-Analyse. |
| § 5.2 | BC | Im Sprint Planning Meeting wird sichergestellt, dass alle Maßnahmen aus der Risiko-Analyse einer in diesem Sprint geplanten Anforderung in das Product Backlog als Anforderung einfließen. |
| § 5.2 | ABC | Im Sprint Planning Meeting durchlaufen alle Anforderungen eine Risiko-Analyse, um notwendige Maßnahmen abzuleiten. |
| § 5.3 | ABC | Die Sicherheitsklassifizierung wird im Sprint Planning für alle umzusetzenden Software-Einheiten überdacht und gegebenenfalls neu vergeben. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.3 | C | Im Sprint Planning werden die Architektur und die Software-Einheiten der umzusetzenden Anforderungen definiert. Die identifizierten Software-Einheiten durchlaufen die Risiko-Analyse. |
| § 5.7 | BC | Notwendige Änderungen an der Software, die durch System-Tests erkannt werden, durchlaufen während des Sprint Plannings die notwendige Risikoanalyse. |
| § 7.1 | BC | Im Sprint Planning werden Workflows und sequentielle Abfolgen von Events in der Software durch die Risiko-Analyse auf Gefahrensituationen untersucht. |
| § 7.1 | BC | Im Sprint Planning werden User Stories daraufhin untersucht, ob sie zu einer Gefahrensituation beitragen können, die in der Risiko-Analyse enthalten ist. |
| § 7.1 | BC | Im Sprint Planning werden für fehlerhaftes Verhalten der Software mögliche Ursachen in Hardware- und Software-Fehlern analysiert. |
| § 7.1 | BC | Im Sprint Planning werden die umzusetzenden Software-Einheiten durch die Risiko-Analyse auf Gefahrensituationen untersucht und die Ergebnisse dokumentiert. |
| § 7.1 | BC | Im Sprint Planning werden die umzusetzenden Anforderungen und deren abgeleitete Software-Einheiten durch die Risiko-Analyse auf mögliche Gefährdungen hin untersucht. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 7.1 | BC | Im Sprint Planning werden mögliche Ursachen für Fehlverhalten von SOUP durch die Risiko-Analyse identifiziert. |
| § 7.1 | ABC | Im Sprint Planning wird die Risiko-Analyse der umzusetzenden Architektur und ihrer Software-Einheiten durchgeführt. |
| § 7.1 | BC | Im Sprint Planning werden SOUPs und ihre bekannten Fehler/Abweichungen durch die Risiko-Analyse betrachtet. |
| § 7.1 | BC | In der Risiko-Analyse während des Sprint Planning Meetings werden alle Anforderungen auf möglichen Mißbrauch untersucht und entsprechende Maßnahmen abgeleitet. |
| § 7.2 | BC | Im Sprint Planning werden zu jeder Software-Einheit mit möglichen Gefahrensituationen entsprechende Maßnahmen identifiziert und festgehalten. |
| § 7.2 | BC | Im Sprint Planning wird der zu erstellenden Software-Einheit eine Sicherheitsklassifizierung zugewiesen (A, B, C), welche weitere Details im Entwicklungsprozess bestimmt. |
| § 7.2 | BC | Im Sprint Planning Meeting wird sichergestellt, dass alle Maßnahmen aus der Risiko-Analyse einer in diesem Sprint geplanten Anforderung in das Product Backlog als Anforderung einfließen. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 9 | ABC | Im Sprint Planning Meeting werden Probleme und Fehler auf sicherheitsrelevante Aspekte mit Hilfe der Risikoanalyse untersucht. |

# Scrum

## Beschreibung

Scrum ist ein agiles Management-Framework für die empirische Prozesskontrolle. Es kann ideal eingesetzt werden bei jeder Art von Produktentwicklung.

Scrum basiert auf den drei Grundpfeilern

- **Transparenz** - für jeden Beteiligten an der Produktentwicklung müssen die verwendeten Prozesse, Inhalte, Anforderungen, Ziele, Visionen, etc. erkennbar sein, damit ein gemeinsames Verständnis entstehen kann und zielgerichtete Teamarbeit möglich wird.

- **Inspektion/Überprüfung** - alle Ergebnisse und Artefakte, sowie die Art und Weise der Zusammenarbeit werden in regelmäßigen Abständen betrachtet und hinsichtlich der beabsichtigten Zielerreichung überprüft.

- **Adaption/Anpassung** - die Erkenntnisse aus der Inspektion/Überprüfung fließen ein in konkrete Veränderungen am Prozess oder der gewünschten Ziele und Inhalte.

Jedes Meeting in Scrum beinhaltet die Aspekte von Inspektion und Adaption, somit ist die empirische Prozesskontrolle bzw. ein kontinuierlicher Verbesserungsprozess Grundbestandteil von Scrum.

In diesem Buch werden die einzelnen Bestandteile von Scrum beschrieben. Es kann jedoch die Lektüre grundlegender Scrum-Bücher nicht ersetzen. Für einen Einstieg in

die Thematik sind die unten aufgelisteten Referenzen geeignet.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Ken Schwaber - Agiles Projektmanagement mit Scrum
- Roman Pichler - Scrum

- scrum.org - Scrum Guide
  ➪ `www.scrum.org/Scrum-Guides`

- scrum.org - What is Scrum?
  ➪ `www.scrum.org/Resources/What-is-Scrum/`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 1 | ABC | Scrum ist ein gültiges Prozess-Framework für den Software-Life-Cycle. |

# Software Maintenance Process

## Beschreibung

Der Software Maintenance-Prozess (Software Wartungsprozess) beschreibt alle Aktivitäten, die für die Änderung von sich im Markt befindlichen Medizinprodukten notwendig sind. Dazu gehört das Einholen und Bewerten von Anwender-Feedback sowie ein installierter Fehlerbehebungsprozess.

Die in diesem Buch beschriebenen Praktiken und Artefakte bilden fast den vollständigen Software Maintenance-Prozess ab, bis auf das explizite Einholen von Feedback bezüglich bereits releaster Produkte. Während der Entwicklung wird diese Anforderung mit Hilfe des Sprint Review Meetings abgedeckt, nach Projektabschluss werden andere Methoden benötigt.

In diesem Buch wird nicht näher auf die Art und Weise eingegangen, wie dieses Feedback von Anwendern und internen Mitarbeitern eingeholt werden kann, da dies unabhängig von dem jeweils eingesetzten Vorgehensmodell im Software-Life-Cycle-Prozess geschehen muss.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- Feggins, Reedy - Using Agile Practices to Support Software Maintenance
  - ➪ `www.ibm.com/developerworks/mydeveloperworks/blogs/c914709e-8097-4537-92ef-8982fc416138/entry/march_10_2012_11_27_am8?lang=en`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.2 | ABC | Über den Software Maintenance Prozess erfolgt das Einholen des Feedbacks zu releaster Software von tatsächlichen Anwendern. |
| § 6.2 | ABC | Über den Software Maintenance Prozess erfolgt das Einholen des Feedbacks zu releaster Software innerhalb der eigenen Organisation. |

# Sprint Planning

## Beschreibung

Im Sprint Planning Meeting einigen sich der Product Owner und das Team auf ein Sprint-Ziel und das dafür umzusetzende Sprint Backlog. Dieses wird aus dem Product Backlog abhängig von den dort gegebenen Prioritäten genommen.

Ein Eintrag im oberen Teil des Product Backlog hat idealerweise folgende Eigenschaften:

- klein genug, damit das Team ihn innerhalb eines Sprints umsetzen kann
- geschätzt, damit die Diskussion über den Inhalt und die Intention des Eintrages stattgefunden hat
- unabhängig[14], damit das Team sofort mit der Umsetzung beginnen kann
- klar definierte Akzeptanzkriterien, damit das Team weiß, wann die Umsetzung als fertig betrachtet werden kann

Diese Eigenschaften finden wir im Artefakt „Definition of Ready“ wieder, welches genau dafür sorgt, dass diese Eigen-

[14] Unabhängigkeit ist nicht unter allen Umständen herstellbar. Wenn sich Abhängigkeiten zwischen Einträgen im Product Backlog ergeben, dann müssen diese Einträge so angeordnet werden, dass sie in einer sinnvoll umsetzbaren Reihenfolge vom Team in den Sprint genommen werden können. Jeff Sutherland spricht mittlerweile nicht mehr von der Eigenschaft „unabhängig (independant)“, sondern von „sofort umsetzbar (immediately actionable)“.

schaften vor dem Sprint Planning Meeting erarbeitet werden.

Für die einzelnen Einträge des Sprint Backlogs werden im zweiten Teil des Sprint Plannings vom Team konkrete Architektur- und Umsetzungsaufgaben identifiziert und geplant.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Cohn, Mike - Sprint Planning Meeting
  ➪ `www.mountaingoatsoftware.com/scrum/sprint-planning-meeting`
- Huether, Derek - Simple Cheat Sheet to Sprint Planning Meeting
  ➪ `www.leadingagile.com/2012/08/simple-cheat-sheet-to-sprint-planning-meeting/`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Im Sprint Planning Meeting wird sichergestellt, dass über die Anforderungen ein einheitliches Verständnis bei allen Beteiligten herrscht. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Im Sprint Planning Meeting werden bestehende Anforderungen und User Stories im Product Backlog neu bewertet und aktualisiert. |
| § 5.2 | ABC | Im Sprint Planning Meeting wird vom Team festgestellt, ob sich Anforderungen im Product Backlog widersprechen. |
| § 5.3 | BC | Im Sprint Planning werden für benötigte SOUP-Komponenten die Funktionalitäten und Performance-Anforderungen festgelegt. |
| § 5.3 | ABC | Die grundlegenden Software-Strukturen und Architektur-Entscheidungen können im Sprint Planning getroffen werden. |
| § 5.3 | BC | Im Sprint Planning werden die Architektur und die Schnittstellen der umzusetzenden Software-Einheiten sowie der einzubindenden externen Software-Einheiten festgelegt oder entsprechende Architekturaufgaben definiert, deren Ergebnisse im nächsten Sprint Review Meeting bewertet werden. |
| § 5.4 | BC | Im Sprint Planning werden die umzusetzenden Anforderungen in eine Architektur überführt und diese wiederum in Software-Einheiten runtergebrochen oder es werden entsprechende Architekturaufgaben definiert, deren Ergebnisse im nächsten Sprint Review Meeting bewertet werden. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.5 | BC | Im Sprint Planning Meeting wird sichergestellt, dass zu jeder Anforderung Akzeptanz-Kriterien für die Integration in andere Strukturen erstellt werden. |
| § 5.7 | BC | Im Sprint Planning erfolgt die Planung der Integrations-Tests sowie der System-Tests. |
| § 6.1 | ABC | Im Sprint Planning finden die notwendigen Risiko-Analysen sowie die Umsetzungsplanung statt für Probleme und Fehler von bereits releaster Software. |
| § 7.1 | BC | Spätestens im Sprint Planning Meeting werden Anforderungen auf ihre korrekte und vollständige Definition geprüft. |
| § 7.2 | BC | Im Sprint Planning werden Software-Einheiten, die eine Risiko-Maßnahme umsetzen, genauso behandelt wie alle anderen Anforderungen und befolgen den definierten Entwicklungsprozess. |
| § 8.2 | ABC | Im Sprint Planning Meeting werden vom Team alle Tasks identifiziert, welche für die Umsetzung einer Änderung (erneut) durchgeführt werden müssen. |
| § 9 | ABC | Im Sprint Planning werden die Fehler- und Problemberichte analysiert. |

## Verknüpfung mit anderen Praktiken und Artefakten

- **Sprint Development Plan = Sprint Backlog** - Im Sprint Planning Meeting wird das Sprint Backlog bzw. der Sprint Development Plan festgelegt.
- **Iteration Notes (Sprint Notizen)** - Im Sprint Planning werden die Iteration Notes (Sprint Notizen) angefangen.
- **Architecture Documentation** - Im Sprint Planning Meeting wird in der zweiten Phase (Sprint Planning 2) die Architektur der umzusetzenden Anforderungen besprochen, skizziert und festgehalten.

# Sprint Retrospective

## Beschreibung

In der Sprint Retrospektive hat das Team die Möglichkeit, Verbesserungsmaßnahmen für die eigenen Prozesse und Strukturen zu identifizieren und zu beschließen. Es geht dabei nicht darum, einen Schuldigen zu identifizieren für die Dinge, die nicht funktioniert haben. Vielmehr ist die Zielsetzung einer Retrospektive, aus der letzten Iteration konkrete Maßnahmen und Veränderungen abzuleiten, die das Team sofort umsetzen kann.

Die Moderation einer Retrospektive erfordert sehr viel Soft Skills vom Scrum Master. Empathie, Mediation, konstruktiver Umgang mit Konflikten, sowie die Fähigkeit, Entscheidungsprozesse im Team zu unterstützen, sind nur einige Aspekte, die ein guter Moderator mitbringen muss.

Auf die vielen Möglichkeiten und Aktivitäten, mit denen eine Retrospektive gestaltet werden kann, wird in diesem Buch nicht weiter eingegangen. Der interessierte Leser findet in den unten aufgeführten Referenzen entsprechende Literatur zu dem Thema.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Norman Kerth - Project Retrospectives
- Esther Derby, Diana Larsen - Agile Retrospectives
- Löffler, Marc - Retrospektiven in der Praxis
- Kua, Patrick - The Retrospectives Handbook
  Leanpub, 2012
  ➪ `leanpub.com/the-retrospective-handbook`

- Agile Alliance - Heartbeat Retrospective
  ➪ `guide.agilealliance.org/guide/heartbeatretro.html`

## Umgesetzte Anforderungen aus der Norm

*Keine direkt umzusetzenden Anforderungen aus der Norm.*

## Verknüpfung mit anderen Praktiken und Artefakten

- **Team Charter / Working Agreements**: Der Team Charter und die Working Agreements können vom Team im Sprint Retrospective Meeting an veränderte Situationen und gegebene Notwendigkeiten angepasst werden.
- **Definition-of-Ready**: Die Definition-of-Ready kann vom Team im Sprint Retrospective Meeting an veränderte Situationen und gegebene Notwendigkeiten angepasst werden.
- **Definition-of-Done**: Die Definition-of-Done kann vom Team im Sprint Retrospective Meeting an veränderte Situ-

ationen und gegebene Notwendigkeiten angepasst werden.

- **Software Development Plan**: Der Software Development Plan kann vom Team im Sprint Retrospective Meeting an veränderte Situationen und gegebene Notwendigkeiten angepasst werden.

# Sprint Review

## Beschreibung

Im Sprint Review Meeting wird vom Team vorgestellt, was im letzten Sprint alles geschehen ist und welche User Stories bzw. Anforderungen umgesetzt sind. Ziel des Sprint Review Meetings ist es, Feedback von allen beteiligten Stakeholdern zu erhalten und auf dieser Basis das Product Backlog für die kommenden Sprints anzupassen.

Es ist ein weitverbreiteter Irrtum, dass das Sprint Review Meeting dafür genutzt werden soll, die umgesetzten Stories und Anforderungen final zu beurteilen, abzunehmen und frei zu geben. Dies soll nach Möglichkeit bereits während des Sprints geschehen, sobald die einzelnen Stories und Anforderungen vom Team umgesetzt wurden. Auch hier gilt das Prinzip des frühen Feedbacks. Das Sprint Review Meeting ist gewissermaßen die letzte Möglichkeit für den Product Owner zur Abnahme und Freigabe, birgt jedoch immer die Gefahr, dass Fehler und Mißverständnisse zu spät entdeckt werden und es damit nicht mehr möglich ist, diese noch im Sprint zu beseitigen.

Ein weiterer Irrtum besteht darin, das Sprint Review Meeting als reine Demonstration der umgesetzten Features zu nutzen. Eine Demo des entstandenen Produktinkrementes ist zwar ein wichtiger Bestandteil des Sprint Review Meetings, es geht jedoch viel mehr darum, das Feedback von echten Anwendern und Stakeholdern einzuholen.

Folgende Fragestellungen sollen im Sprint Review Meeting erörtert werden:

- Sind wir auf dem richtigen Weg mit unserer Produktentwicklung?
- Hilft das, was wir erzeugt haben, wirklich dem Anwender?
- Welche (neuen) Erkenntnisse haben die Anwender und Stakeholder bezüglich der weiteren Produktgestaltung gewonnen?
- Welche Teile des kommenden Product Backlogs sind noch wichtig, welche sind obsolet?

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master
- Stakeholder

## Referenzen


- Cohn, Mike - Sprint Review Meeting
  ➪ `www.mountaingoatsoftware.com/scrum/sprint-review-meeting`
- Bradley, Charles - Tips for a Good Sprint Review
  ➪ `www.scrumcrazy.com/Tips+for+a+Good+Sprint+Review`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 1 | ABC | Im Sprint Review wird die Konsistenz geprüft zwischen der erzeugten Software, ihrer Anforderungen und weiterer Ergebnisse bzw. Inputs. |
| § 1 | ABC | Im Sprint Review werden alle Ergebnisse des gesamten Software-Life-Cycles auf gegenseitige Konsistenz geprüft. |
| § 4 | ABC | Im Sprint Review Meeting wird festgestellt, ob das erzeugte Software-Produkt die Bedürfnisse und Anforderungen der Anwender befriedigt. |
| § 4 | ABC | Im Sprint Review Meeting wird festgestellt, ob die Entwicklung des erzeugten Software-Produkts den normativen Anforderungen entspricht. Sind die normativen Anforderungen in der Definition-of-Done entsprechend formuliert, kann diese als Checkliste im Sprint Review Meeting genutzt werden. |
| § 5.2 | ABC | Die Ergebnisse und Erkenntnisse aus dem Sprint Review Meeting fließen als Feedback direkt in die Anforderungen ein und können im anschließenden Sprint Planning Meeting bedacht werden. |
| § 5.4 | C | Im Sprint Review wird präsentiert, dass das umgesetzte Software-Design der Software-Architektur entspricht. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.5 | BC | Im Sprint Review zeigt das Team, welche Akzeptanzkriterien in Form von Testfällen die Software erfolgreich durchlaufen hat. |
| § 5.6 | BC | Im Sprint Review werden die Integrationstests auf Korrektheit geprüft. |
| § 5.7 | BC | Im Sprint Review wird über Verifikation und Testen berichtet. |
| § 5.7 | BC | Im Sprint Review Meeting berichtet das Team über die durchgeführten Tests aller umgesetzten Anforderungen. |
| § 5.7 | BC | Im Sprint Review werden die umgesetzten Anforderung auf Basis ihrer Akzeptanzkriterien abgenommen und freigegeben.<br><br>Hinweis: das Sprint Review Meeting ist nicht dafür gedacht, die umsetzten Features durch den Product Owner abnehmen zu lassen. Es stellt nur den letzten Zeitpunkt dar, an dem dies geschehen kann. Sinnvollerweise werden Features bereits während des Sprints direkt nach der Umsetzung vom Product Owner abgenommen. |
| § 5.7 | BC | Im Sprint Review Meeting präsentiert das Team die erzeugten und durchgeführten Tests zu den umgesetzten Anforderungen. (Es werden keine Tests erstellt, zu denen keine Anforderungen existieren.) |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.7 | ABC | Im Sprint Review werden die entwickelten und durchlaufenen Tests, sowie die Test-Abdeckung demonstriert. |
| § 5.8 | BC | Im Sprint Review werden die umgesetzten Anforderungen evaluiert und freigegeben, bevor sie in ein Software-Release gelangen können. Dies beinhaltet auch die Bewertung der durchgeführten Regressionstests. |
| § 7.3 | BC | Im Sprint Review werden umgesetzte Risiko-Maßnahmen auf neue mögliche Gefahrensituationen hin untersucht. |
| § 7.3 | BC | Im Sprint Review wird die Umsetzung aller Risiko-Maßnahmen berichtet und verifiziert. |
| § 9 | ABC | Problem- und Fehlermeldungen werden während des Sprints auf Trends und Tendenzen hin untersucht. Die Ergebnisse dieser Untersuchung werden im Sprint Review vorgestellt. |
| § 9 | ABC | Im Sprint Review wird berichtet, welche neuen Probleme und Fehler gefunden wurden. |
| § 9 | ABC | Im Sprint Review wird berichtet, welche Änderungen (Change Requests) an der Software durchgeführt wurden. |
| § 9 | ABC | Im Sprint Review wird berichtet, welche Fehler und Probleme behoben wurden. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 9 | ABC | Im Sprint Review wird berichtet, ob Problemlösungen dazu geführt haben, nachteilige Entwicklungen umzukehren. |

## Verknüpfung mit anderen Praktiken und Artefakten

- **Iteration Notes (Sprint Notizen)** - Im Sprint Review werden die Iteration Notes (Sprint Notizen) geschrieben und fertiggestellt.
- **Release Notes** - Im Sprint Review werden die Release Notes inkrementell erweitert.

# Team Kick-Off Meeting

## Beschreibung

Im Team Kick-Off-Meeting einigt sich das Team auf die grundlegenden Projektziele, die anzuwendenden Vorgehensweisen und Methoden, sowie die Art und Weise der eigenen Zusammenarbeit. In diesem Meeting werden der initiale Software-Entwicklungsplan und der Software-Maintenance-Plan entworfen.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- MindTools - Team Charters (mittlerweile kostenpflichtiger Artikel)
  ➪ `www.mindtools.com/pages/article/newTMM_95.htm`

## Umgesetzte Anforderungen aus der Norm

*Keine direkt umzusetzenden Anforderungen aus der Norm.*

## Verknüpfung mit anderen Praktiken und Artefakten

- **Definition-of-Done** - Die Definition-of-Done wird initial in einem Team Kick-Off Meeting festgelegt.
- **Definition-of-Ready** - Die Definition-of-Ready wird initial in einem Team Kick-Off Meeting festgelegt.
- **Software Entwicklungsplan** - Der Software Entwicklungsplan wird initial in einem Team Kick-Off Meeting festgelegt.
- **Software-Maintenance-Plan** - Der Software Maintenance Plan wird initial in einem Team Kick-Off Meeting erstellt.
- **Team Charter/Working Agreements** - Ein Team Charter und die Working Agreements eines Teams werden initial in einem Team Kick-Off Meeting festgelegt.

# Test Automation

## Beschreibung

Eine Test-Automation ist die Basis für kontinuierliche Integration und ermöglicht eine effiziente Verifikation nach jeder kleinen Änderung am System.

Oft wird die Frage gestellt, ob wirklich jeder Test automatisiert werden soll. Die gültige Antwort darauf lautet: jeder Test, der mehr als ein mal durchgeführt werden wird, sollte automatisiert werden.

Durch Test-Automation wird ein Sicherheitsnetz hergestellt, welches uns ermöglicht, jederzeit ein lauffähiges Gesamtsystem herstellen zu können. Die gewonnene Sicherheit meldet uns Fehler, die sich durch Seiteneffekte in Schnittstellen und Funktionalitäten eingeschlichen haben. Diese Fehler können dann sofort beseitigt werden, so dass wir mit einem funktionierenden Gesamtsystem weiter arbeiten können.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Jez Humble, David Farley - Continuous Delivery
- Lisa Crispin - Agile Testing
- Markus Gärtner - ATDD in der Praxis

- Jeffries, Ron - Automating „All“ Tests
  ➪ `xprogramming.com/articles/automatedtesting/`
- Jeffries, Ron - Automating Story Tests
  ➪ `xprogramming.com/blog/automating-story-tests/`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.5 | BC | Durch Test-Automation wird sichergestellt, dass die Software immer wieder die definierten Akzeptanzkriterien in Form von automatisierten Tests erfolgreich durchläuft. |
| § 5.5 | BC | Durch Test-Automation können die Ergebnisse aller Unit-Tests als Dokumentation erzeugt werden. |
| § 5.6 | BC | Regressionstests vorhandener Software-Versionen können durch Test-Automation jederzeit sicherstellen, dass keine Fehler im Software-System entstanden sind. |
| § 5.6 | BC | Durch Test-Automation können bestehende Testfälle jederzeit wiederholt werden. |
| § 5.6 | BC | Mit einer Test-Automation kann durch die Durchführung aller vorhandenen Tests eine Dokumentation der Test-Resultate erzeugt werden. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.7 | BC | Mit Hilfe einer umfassenden Test-Automation kann sichergestellt und nachgewiesen werden, dass alle Anforderungen getestet und verifiziert sind. |
| § 5.7 | BC | Durch Test-Automation können die Testfälle aller Software-Anforderungen jederzeit durchgeführt werden. |
| § 5.7 | ABC | Durch Test-Automation kann das gesamte Software-System nach einer Änderung vollständig mit allen vorhandenen Testfällen geprüft werden. |
| § 5.7 | BC | Durch Test-Automation kann das gesamte Software-System nach einer Änderung vollständig mit allen vorhandenen Testfällen geprüft werden. |
| § 5.7 | BC | Durch Test-Automation können alle vorhandenen Testfälle des Software-Systems nach Änderungen jederzeit durchgeführt werden. |
| § 8.2 | ABC | Durch Test-Automation kann das Software-System jederzeit nach einer Änderung verifiziert werden. |

## Verknüpfung mit anderen Praktiken und Artefakten

- **Test Documentation** - Die benötigte Test Dokumentation wird mit Hilfe der Test Automation bei Bedarf auf Knopfdruck automatisch erzeugt.
- **Continuous Integration** - Kontinuierliche Integration benötigt eine funktionierende Test-Automatisierung, um ihren Nutzen entfalten zu können.

# Test-Driven Development (TDD) / Test-Driven Design

## Beschreibung

Test-Driven Development (TDD) ist eine Entwicklungspraktik aus dem Extreme Programming (XP). Durch den angewendeten Test-First-Ansatz wird sichergestellt, dass (a) sämtlicher Produktivcode mit Tests versehen ist und (b) nur soviel Produktivcode entsteht, wie auch tatsächlich benötigt wird.

Die Vorteile von Test-Driven Development (TDD) sind mindestens folgende:

- Deutlich geringere Fehlerraten während und nach der Entwicklung eines Systems.
- Durchgängig testbarer Code entsteht durch die Anwendung von Test-Driven Development (TDD), so dass Testbarkeit nicht nachträglich und umständlich eingebaut werden muss.
- Ein jederzeit lauffähiges System entsteht automatisch.
- Eine ideale Entwicklerdokumentation entsteht durch die geschriebenen Tests, da diese Beispielimplementierungen für die Schnittstellen im Code bereit stellen.

Die anfängliche Investition in Test-Driven Development (TDD) zahlt sich sehr schnell durch die gewonnene Sicherheit und die geringeren Fehlerraten aus.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Kent Beck - Test-Driven Development
- Steve Freeman, Nat Pryce - Growing Object-Oriented Software, Guided by Tests

- Martin, Robert C. - The Three Laws of TDD
  ➪ `butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd`

- Cunningham, Ward - Test Driven Development
  ➪ `c2.com/cgi/wiki?TestDrivenDevelopment`

- Agile Alliance - Test-Driven Development
  ➪ `guide.agilealliance.org/guide/tdd.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.3 | BC | Durch die Anwendung von Test-Driven Development (TDD) entsteht während der Umsetzung von Anforderungen eine funktionierende Software-Architektur. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.4 | C | Durch die Anwendung von Test-Driven Development (TDD) entsteht ein "Design by Contract" über die Schnittstellenspezifikation aller Software-Einheiten in Form von ausführbaren Tests gegen diese Schnittstellen. |
| § 5.4 | C | Durch die Anwendung von Test-Driven Development (TDD) entsteht die Schnittstellenspezifikation aller Software-Einheiten in Form von ausführbaren Tests gegen diese Schnittstellen. |
| § 5.4 | ABC | Durch Test-Driven Development (TDD) wird direkt bei der Umsetzung von Software-Einheiten entschieden, wann und wie diese weiter untergliedert werden in einzelne Einheiten. |
| § 5.4 | BC | Durch den Einsatz von Test-Driven Development (TDD) entstehen Software-Einheiten und ihre Schnittstellen auf unterster Ebene, welche die darüberliegenden Software-Strukturen verfeinern. |
| § 5.5 | BC | Durch Test-getriebene Entwicklung (TTD) wird sichergestellt, dass die entwickelte Software entsprechende Akzeptanzkriterien auf unterster Ebene befriedigt. |
| § 5.5 | ABC | Mit Test-Driven Development (TDD) entsteht das Design des zu erstellenden Codes durch ausführbare Schnittstellen-Tests. Diese werden dann durch produktiven Code erfolgreich durchlaufen. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.5 | BC | Durch Test-getriebene Entwicklung wird die Software-Verifizierung sichergestellt. TDD ist ein definierter, korrekter Prozess. |

## Verknüpfung mit anderen Praktiken und Artefakten:

- **Architecture Documentation** - Die Architektur Dokumentation auf Detailebene entspricht der Menge aller ausführbaren Tests, welche durch Test-Driven Development entstehen.

# Unit Testing

## Beschreibung

Unit Tests sind kleine Programme, welche automatisiert gegen Schnittstellen von Methoden auf unterster Code-Ebene testen.

Unit Tests werden idealerweise durch die Anwendung von Test-Driven Development (TDD) erzeugt und damit mit dem Test-First-Verfahren entwickelt: zunächst wird ein Unit-Test implementiert, welcher eine noch nicht existierende Funktionalität testet. Dieser Unit-Test schlägt fehl (wird rot). Die nicht exisitierende Funktionalität wird dann mit dem einfachsten Mittel realisiert, bis der Unit-Test durchläuft (grün wird). Danach kann der Code der umgesetzten Funktionalität durch geeignetes Refactoring verbessert und „clean" gemacht werden.

Unit-Tests werden programmiert und gehören damit genau so zum Code, wie jeder „normale" Code auch. Dies wird oftmals vergessen mit dem Effekt, dass die Unit-Test-Code-Basis nach und nach erodiert und selbst in die Wartbarkeitsfalle hineinläuft. Durch regelmäßiges Refactoring des Test-Codes kann dieser Effekt vermieden werden.

Zum einen sind Unit-Tests White-Box-Tests gegen Schnittstellen auf tiefster Code-Ebene, zum anderen finden sich unter dem Begriff Unit-Test auch allgemeine Tests auf der Ebene beliebiger Software-Einheiten. Die Terminologie „Unit-Test" führt immer wieder zu Verwirrungen. Eine team- oder organisationsspezifische Definition des Begriffes "Unit Test" ist daher im Vorfeld notwendig.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Roy Osherove - The Art of Unit Testing
- Osherove, Roy - The Art of Unit Testing - Official Book Site
  ➪ `artofunittesting.com`
- Wikipedia - Unit Testing
  ➪ `en.wikipedia.org/wiki/Unit_testing`
- Cunningham, Ward - Unit Test
  ➪ `c2.com/cgi/wiki?UnitTest`
- Wells, Don - Unit Tests
  ➪ `www.extremeprogramming.org/rules/unittests.html`
- Agile Alliance - Unit Testing
  ➪ `guide.agilealliance.org/guide/unittest.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.4 | ABC | Software-Einheiten dürfen mit Unit Tests getestet werden, ohne die Einbindung anderer Software-Einheiten. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.5 | BC | Die Software wird durch Unit Tests auf unterster Ebene verifiziert. Durch Test-Automation kann eine Dokumentation der Testergebnisse erzeugt werden. |
| § 5.5 | C | Technische Akzeptanzkriterien, wie z.B. Sequenzen, Abläufe, Kontrollflüsse, Fehlerbehandlung, Initialisierung und Grenzwerte, werden mit Hilfe von Unit Tests definiert. |
| § 5.6 | ABC | Unit Tests entsprechen Black-Box-Tests bezüglich der atomaren Schnittstellen und White-Box-Tests bezüglich der Software-Strukturen. |

# Usability Process

## Beschreibung

Die Usability-Prozesse dienen dazu, das vom System bereitgestellte User Interface auf Gebrauchstauglichkeit hin zu untersuchen, zu prüfen, sowie entsprechendes Anwenderfeedback zu erhalten.

In agilen Methoden erhalten wir dieses Feedback durch die direkte Einbindung der echten Anwender unseres Systems. Dies kann beispielsweise im Sprint Review Meeting geschehen, aber auch in eigenen Usability-Veranstaltungen beim Anwender.

In diesem Buch wird nicht näher auf Usability-Prozesse eingegangen, da diese in einer eigenen Norm IEC 62366 behandelt und für viele Beispiele im Detail beschrieben sind.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Qualitätsmanager

## Referenzen

- IEC 62366 - Edition 1.0 - 2007-10
  Medical devices - Application of usability engineering to medical devices

- Agile Alliance - Usability Testing
  ➪ `guide.agilealliance.org/guide/usability.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Ein Gebrauchstauglichkeitsprozess ist definiert und wird umgesetzt. |

# Zero Bug Tolerance

## Beschreibung

Zero Bug Tolerance sorgt dafür, dass während einer Produktentwicklung keine Fehler verwaltet werden, sondern dass diese behoben werden. Das Verwalten von Fehlern und die Bewertung und Auswertung dadurch entstehender Fehlerlisten kostet eine große Menge an Zeit und Energie, welche die Effizienz für wichtige Entwicklungsaufgaben reduziert.

Auf den ersten Blick mag es utopisch und unrealistisch klingen, keine Fehler verwalten zu wollen. Dies würde in letzter Konsequenz dazu führen, kein Fehler-Tracking-System mehr zu benötigen[15].

Die Anwendung der Zero Bug Tolerance ist jedoch denkbar einfach:

- Entdeckte/gefundene Fehler werden schnellstmöglich von Team und Product Owner bewertet. Sie verbleiben nicht erst lange „auf Halde" in einem eigenen Verwaltungssystem, sondern kommen direkt zur Bewertung auf den Tisch.
- Entweder handelt es sich um einen dringlichen Fehler, welche eine bereits umgesetzte Funktionalität beeinträchtigt. Dann wird dieser Fehler weit oben im Product Back-

[15] In der Realität werden Fehler-Tracking-Systeme notwendigerweise weiterhin dafür genutzt, Daten und Entscheidungen zu Fehlern nachvollziehbar zu dokumentieren.

log einsortiert, um möglichst in der nächsten Iteration behoben werden zu können.
- Oft handelt es sich um einen nicht nachvollziehbaren oder sogar unwichtigen „Fehler", der die Funktionalität für den Anwender in keiner Weise einschränkt. Diese Art von Fehler können direkt gelöscht werden[16], da der Nutzen ihrer Behebung in keinem Verhältnis zum Aufwand steht.
- Manchmal wird aus einer Fehlermeldung ein Feature-Wunsch. In diesem Fall kann ein entsprechender Eintrag vom Product Owner im Product Backlog angelegt werden.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen


- Hazrati, Vikas - The Holy Grail of Zero Defect Systems
  ➪ `www.infoq.com/news/2011/02/zero-defect-systems`

- Spolsky, Joel - The Joel Test: 12 Steps to Better Code
  ➪ `www.joelonsoftware.com/articles/fog0000000043.html (Punkt 5)`


[16] Dieses Vorgehen erfordert Mut und die Einsicht, dass durch das Aufbewahren alter und unwichtiger Fehlermeldungen kein Mehrwert für eine spätere Fehlerbehebung gewonnen wird. Sollte Ihr Umfeld dieses Vorgehen nicht zulassen, dann versuchen Sie zumindest, solche Fehlermeldungen in ein Archiv zu verschieben, welches nicht jederzeit Sichtbar ist und den Blick auf das Wesentliche verschleiert.

- Radcliff, Deb - Zero Tolerance for Bugs
  ➪ sdt.bz/32548

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.6 | BC | Eine konsequente Zero-Bug-Tolerance führt dazu, dass alle Fehler, welche während der Software-Integration gefunden werden, schnellstmöglich behoben oder als irrelevant definiert werden. |
| § 5.7 | BC | Eine konsequente Zero-Bug-Tolerance führt dazu, dass alle Fehler, welche während der Software- und System-Tests gefunden werden, schnellstmöglich behoben oder als irrelevant definiert werden. |
| § 5.7 | ABC | Bei einer Zero Bug Tolerance muss nicht jedes Problem und nicht jeder Fehler behoben werden. Falls entschieden wird, einen Fehler nicht zu beheben, muss die Begründung dokumentiert werden. Dies kann direkt im entsprechenden Backlog-Eintrag erfolgen. |
| § 5.8 | BC | Bei Anwendung der Zero Bug Tolerance durchläuft jeder Fehler und jede Abweichung die Risiko-Analyse. Nur wenn ein Fehler und eine Abweichung nicht zu einem inakzeptablen Risiko beitragen, darf entschieden werden, den Fehler und die Abweichung nicht zu beheben. |

# Dokumente, Artefakte, Pläne

In diesem Abschnitt folgt die detaillierte Beschreibung der benötigten Dokumente, Artefakte und Pläne in alphabetischer Reihenfolge.

# Architecture Documentation

## Beschreibung

Architektur-Dokumentation wird von der IEC 62304 gefordert und kann auf Detailebene durch die Menge aller ausführbaren Tests aus dem Test-Driven Development (TDD), auf höherer Ebene durch Architekturplanung im Sprint Planning Meeting beschrieben werden.

Bei der Architektur-Dokumentation gibt es drei Aspekte, welche beschrieben werden sollten:

- Design-Entscheidungen und ihre Begründungen
- angewendete Design Patterns
- eingesetzte Entwicklungs-Frameworks

Es ist nicht wichtig, jede einzelne Code-Zeile in die Architektur-Dokumentation einfließen zu lassen. Sie sollte aus Sicht des Lesers geschrieben sein, so dass sich neue Teammitglieder (Entwickler und Tester) leicht zurecht finden können. Dies bedeutet auch, dass sich die Architektur-Dokumentation auf möglichst wenige Seiten beschränkt.

Durch entsprechende Werkzeuge lässt sich Architektur-Dokumentation in UML-Form automatisch erzeugen, so dass jederzeit die aktuelle Architektur eingesehen werden kann.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- Zörner, Stefan - „Frag das Team!“ für Architekturdokumentation?
  ➪ `www.oose.de/teamblog/team/frag-das-team-fur-architekturdokumentation/`
- Amoussou, Joel - Software Architecture Documentation in Agile Projects
  ➪ `efasoft.blogspot.de/2010/10/software-architecture-documentation-in.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.3 | BC | Mit der Architektur-Dokumentation ist es möglich, die Architektur zu verifizieren. |
| § 5.3 | BC | Die umgesetzte Software-Architektur kann mit Hilfe von UML-Diagrammen beschrieben und dokumentiert werden. |

# Definition of Done

## Beschreibung

Die Definition-of-Done beschreibt alle Aktivitäten bzw. Zustände, welche eine Anforderung durchlaufen muss, bevor sie als "abgeschlossen" betrachtet werden darf.

Ein typischer Dialog zwischen Projektleiter und Entwickler lautet wie folgt.

> *Projektleiter: „Ist das Feature fertig?"*
> *Entwickler: „Ja, das Feature funktioniert und ist fertig."*
> *Projektleiter: „Also können wir es zum Kunden ausrollen?"*
> *Entwickler: „Nein, dafür müssen die Tester nochmal ran und es muss noch ins Deployment aufgenommen werden."*

Während der kurzen Iterationen agiler Prozesse möchten wir sicher sein, dass eine Anforderung tatsächlich umgesetzt ist und wir diese nicht noch einmal anfassen müssen, um sie abzuschließen. Was dies tatsächlich bedeutet, wird über die Definition-of-Done festgelegt.

Die Definition-of-Done kann für jedes Team unterschiedlich sein, da es in jedem Umfeld eigene Aspekte gibt, die vom Team umgesetzt werden müssen. Wichtig ist in jedem Fall, dass zwischen Product Owner und Entwicklungs-Team Einigkeit darüber herrscht, was die Aussage „die Anforderung ist fertig" bedeutet.

Die einzelnen Aspekte einer Definition-of-Done betrachten nicht die Funktionalität einer Anforderung, sondern deren Qualität.

Typische Inhalte einer Definition-of-Done lauten beispielsweise:

- Code ist im Versionierungssystem eingecheckt
- Build-Server läuft ohne Warnungen und Fehler durch
- alle Unit-Tests sind nach dem Build grün
- Pair Programming oder Code-Review wurde durchgeführt
- Product Owner hat „OK" gegeben
- etc.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen


- Panchal, Dhaval
  ➪ `www.scrumalliance.org/community/articles/2008/september/what-is-definition-of-done-(dod)`

- scrum.org - Definition of Done
  ➪ `www.scrum.org/Resources/Scrum-Glossary/Definition-of-Done`

- Agile Alliance - Definition of Done
  ➪ `guide.agilealliance.org/guide/sashimi.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 1 | ABC | Mit der Einhaltung einer Definition-of-Done (DoD) wird sichergestellt, dass fertiggestellte Ergebnisse weiterverarbeitet werden können. |
| § 5.1 | ABC | Der Risiko-Management-Prozess ist für jede einzelne Anforderung in der Definition-of-Done enthalten. |
| § 5.1 | ABC | Die Definition-of-Done enthält alle Verizifierungsaufgaben, die für eine Anforderung durchgeführt werden müssen. |
| § 5.1 | ABC | Die Definition-of-Done beschreibt alle durchzuführenden Aktivitäten, um eine Anforderung vollständig umzusetzen. |
| § 5.1 | BC | Die Definition-of-Done beinhaltet das Einchecken von umgesetzten Anforderungen in ein Versionierungssystem. |
| § 5.6 | ABC | Eine Anforderung oder User Story kann als erledigt betrachtet werden, wenn die entsprechenden Integrations- und System-Tests mit einer Simulation, einem Prototypen oder einem Mock durchgeführt wurden. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.7 | BC | Die Definition-of-Done stellt sicher, dass Verifikation und entsprechende Test-Prozeduren eingehalten werden. |
| § 5.8 | BC | In der Definition-of-Done wird festgelegt, dass die Software bzw. jede User Story verifiziert sein muss, bevor sie ausgeliefert werden darf. |
| § 5.8 | BC | Durch die Definition-of-Done wird sichergestellt, dass alle Aufgaben zum Zeitpunkt eines Software-Release abgeschlossen sind. |

# Definition of Ready

## Beschreibung

Die Definition-of-Ready beschreibt alle Aktivitäten bzw. Zustände, welche eine Anforderung durchlaufen muss, bevor sie bereit ist, in einen Sprint aufgenommen und umgesetzt zu werden.

Wie die Definition-of-Done kann auch die Definition-of-Ready für jedes Team unterschiedlich sein.

Das wesentliche Ziel der Definition-of-Ready ist das gemeinsame Verständnis zwischen Product Owner und Entwicklungs-Team darüber, welche Idee eine Anforderung oder User Story repräsentiert und welcher Anwendernutzen tatsächlich umgesetzt werden soll.

Typische Aspekte einer Definition-of-Ready beinhalten beispielsweise:

- Eine Story ist geschätzt, was bedeutet, dass eine inhaltliche Diskussion zwischen Product Owner und Entwicklungs-Team stattgefunden hat.
- Eine Story ist klein genug, um innerhalb einer Iteration umgesetzt werden zu können.
- Eine Story hat definierte Akzeptanzkriterien, damit sie während der Iteration vom Team geprüft werden kann.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Pichler, Roman - The Definition of Ready
  ➪ `www.romanpichler.com/blog/product-backlog/the-definition-of-ready/`
- Agile Alliance - Definition of Ready
  ➪ `guide.agilealliance.org/guide/ready.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Die Definition-of-Ready einer Anforderung beinhaltet eine Bewertung durch das Team darüber, ob diese Anforderung bereit ist, in einem Sprint umgesetzt zu werden. |
| § 7.1 | BC | Die Definition-of-Ready einer Anforderung fordert deren korrekte und vollständige Definition. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 8.2 | ABC | Die Umsetzung von Anforderungen, welche das bestehende Software-System verändern (in einem inkrementell-iterativen Prozess also alle Anforderungen), muss vom Product Owner beauftragt sein. Dies kann mit Hilfe der Definition-of-Ready sichergestellt werden. |

# Iteration Notes (Sprint Notizen)

## Beschreibung

Iteration Notes (Sprint Notizen) beschreiben:

- was in einer Iteration passiert ist,
- wer beteiligt war,
- wann die Iteration stattfand,
- welche Anforderungen umgesetzt wurden und
- welche Fehler, Probeme und Risiken identifiziert wurden.

Iteration Notes (Sprint Notizen) dienen zum Einen der Stakeholder-Kommunikation mit Kunden, Anwendern und Magement, die nicht am Sprint Review Meeting teilnehmen können. Zum Anderen wird auf diesem Wege eine Projektdokumentation aufgebaut, welche den Entstehungsprozess des Systems für Auditoren, aber auch für neue Teammitglieder nachvollziehbar macht.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.6 | BC | Die Iteration Notes (Sprint Notizen) beinhalten die Liste der in der Iteration beteiligten Team-Mitglieder, wie zum Beispiel auch die Tester. |
| § 5.8 | BC | Nicht behobene Fehler und Abweichungen werden in den Iterations-Notizen festgehalten. |
| § 7.3 | BC | Die Iteration Notes (Sprint Notizen) beinhalten die umgesetzten Maßnahmen aus der Risikoanalyse und deren Verifizierung im aktuellen Sprint. |
| § 9 | ABC | In den Iteration Notes (Sprint Notizen) sind alle Fehlerberichte aus externen Quellen enthalten, sowie deren abgeleitete Tests zur Verifikation. |

# Linkable IDs

## Beschreibung

Verweisende IDs sind die Basis für sämtliche Anforderungen an die Traceability. Sie können automatisch z.B. über eine Versionskontrolle oder eine Anwendung zur Verwaltung des Product Backlogs vergeben werden. Auch die manuelle Vergabe bei der Arbeit mit einem reinen Papier-Product Backlog ist möglich.

In diesem Buch wird nicht weiter auf die Techniken zur Vergabe und den Umgang mit Linkable IDs eingegangen.

Wichtig an dieser Stelle ist zu erwähnen, dass die Traceability eine wesentliche Voraussetzung ist, um die Dokumentation der geplanten Aktivitäten und der Ergebnisse nachvollziehbar zu erstellen. Projekt-Auditierungen durch benannte Stellen werfen durch eine gute Nachvollziehbarkeit sehr viel weniger Fragen auf, so dass die Umsetzung eines durchgängigen, funktionierenden Traceability-Systems es im Regelfall einen lohnenswerten Aufwand darstellt.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | User Stories und darüberliegende bzw. externe Anforderungen werden mit Hilfe eindeutiger, referenzierbarer IDs versehen. |
| § 5.2 | ABC | User Stories und darüberliegende bzw. externe Anforderungen werden mit Hilfe eindeutiger, referenzierbarer IDs versehen. |
| § 7.3 | BC | Mit Hilfe von verweisenden IDs kann die Traceability hergestellt werden von erkannten Gefahrensituationen über Maßnahmen über umzusetzende Software-Einheiten bis zur Verifikation dieser Software-Einheiten. |

# Product Backlog

## Beschreibung

Das Product Backlog ist eine nach Wichtigkeit sortierte Liste aller Wünsche, Ideen und Anforderungen an das zu entwickelnde System. Das Product Backlog ist kein fixes Dokument mit starrem Umfang, sondern muss sich im Laufe der Entwicklung kontinuierlich verändern, um dem empirischen Prozess agiler Methoden genüge zu tun und Anwenderfeedback einfließen zu lassen.

Die wichtigsten, also obersten Einträge im Product Backlog werden mit Hilfe der Definition-of-Ready in einen Zustand gebracht, dass sie vom Entwicklungs-Team im Sprint Planning Meeting für die nächste Iteration eingeplant werden können.

Das Product Backlog hat nichts mit einer klassischen Software-Requirements-Specification (SRS) zu tun[17]. Es ist ein lebendes Dokument, welches sich im Laufe der Zeit verändern soll, um die jeweiligen Erkenntnisse abzubilden. Das Product Backlog soll nur Einträge enthalten, die unmittelbar dazu beitragen, das Projekt- oder Produktziel zu erreichen.

Der Product Owner trägt die Verantwortung für das Product Backlog und dessen kontinuierlicher Pflege.

---

[17] Dies soll nicht bedeuten, das ein Product Backlog nicht wie eine SRS gehandhabt werden kann. Es ist eben nur keine klassische, sondern eine agile und dynamische Software-Requirements-Specification.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- Roman Pichler - Agiles Produktmanagement mit Scrum
- Cohn, Mike - Scrum Product Backlog
  ➪ `www.mountaingoatsoftware.com/scrum/product-backlog`
- Agile Alliance - Backlog
  ➪ `guide.agilealliance.org/guide/backlog.html`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Zu erzeugende Anwender-Dokumentation wird als Anforderung im Product Backlog verwaltet. |
| § 5.2 | ABC | Anforderungen aus Normen und Richtlinien werden im Product Backlog verwaltet. |
| § 5.2 | ABC | Das Product Backlog enthält alle Sicherheits-Anforderungen (Security). |
| § 5.2 | ABC | Das Product Backlog enthält alle Anforderungen, die sich aus dem Risikomanagement-Prozess ergeben. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Im Product Backlog befinden sich grobe System-Anforderungen, welche iterativ in kleinere Software-Anforderungen verfeinert werden. |
| § 5.7 | BC | Fehler, welche während der System-Verifikation gefunden und nicht sofort behoben werden, werden in das Product Backlog einsortiert. |
| § 5.8 | BC | Im Product Backlog sind Maßnahmen und Aktivitäten enthalten, die eine verläßliche Auslieferung der Software gewährleisten. |

# Release Notes

## Beschreibung

Release Notes beinhalten alle für den Anwender, Kunden und Operator des Systems notwendigen Informationen, um das System aufsetzen und benutzen zu können.

Dieses Buch geht nicht weiter auf das Thema „Release Notes“ ein, da es sich nicht um ein spezifisches, agiles Artefakt handelt.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.8 | ABC | In den Release Notes ist die Version des Software-Release enthalten. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.2 | ABC | Die Release Notes einer Software beinhalten alle Änderungen an dem Software-System seit dem letzten Release. |

# Software Development Plan

## Beschreibung

Der Software-Entwicklungsplan beschreibt die grundlegenden Vorgehensweisen für die Produktentwicklung, sowie die eingesetzten Werkzeuge und die Teamzusammensetzung.

Dieses Buch geht nicht weiter auf das Thema „Software Development Plan“ ein, da es sich dabei nicht um ein spezifisches, agiles Artefakt handelt.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master
- Qualitätsmanager

## Referenzen

- IEC 62304 - Edition 1.0 - 2006-05
  Medical device software - Software life cycle processes

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.1 | ABC | Der Software-Entwicklungsplan erwähnt Scrum als eingesetztes, agiles Vorgehensmodell mit dem darin enthaltenen Sprint Review Meeting, in welchem die Forderungen aus Paragraph 4.1 umgesetzt werden: Sicherstellung der Umsetzung von Kundenanforderungen sowie regulatorischen Anforderungen. |
| § 5.1 | ABC | Im Software Development Plan werden folgende Scrum-Artefakte aufgeführt:<br><br>* Product Backlog<br><br>* Sprint Backlog<br><br>* Definition of Done<br><br>* Team Charter / Working Agreements<br><br>* Definition of Ready<br><br>* Iteration Notes (Sprint Notizen)<br><br>* Architecture Documentation |
| § 5.8 | BC | Der Software-Entwicklungsplan beschreibt die Vorgensweise zur Erstellung eines Software-Release, sowie das dafür notwendige Environment. |

# Software Maintenance Plan

## Beschreibung

Der Software-Wartungsplan beschreibt die Prozesse zur Fehlerbehandlung und dem Umgang mit Anwenderfeedback nach der Markteinführung eines Produktes.

Dieses Buch geht nicht weiter auf das Thema „Software Maintenance Plan“ ein, da es sich dabei nicht um ein spezifisches, agiles Artefakt handelt.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Qualitätsmanager

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.1 | ABC | Im Software Maintenance Plan wird festgelegt, anhand welcher Kriterien ein Feedback als Problem/Fehler betrachtet wird. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 6.1 | ABC | Im Software Maintenance Plan wird der Einsatz des Versionierungs-Systems beschrieben. |
| § 6.1 | ABC | Im Software Maintenance Plan werden alle Aktivitäten und Aufgaben des Maintenance Prozesses beschrieben. |
| § 6.1 | ABC | Im Software Maintenance Plan wird festgelegt, wie nach einem Release das Feedback eingeholt, dokumentiert, ausgewertet, umgesetzt und verfolgt wird. |

# Sprint Development Plan = Sprint Backlog

## Beschreibung

Das Sprint Backlog ist das Ergebnis des Sprint Planning Meetings und beinhaltet alle Anforderungen und Aktivitäten, die während des nächsten Sprints umgesetzt werden sollen.

Im Sprint Planning Meeting nimmt sich das Entwicklungs-Team so viele Einträge aus dem priorisierten Product Backlog, wie es zutraut, im nächsten Sprint umsetzen zu können. Diese Einträge repräsentieren das Sprint Backlog. Im zweiten Teil des Sprint Planning Meetings erzeugt das Entwicklungs-Team für jeden einzelnen Eintrag im Sprint Backlog die entsprechenden Tasks, die durchgeführt werden müssen, um den Eintrag umzusetzen.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Cohn, Mike - Sprint Backlog
  ➪ `www.mountaingoatsoftware.com/scrum/sprint-backlog`
- Agile Alliance - Iteration
  ➪ `guide.agilealliance.org/guide/iteration.html`
- Felix, Luciano - 9 Tips for Creating a Good Sprint Backlog
  ➪ `www.scrumalliance.org/community/articles/2009/march/9-tips-for-creating-a-good-sprint-backlog`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.1 | BC | Das Sprint Backlog enthält alle Anforderungen, welche innerhalb eines Sprints in das Gesamtsystem integriert und integrationsgetestet werden. |
| § 5.1 | ABC | Im Sprint Backlog sind alle Produktergebnisse enthalten, welche auch im Sprint verifiziert werden müssen. |
| § 5.1 | ABC | Das Sprint Backlog beinhaltet iterativ-inkrementell auch alle Anforderungen, welche sich aus dem Gesamtsystem an die Software ergeben. |
| § 5.1 | ABC | Der Software-Entwicklungsplan wird iterativ-inkrementell verändert durch die entstehenden Sprint Backlogs. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.6 | BC | Fehler, welche während der Software-Integration auftauchen, werden im Sprint Backlog gepflegt, um sie schnellstmöglich zu beheben. |
| § 9 | ABC | Fehlerberichte werden im Sprint Backlog verwaltet, um schnellstmöglich behoben zu werden. |

# Team Charter / Working Agreements

## Beschreibung

Ein Team Charter beschreibt die grundlegenden Arbeitsweisen innerhalb eines Teams, die eingesetzten Methoden und Werkzeuge, sowie die Ziele und Vision des Projektes. Im weitesten Sinne gehören auch die Definition-of-Done und die Definition-of-Ready zu den Working Agreements.

Team Charter und Working Agreements unterscheiden sich von Team zu Team, da die Inhalte jeweils aus der vorhandenen Team-Kultur entstehen. Ob Pünktlichkeit auf der Liste steht oder eine Meetingkultur ohne Mobiltelefone verabschiedet wird, liegt ausschließlich am Team.

In diesem Buch wird das Thema nicht weiter behandelt, da es keine „Best Practice"-Beispiele gibt. Jedes Team muss für sich selbst definieren, wie es zusammen arbeiten möchte und welche teaminternen Regeln auf die Team Charter und Working Agreements geschrieben werden sollen.

Als Anhaltspunkt mögen folgende Fragen dienen:

- Welche Werte sind uns wichtig für die Zusammenarbeit als Team?
- Welche Verhaltensregeln wollen wir uns geben?
- Welche fachlichen Regeln an unsere Arbeit wollen wir uns geben?

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team
- Scrum Master

## Referenzen

- Bunio, Terry - Agile Development Team Charter
  ➪ `www.infoq.com/articles/agile-development-team-charter`
- Derby, Esther - Norms, Values, Working Agreements, Simple Rules
  ➪ `agile.dzone.com/news/norms-values-working`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.1 | BC | Die Working Agreements eines Teams beschreiben alle Tools und Dinge, welche das endgültige Produkt beeinflussen könnten. |
| § 5.1 | ABC | Die Working Agreements eines Teams beschreiben das eingesetzte Versionierungssystem sowie dessen grundsätzliche Handhabung. |
| § 5.1 | C | In den Working Agreements eines Teams wird festgelegt, mit welchen Standards, Methoden und Tools die Umsetzung von Anforderungen in funktionierende Software geschieht. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.1 | ABC | Im Team Charter wird die Sprint-Länge festgelegt und damit die iterativen Meilensteine, zu denen umgesetzte Anforderungen verifiziert werden. |

# Test Documentation

## Beschreibung

Test-Dokumentation soll nach Möglichkeit auf Knopfdruck erzeugt werden können und sich auf die wesentlichen Ergebnisse und Aussagen beschränken.

Die Test-Dokumentation beinhaltet folgende Fragmente:

- Software Development Plan: beschreibt die Test-Tool-Chain, sowie eine Liste der Tester (und Entwickler) im Team
- Sprint-Notizen: beschreiben gefundene und offene Fehler und Abweichungen, sowie die jeweils aktuelle Software-Version
- Ergebnisse der automatisierten Unit-, Integrations- und Regressionstests

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 9 | ABC | In der Test-Dokumentation werden folgende Informationen abgelegt: Test-Ergebnisse, gefundene Abweichungen, getestete Software-Version, Hardware- und Software-Konfigurationen, eingesetzte Test-Tools, Test-Datum, Test-Personal |

## Verknüpfung mit anderen Praktiken und Artefakten:

- **Exploratory Testing** - Die benötigte Test Dokumentation wird mit den Ergebnissen der explorativen und manuellen Tests angereichert.

# User Story

## Beschreibung

Eine User Story steht in diesem Kontext für sämtliche Typen von Anforderungen, die in einem Product Backlog oder Sprint Backlog enthalten sein können.

Unter den Begriff fallen:

- User Story,
- technische Story,
- Fehlerbeschreibung,
- Problembeschreibung,
- Anforderung,
- etc.

Echte User Stories helfen dem Product Owner und dem Entwicklungs-Team dabei, den Fokus auf den Anwendernutzen zu legen. Bei jeder Diskussion über Inhalte und Anforderungen an das zu erstellende System sollen Fragen folgender Art gestellt werden:

- „Für welchen Anwender erzeugt dies genau welchen Nutzen?"
- „Welches Problem wird durch die Umsetzung dieser Story behoben?"

User Stories befinden sich anfänglich im Product Backlog und durchlaufen ihren Lebenszyklus über die Definition-of-Ready in das Sprint Backlog des Entwicklungs-Teams. Dort werden sie zerlegt in Tasks, die während der Iteration abgearbeitet werden, bis die User Story über die Definition-of-

Done vom Product Owner abgenommen wird und aus der weiteren Betrachtung herausfällt.

## Verantwortliche/beteiligte Rollen

- Product Owner
- Entwicklungs-Team

## Referenzen

- Mike Cohn - User Stories Applied
- Agile Alliance - User Stories
  ➪ `guide.agilealliance.org/guide/stories.html`
- Wells, Don - User Stories
  ➪ `www.extremeprogramming.org/rules/userstories.html`
- Cohn, Mike - User Stories
  ➪ `www.mountaingoatsoftware.com/topics/user-stories`

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 4 | ABC | User Stories, welche durch Story Splitting aus einer übergeordneten User Story entstanden sind, erben die Sicherheitsklassifizierung dieser übergeordneten User Story. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 4 | ABC | Die Sicherheitsklassifizierung wird als Attribut jeder User Story zugeordnet, welche die Risiko-Analyse durchlaufen hat. |
| § 4 | ABC | Einer User Story, der noch keine Sicherheitsklassifizierung zugeordnet wurde, wird per Default die Sicherheitsklassifizierung C zugeordnet. |
| § 5.1 | ABC | Eine User Story besteht neben der reinen Beschreibung auch aus definierten Akzeptanztests, welche zur Verifizierung der Umsetzung benötigt werden. |
| § 5.2 | ABC | Anforderungen an die Gebrauchstauglichkeit werden über User Stories definiert. |
| § 5.2 | ABC | Anforderungen an die Wartung durch den Anwender werden mit User Stories definiert. |
| § 5.2 | ABC | Daten-Definitionen und Datenbank-Anforderungen werden über entsprechende User Stories spezifiziert. |
| § 5.2 | ABC | Die Eingaben und Ausgaben des gesamten Software-Systems werden über entsprechende User Stories definiert. |
| § 5.2 | ABC | Anforderungen an die Installation des Software-Produktes und deren Akzeptanzkriterien werden über entsprechende User Stories beschrieben. |
| § 5.2 | ABC | Schnittstellen von und zu anderen Systemen werden über User Stories beschrieben. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.2 | ABC | Eine User Story beschreibt eine funktionale Anforderung bzw. eine benötigte Fähigkeit des zu erstellenden Produkts. |
| § 5.2 | ABC | Eine User Story beinhaltet alle ermittelten Maßnahmen aus ihrer eigenen Risiko-Analyse. |
| § 5.2 | ABC | Alarme, Warnungen und andere wichtige Meldungen für den Anwender werden mit Hilfe von User Stories definiert. |
| § 5.2 | ABC | Anforderungen an Betrieb und Wartung des Systems werden mit User Stories beschrieben |
| § 5.2 | ABC | User Stories sind so formuliert, dass sich Test- und Akzeptanz-Kriterien ableiten lassen. |
| § 5.7 | BC | Eine User Story beinhaltet alle Testfälle, welche notwendig sind, um die Anforderungen dieser User Story prüfen und sicherstellen zu können. |
| § 5.7 | BC | User Stories beinhalten neben der Definition einer Anforderung auch deren Akzeptanz- und Testkriterien. |
| § 6.2 | ABC | User Stories können auch Fehler- und Problemberichte darstellen, welche Abweichungen der Anforderungen und der Spezifikation beinhalten. |
| § 8.2 | ABC | Eine User Story, welche auf Basis eines Problems formuliert wird, beinhaltet den zugehörigen Problembericht, um Traceability herzustellen. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 9 | ABC | In User Stories können Fehler- und Problemberichte beschrieben sein. |
| § 9 | ABC | User Stories können Fehlermeldungen, Change Requests, sowie andere Einträge enthalten, um das Systemverhalten zu ändern oder zu korrigieren. |
| § 9 | ABC | Die Ergebnisse von Fehleranalysen und -bewertungen werden in der User Story bzw. dem Fehlereintrag direkt festgehalten. |

# Versioning System

## Beschreibung

Ein Versionierungssystem ist ein grundlegendes Werkzeug für die Entwicklung von Softwareprodukten.

In diesem Buch wird das Thema „Versionierungssystem" nicht näher beschrieben, da es sich nicht um ein spezifisches, agiles Artefakt handelt.

## Verantwortliche/beteiligte Rollen

- Entwicklungs-Team

## Referenzen

- keine

## Umgesetzte Anforderungen aus der Norm

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 5.8 | BC | Durch ein Versionierungs-System werden alle Bestandteile eines Software-Release archiviert. |

| § | SK | Herstellung der Normkonformität |
|---|---|---|
| § 8.1 | ABC | Alle zu einem Software-System gehörenden Bestandteile und deren Versionen werden in einem Versionierungs-System abgelegt. Aus diesem Versionierungs-System kann die gültige, vollständige System-Konfiguration aller Bestandteile festgestellt werden. |
| § 8.1 | ABC | Mit einem Versionierungs-System können alle zu einem Software-System gehörenden Komponenten und deren Versionen identifiziert und verwaltet werden. |
| § 8.1 | ABC | Die im Software-System eingesetzten Dritt-Party-Komponenten werden in ihrem Ursprungszustand mit einem Versionierungs-System verwaltet. |
| § 8.3 | ABC | Mit Hilfe des Versionierungs-Systems kann die Historie aller Bestandteile des Software-Systems nachvollzogen werden. |

## Verknüpfung mit anderen Praktiken und Artefakten:

- **Linkable IDs** - Verweisende Identifier (Linkable IDs) werden beim Einsatz eines Versionierungssystems automatisch vergeben, da Change Sets mit einheitlichem Zeitstempel versehen sind und komplette Versionierungsstände mit Labels versehen werden können.

- **Kontinuierliche Integration** - Kontinuierliche Integration bedient sich eines Versionierungssystems, um jederzeit kleine Änderungen am System integrieren zu können.
- **Early Integration** - Frühe Integration bedient sich eines Versionierungssystems, um Änderungen am System sofort integrieren zu können.

# Von den Anforderungen der Norm zu Praktiken und Artefakten

Dieses Kapitel beinhaltet die Gesamttabelle aller Normtexte mit Verweis aus den entsprechenden Paragraphen und die Sicherheitsklassifizierung, sowie eine Beschreibung der jeweiligen Normkonformität und eine Praktik bzw. ein Artefakt, mit dem die Konformität hergestellt werden kann.

Hinweis! Die originalen Normtexte sind aus urheberrechtlichen Gründen nicht in diesem Buch enthalten. Es existiert nur der Verweis auf die entsprechenden Paragraphen der IEC 62304.

Bei Interesse an der detaillierten Abbildung der in diesem Buch beschriebenen Praktiken und Artefakte auf die originalen Normtexte wenden Sie sich bitte an den Autor.

# § 1 - Scope

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Mit der Einhaltung einer Definition-of-Done (DoD) wird sichergestellt, dass fertiggestellte Ergebnisse weiterverarbeitet werden können. | Definition of Done |
| ABC | Im Sprint Review wird die Konsistenz geprüft zwischen der erzeugten Software, ihrer Anforderungen und weiterer Ergebnisse bzw. Inputs. | Sprint Review |
| ABC | Scrum ist ein gültiges Prozess-Framework für den Software-Life-Cycle. | Scrum |
| ABC | Im Sprint Review werden alle Ergebnisse des gesamten Software-Life-Cycles auf gegenseitige Konsistenz geprüft. | Sprint Review |
| ABC | Risiko-Management wird in erster Linie im Sprint Planning durchgeführt. | Risk Analysis and Management |

# § 4 - General requirements

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Sprint Planning wird jeder Anforderung und User Story (auf jeder E-bene) eine Sicherheitsklassifizierung zugeordnet durch die Anwendung der Risiko-Analyse. | Risk Analysis and Management |
| ABC | Im Sprint Review Meeting wird festgestellt, ob die Entwicklung des erzeugten Software-Produkts den normativen Anforderungen entspricht. | Sprint Review |
| ABC | Im Sprint Review Meeting wird festgestellt, ob das erzeugte Software-Produkt die Bedürfnisse und Anforderungen der Anwender befriedigt. | Sprint Review |
| ABC | Die Sicherheitsklassifizierung wird als Attribut jeder User Story zugeordnet, welche die Risiko-Analyse durchlaufen hat. | User Story |
| ABC | Einer User Story, der noch keine Sicherheitsklassifizierung zugeordnet wurde, wird per Default die Sicherheitsklassifizierung C zugeordnet. | User Story |
| ABC | Risiko-Management wird im Sprint Planning Meeting durchgeführt. | Risk Analysis and Management |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Modernes Software-Engineering wird umgesetzt mit der Anwendung von Clean Code und SOLID-Prinzipien, TDD, Unit Tests, Design Patterns und vielen anderen Methoden, die hier nicht vollständig aufgeführt werden können. | Clean Code / SOLID Principles |
| ABC | Durch die korrekte Anwendung objekt-orientierter Prinzipien wie "Separation of Concerns" und "Single Responsibility Principle" entstehen lose gekoppelte Architekturen. | Clean Code / SOLID Principles |
| ABC | Ein Qualitäts-Management-Prozess ist definiert und wird eingesetzt. | Quality Management Prozess |
| ABC | User Stories, welche durch Story Splitting aus einer übergeordneten User Story entstanden sind, erben die Sicherheitsklassifizierung dieser übergeordneten User Story. | User Story |

# § 5 - Software development process

# § 5.1 - Software development planning

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Das Sprint Backlog beinhaltet iterativ-inkrementell auch alle Anforderungen, welche sich aus dem Gesamtsystem an die Software ergeben. | Sprint Development Plan = Sprint Backlog |
| ABC | Die Definition-of-Done beschreibt alle durchzuführenden Aktivitäten, um eine Anforderung vollständig umzusetzen. | Definition of Done |
| ABC | Der Software-Entwicklungsplan erwähnt Scrum als eingesetztes, agiles Vorgehensmodell mit dem darin enthaltenen Sprint Review Meeting, in welchem die Forderungen aus Paragraph 4.1 umgesetzt werden: Sicherstellung der Umsetzung von Kundenanforderungen sowie regulatorischen Anforderungen. | Software Development Plan |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Der Risiko-Management-Prozess ist für jede einzelne Anforderung in der Definition-of-Done enthalten. | Definition of Done |
| BC | Konsequent angewandte Continuous Integration ist die Aktivität, durch welche eine frühzeitige Integration aller beteiligten Komponenten inklusive der dazugehörigen Integrationstests sichergestellt wird. | Continuous Integration |
| BC | Das Sprint Backlog enthält alle Anforderungen, welche innerhalb eines Sprints in das Gesamtsystem integriert und integrationsgetestet werden. | Sprint Development Plan = Sprint Backlog |
| C | In den Working Agreements eines Teams wird festgelegt, mit welchen Standards, Methoden und Tools die Umsetzung von Anforderungen in funktionierende Software geschieht. | Team Charter / Working Agreements |
| ABC | Im Sprint Backlog sind alle Produktergebnisse enthalten, welche auch im Sprint verifiziert werden müssen. | Sprint Development Plan = Sprint Backlog |
| ABC | Im Team Charter wird die Sprint-Länge festgelegt und damit die iterativen Meilensteine, zu denen umgesetzte Anforderungen verifiziert werden. | Team Charter / Working Agreements |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Eine User Story besteht neben der reinen Beschreibung auch aus definierten Akzeptanztests, welche zur Verifizierung der Umsetzung benötigt werden. | User Story |
| ABC | Die Definition-of-Done enthält alle Verizifierungsaufgaben, die für eine Anforderung durchgeführt werden müssen. | Definition of Done |
| ABC | Die Working Agreements eines Teams beschreiben das eingesetzte Versionierungssystem sowie dessen grundsätzliche Handhabung. | Team Charter / Working Agreements |
| BC | Die Definition-of-Done beinhaltet das Einchecken von umgesetzten Anforderungen in ein Versionierungssystem. | Definition of Done |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Software Development Plan werden folgende Scrum-Artefakte aufgeführt:<br><br>* Product Backlog<br><br>* Sprint Backlog<br><br>* Definition of Done<br><br>* Team Charter / Working Agreements<br><br>* Definition of Ready<br><br>* Iteration Notes (Sprint Notizen)<br><br>* Architecture Documentation | Software Development Plan |
| BC | Die Working Agreements eines Teams beschreiben alle Tools und Dinge, welche das endgültige Produkt beeinflussen könnten. | Team Charter / Working Agreements |
| ABC | Der Software-Entwicklungsplan wird iterativ-inkrementell verändert durch die entstehenden Sprint Backlogs. | Sprint Development Plan = Sprint Backlog |

# § 5.2 - Software requirements analysis

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Die Definition-of-Ready einer Anforderung beinhaltet eine Bewertung durch das Team darüber, ob diese Anforderung bereit ist, in einem Sprint umgesetzt zu werden. | Definition of Ready |
| ABC | Im Product Backlog befinden sich grobe System-Anforderungen, welche iterativ in kleinere Software-Anforderungen verfeinert werden. | Product Backlog |
| ABC | Im Product Backlog Grooming Meeting werden bestehende Anforderungen und User Stories im Product Backlog neu bewertet und aktualisiert. | Product Backlog Grooming |
| ABC | Im Sprint Planning Meeting werden bestehende Anforderungen und User Stories im Product Backlog neu bewertet und aktualisiert. | Sprint Planning |
| ABC | Die Ergebnisse und Erkenntnisse aus dem Sprint Review Meeting fließen als Feedback direkt in die Anforderungen ein und können im anschließenden Sprint Planning Meeting bedacht werden. | Sprint Review |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Ein Gebrauchstauglichkeitsprozess ist definiert und wird umgesetzt. | Usability Process |
| ABC | Daten-Definitionen und Datenbank-Anforderungen werden über entsprechende User Stories spezifiziert. | User Story |
| ABC | Eine User Story beschreibt eine funktionale Anforderung bzw. eine benötigte Fähigkeit des zu erstellenden Produkts. | User Story |
| ABC | Anforderungen an die Installation des Software-Produktes und deren Akzeptanzkriterien werden über entsprechende User Stories beschrieben. | User Story |
| ABC | Schnittstellen von und zu anderen Systemen werden über User Stories beschrieben. | User Story |
| ABC | Anforderungen aus Normen und Richtlinien werden im Product Backlog verwaltet. | Product Backlog |
| ABC | Anforderungen an Betrieb und Wartung des Systems werden mit User Stories beschrieben | User Story |
| ABC | Das Product Backlog enthält alle Sicherheits-Anforderungen (Security). | Product Backlog |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Die Eingaben und Ausgaben des gesamten Software-Systems werden über entsprechende User Stories definiert. | User Story |
| ABC | Alarme, Warnungen und andere wichtige Meldungen für den Anwender werden mit Hilfe von User Stories definiert. | User Story |
| ABC | Anforderungen an die Gebrauchstauglichkeit werden über User Stories definiert. | User Story |
| ABC | Zu erzeugende Anwender-Dokumentation wird als Anforderung im Product Backlog verwaltet. | Product Backlog |
| ABC | Anforderungen an die Wartung durch den Anwender werden mit User Stories definiert. | User Story |
| BC | Im Sprint Planning Meeting wird sichergestellt, dass alle Maßnahmen aus der Risiko-Analyse einer in diesem Sprint geplanten Anforderung in das Product Backlog als Anforderung einfließen. | Risk Analysis and Management |
| ABC | Im Sprint Planning Meeting durchlaufen alle Anforderungen eine Risiko-Analyse, um notwendige Maßnahmen abzuleiten. | Risk Analysis and Management |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Eine User Story beinhaltet alle ermittelten Maßnahmen aus ihrer eigenen Risiko-Analyse. | User Story |
| ABC | Im Sprint Planning Meeting wird sichergestellt, dass über die Anforderungen ein einheitliches Verständnis bei allen Beteiligten herrscht. | Sprint Planning |
| ABC | User Stories sind so formuliert, dass sich Test- und Akzeptanz-Kriterien ableiten lassen. | User Story |
| ABC | User Stories und darüberliegende bzw. externe Anforderungen werden mit Hilfe eindeutiger, referenzierbarer IDs versehen. | Linkable IDs |
| ABC | User Stories und darüberliegende bzw. externe Anforderungen werden mit Hilfe eindeutiger, referenzierbarer IDs versehen. | Linkable IDs |
| ABC | Im Sprint Planning Meeting wird vom Team festgestellt, ob sich Anforderungen im Product Backlog widersprechen. | Sprint Planning |
| ABC | Das Product Backlog enthält alle Anforderungen, die sich aus dem Risikomanagement-Prozess ergeben. | Product Backlog |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Mit Hilfe von definierten Akzeptanz-Tests wird sichergestellt, dass das erstellte Software-Produkt die Bedürfnisse der Anwender befriedigt und die Anforderungen korrekt umgesetzt wurden. | Acceptance Testing |

# § 5.3 - Software architectural design

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Die grundlegenden Software-Strukturen und Architektur-Entscheidungen können im Sprint Planning getroffen werden. | Sprint Planning |
| BC | Im Sprint Planning werden die Architektur und die Schnittstellen der umzusetzenden Software-Einheiten sowie der einzubindenden externen Software-Einheiten festgelegt. | Sprint Planning |
| BC | Mit der Architektur-Dokumentation ist es möglich, die Architektur zu verifizieren. | Architecture Documentation |
| ABC | Die Sicherheitsklassifizierung wird im Sprint Planning für alle umzusetzenden Software-Einheiten überdacht und gegebenenfalls neu vergeben. | Risk Analysis and Management |
| C | Im Sprint Planning werden die Architektur und die Software-Einheiten der umzusetzenden Anforderungen definiert. Die identifizierten Software-Einheiten durchlaufen die Risiko-Analyse. | Risk Analysis and Management |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Product Backlog Grooming werden die Funktionalitäten und Performance-Anforderungen an einzusetzende SOUP-Komponenten definiert. | Product Backlog Grooming |
| BC | Im Sprint Planning werden für benötigte SOUP-Komponenten die Funktionalitäten und Performance-Anforderungen festgelegt. | Sprint Planning |
| BC | Im Backlog Grooming werden die Anforderungen an Hardware und Software identifiziert, welche durch den Einsatz von SOUP notwendig werden. | Product Backlog Grooming |
| BC | Durch die Anwendung von Test-Driven Development (TDD) entsteht während der Umsetzung von Anforderungen eine funktionierende Software-Architektur. | Test-Driven Development (TDD) |
| BC | Die umgesetzte Software-Architektur kann mit Hilfe von UML-Diagrammen beschrieben und dokumentiert werden. | Architecture Documentation |

# § 5.4 - Software detailed design

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Durch die Anwendung objekt-orientierter Prinzipien wird die Aufteilung von Software-Einheiten in weitere Einheiten getrieben. | Clean Code / SOLID Principles |
| ABC | Durch Test-Driven Development (TDD) wird direkt bei der Umsetzung von Software-Einheiten entschieden, wann und wie diese weiter untergliedert werden in einzelne Einheiten. | Test-Driven Development (TDD) |
| C | Durch die Anwendung von Test-Driven Development (TDD) entsteht die Schnittstellenspezifikation aller Software-Einheiten in Form von ausführbaren Tests gegen diese Schnittstellen. | Test-Driven Development (TDD) |
| C | Durch die Anwendung von Test-Driven Development (TDD) entsteht ein "Design by Contract" über die Schnittstellenspezifikation aller Software-Einheiten in Form von ausführbaren Tests gegen diese Schnittstellen. | Test-Driven Development (TDD) |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Sprint Planning werden die umzusetzenden Anforderungen in eine Architektur überführt und diese wiederum in Software-Einheiten runtergebrochen. | Sprint Planning |
| BC | Durch den Einsatz von Test-Driven Development (TDD) entstehen Software-Einheiten und ihre Schnittstellen auf unterster Ebene, welche die darüberliegenden Software-Strukturen verfeinern. | Test-Driven Development (TDD) |
| ABC | Software-Einheiten dürfen mit Unit Tests getestet werden, ohne die Einbindung anderer Software-Einheiten. | Unit Testing |
| C | Im Sprint Review wird präsentiert, dass das umgesetzte Software-Design der Software-Architektur entspricht. | Sprint Review |

# § 5.5 - Software unit implementation and vericifation

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Durch Akzeptanztests wird sichergestellt, dass die entwickelte Software entsprechende Akzeptanzkriterien befriedigt. | Acceptance Testing |
| BC | Im Sprint Review zeigt das Team, welche Akzeptanzkriterien in Form von Testfällen die Software erfolgreich durchlaufen hat. | Sprint Review |
| BC | Durch Test-Automation wird sichergestellt, dass die Software immer wieder die definierten Akzeptanzkriterien in Form von automatisierten Tests erfolgreich durchläuft. | Test Automation |
| BC | Durch Test-getriebene Entwicklung (TTD) wird sichergestellt, dass die entwickelte Software entsprechende Akzeptanzkriterien auf unterster Ebene befriedigt. | Test-Driven Development (TDD) |
| BC | Mit Hilfe von Akzeptanz-Tests wird sichergestellt, dass eine in Software umgesetzte Anforderung abgekapselt funktioniert und damit für eine Integration in andere Strukturen bereit ist. | Acceptance Testing |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Sprint Planning Meeting wird sichergestellt, dass zu jeder Anforderung Akzeptanz-Kriterien für die Integration in andere Strukturen erstellt werden. | Sprint Planning |
| BC | Durch Test-getriebene Entwicklung wird die Software-Verifizierung sichergestellt. TDD ist ein definierter, korrekter Prozess. | Test-Driven Development (TDD) |
| C | Technische Akzeptanzkriterien, wie z.B. Sequenzen, Abläufe, Kontrollflüsse, Fehlerbehandlung, Initialisierung und Grenzwerte, werden mit Hilfe von Unit Tests definiert. | Unit Testing |
| BC | Durch Test-Automation können die Ergebnisse aller Unit-Tests als Dokumentation erzeugt werden. | Test Automation |
| BC | Die Software wird durch Unit Tests auf unterster Ebene verifiziert. Durch Test-Automation kann eine Dokumentation der Testergebnisse erzeugt werden. | Unit Testing |
| ABC | Mit Test-Driven Development (TDD) entsteht das Design des zu erstellenden Codes durch ausführbare Schnittstellen-Tests. Diese werden dann durch produktiven Code erfolgreich durchlaufen. | Test-Driven Development (TDD) |

# § 5.6 - Software integration and integration testing

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Regressionstests vorhandener Software-Versionen können durch Test-Automation jederzeit sicherstellen, dass keine Fehler im Software-System entstanden sind. | Test Automation |
| BC | Mit einer Test-Automation kann durch die Durchführung aller vorhandenen Tests eine Dokumentation der Test-Resultate erzeugt werden. | Test Automation |
| BC | Eine konsequente Zero-Bug-Tolerance führt dazu, dass alle Fehler, welche während der Software-Integration gefunden werden, schnellstmöglich behoben oder als irrelevant definiert werden. | Zero Bug Tolerance |
| BC | Fehler, welche während der Software-Integration auftauchen, werden im Sprint Backlog gepflegt, um sie schnellstmöglich zu beheben. | Sprint Development Plan = Sprint Backlog |
| BC | Im Sprint Review werden die Integrationstests auf Korrektheit geprüft. | Sprint Review |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Die Iteration Notes (Sprint Notizen) beinhalten die Liste der in der Iteration beteiligten Team-Mitglieder, wie zum Beispiel auch die Tester. | Iteration Notes (Sprint Notizen) |
| ABC | Akzeptanztests entsprechen Black-Box-Tests. | Acceptance Testing |
| ABC | Exploratives Testen entspricht Black-Box-Tests. | Exploratory and Manual Testing |
| ABC | Integrations-Tests entsprechen Black-Box-Tests bezüglich der Schnittstellen einzelner Systemkomponenten und White-Box-Tests bezüglich der System-Architektur. | Integration Testing |
| ABC | Unit Tests entsprechen Black-Box-Tests bezüglich der atomaren Schnittstellen und White-Box-Tests bezüglich der Software-Strukturen. | Unit Testing |
| BC | Die einzelnen Software-Einheiten werden kontinuierlich integriert. | Continuous Integration |
| BC | Durch Test-Automation können bestehende Testfälle jederzeit wiederholt werden. | Test Automation |
| ABC | Integrations- sowie System-Tests können mit Simulationen, Prototypen und Mocks durchgeführt werden. | Integration Testing |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Eine Anforderung oder User Story kann als erledigt betrachtet werden, wenn die entsprechenden Integrations- und System-Tests mit einer Simulation, einem Prototypen oder einem Mock durchgeführt wurden. | Definition of Done |
| BC | Die integrierten Software-Einheiten werden durch Integrationstests verifiziert. Durch Test-Automation kann eine Dokumentation der Testergebnisse erzeugt werden. | Integration Testing |
| BC | Die Software-Integration wird durch Integrationstests verifiziert. | Integration Testing |

# § 5.7 - Software system testing

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Eine konsequente Zero-Bug-Tolerance führt dazu, dass alle Fehler, welche während der Software- und System-Tests gefunden werden, schnellstmöglich behoben oder als irrelevant definiert werden. | Zero Bug Tolerance |
| BC | Fehler, welche während der System-Verifikation gefunden und nicht sofort behoben werden, werden in das Product Backlog einsortiert. | Product Backlog |
| BC | Durch Test-Automation können die Testfälle aller Software-Anforderungen jederzeit durchgeführt werden. | Test Automation |
| BC | Eine User Story beinhaltet alle Testfälle, welche notwendig sind, um die Anforderungen dieser User Story prüfen und sicherstellen zu können. | User Story |
| ABC | Bei einer Zero Bug Tolerance muss nicht jedes Problem und nicht jeder Fehler behoben werden. Falls entschieden wird, einen Fehler nicht zu beheben, muss die Begründung dokumentiert werden. Dies kann direkt im entsprechenden Backlog-Eintrag erfolgen. | Zero Bug Tolerance |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Sprint Planning erfolgt die Planung der Integrations-Tests sowie der System-Tests. | Sprint Planning |
| BC | Anforderungen können bereits vor ihrer endgültigen Umsetzung getestet und integriert werden. | Early Integration / Early Testing |
| ABC | Im Sprint Review werden die entwickelten und durchlaufenen Tests, sowie die Test-Abdeckung demonstriert. | Sprint Review |
| BC | Im Sprint Review Meeting berichtet das Team über die durchgeführten Tests aller umgesetzten Anforderungen. | Sprint Review |
| BC | Mit Hilfe einer umfassenden Test-Automation kann sichergestellt und nachgewiesen werden, dass alle Anforderungen getestet und verifiziert sind. | Test Automation |
| BC | Im Sprint Review Meeting präsentiert das Team die erzeugten und durchgeführten Tests zu den umgesetzten Anforderungen. (Es werden keine Tests erstellt, zu denen keine Anforderungen existieren.) | Sprint Review |
| BC | User Stories beinhalten neben der Definition einer Anforderung auch deren Akzeptanz- und Testkriterien. | User Story |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Sprint Review werden die umgesetzten Anforderung auf Basis ihrer Akzeptanzkriterien abgenommen und freigegeben. | Sprint Review |
| BC | Im Sprint Review wird über Verifikation und Testen berichtet. | Sprint Review |
| BC | Die Definition-of-Done stellt sicher, dass Verifikation und entsprechende Test-Prozeduren eingehalten werden. | Definition of Done |
| ABC | Durch Test-Automation kann das gesamte Software-System nach einer Änderung vollständig mit allen vorhandenen Testfällen geprüft werden. | Test Automation |
| BC | Durch Test-Automation kann das gesamte Software-System nach einer Änderung vollständig mit allen vorhandenen Testfällen geprüft werden. | Test Automation |
| BC | Notwendige Änderungen an der Software, die durch System-Tests erkannt werden, durchlaufen während des Sprint Plannings die notwendige Risikoanalyse. | Risk Analysis and Management |
| BC | Durch Test-Automation können alle vorhandenen Testfälle des Software-Systems nach Änderungen jederzeit durchgeführt werden. | Test Automation |

# § 5.8 - Software release

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Durch ein Versionierungs-System werden alle Bestandteile eines Software-Release archiviert. | Versioning System |
| BC | Nicht behobene Fehler und Abweichungen werden in den Iterations-Notizen festgehalten. | Iteration Notes (Sprint Notizen) |
| BC | Der Software-Entwicklungsplan beschreibt die Vorgensweise zur Erstellung eines Software-Release, sowie das dafür notwendige Environment. | Software Development Plan |
| ABC | In den Release Notes ist die Version des Software-Release enthalten. | Release Notes |
| BC | Durch die Definition-of-Done wird sichergestellt, dass alle Aufgaben zum Zeitpunkt eines Software-Release abgeschlossen sind. | Definition of Done |
| BC | Im Sprint Review werden die umgesetzten Anforderungen evaluiert und freigegeben, bevor sie in ein Software-Release gelangen können. | Sprint Review |
| BC | In der Definition-of-Done wird festgelegt, dass die Software bzw. jede User Story verifiziert sein muss, bevor sie ausgeliefert werden darf. | Definition of Done |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Release Planning wird entschieden, welche Maßnahmen und Aktivitäten durchzuführen sind, um eine verläßliche Auslieferung der Software zu gewährleisten. | Release Planning Meeting |
| BC | Im Product Backlog sind Maßnahmen und Aktivitäten enthalten, die eine verläßliche Auslieferung der Software gewährleisten. | Product Backlog |
| BC | Bei Anwendung der Zero Bug Tolerance durchläuft jeder Fehler und jede Abweichung die Risiko-Analyse. Nur wenn ein Fehler und eine Abweichung nicht zu einem inakzeptablen Risiko beitragen, darf entschieden werden, den Fehler und die Abweichung nicht zu beheben. | Zero Bug Tolerance |

# § 6 - Software maintenance process

# § 6.1 - Establish software maintenance plan

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Software Maintenance Plan werden alle Aktivitäten und Aufgaben des Maintenance Prozesses beschrieben. | Software Maintenance Plan |
| ABC | Im Software Maintenance Plan wird festgelegt, anhand welcher Kriterien ein Feedback als Problem/Fehler betrachtet wird. | Software Maintenance Plan |
| ABC | Im Software Maintenance Plan wird festgelegt, wie nach einem Release das Feedback eingeholt, dokumentiert, ausgewertet, umgesetzt und verfolgt wird. | Software Maintenance Plan |
| ABC | Im Software Maintenance Plan wird der Einsatz des Versionierungs-Systems beschrieben. | Software Maintenance Plan |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Product Backlog Grooming werden Probleme und Fehler von bereits releaster Software besprochen. | Product Backlog Grooming |
| ABC | Im Release Planning findet die Vorbewertung und Priorisierung statt für Probleme und Fehler von bereits releaster Software. | Release Planning Meeting |
| ABC | Im Sprint Planning finden die notwendigen Risiko-Analysen sowie die Umsetzungsplanung statt für Probleme und Fehler von bereits releaster Software. | Sprint Planning |

# § 6.2 - Problem and modification analysis

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Product Backlog Grooming Meeting werden neue und geänderte Anforderungen auf ihren Einfluß auf bestehende Produkte und Systeme untersucht. | Product Backlog Grooming |
| ABC | Der Product Owner läßt im Product Backlog Grooming Meeting neue und geänderte Anforderungen vom Team bewerten und erteilt deren Freigabe durch Aufnahme und Einsortierung in sein Product Backlog. | Product Backlog Grooming |
| ABC | Im Product Backlog Grooming Meeting werden Fehler- und Problem-Berichte bewertet und ihr Einfluß auf die Sicherheit bestehender Produkte und Systeme festgestellt, um notwendige Anforderungsänderungen im Product Backlog abzuleiten. | Product Backlog Grooming |
| ABC | Probleme und deren Konsequenzen werden über einen "Problem Communication"-Prozess an Anwender kommuniziert. | Problem Communication |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Die Release Notes einer Software beinhalten alle Änderungen an dem Software-System seit dem letzten Release. | Release Notes |
| ABC | Über den Software Maintenance Prozess erfolgt das Einholen des Feedbacks zu releaster Software innerhalb der eigenen Organisation. | Software Maintenance Process |
| ABC | Über den Software Maintenance Prozess erfolgt das Einholen des Feedbacks zu releaster Software von tatsächlichen Anwendern. | Software Maintenance Process |
| ABC | User Stories können auch Fehler- und Problemberichte darstellen, welche Abweichungen der Anforderungen und der Spezifikation beinhalten. | User Story |

# § 6.3 - Modification implementation

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Release Planning wird festgelegt, ob ein Release als vollständiges System-Release erstellt wird oder als Patch zu einem bestehenden System-Release. | Release Planning Meeting |
| ABC | Das Release Planning unterscheidet nicht, ob ein Release aus neuen Anforderungen besteht oder aus Änderungen an einem bestehenden System. | Release Planning Meeting |
| ABC | Im Product Backlog Grooming werden Änderungen an der Software analysiert, bewertet und ins Product Backlog einsortiert. Damit durchlaufen Änderungen den gleichen Entwicklungsprozess wie Anforderungen und User Stories. | Product Backlog Grooming |

# § 7 - Software risk management process

# § 7.1 - Analysis of software contributing to hazardous situations

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Sprint Planning wird die Risiko-Analyse der umzusetzenden Architektur und ihrer Software-Einheiten durchgeführt. | Risk Analysis and Management |
| BC | Im Sprint Planning werden mögliche Ursachen für Fehlverhalten von SOUP durch die Risiko-Analyse identifiziert. | Risk Analysis and Management |
| BC | Im Sprint Planning werden für fehlerhaftes Verhalten der Software mögliche Ursachen in Hardware- und Software-Fehlern analysiert. | Risk Analysis and Management |
| BC | Spätestens im Sprint Planning Meeting werden Anforderungen auf ihre korrekte und vollständige Definition geprüft. | Sprint Planning |
| BC | Die Definition-of-Ready einer Anforderung fordert deren korrekte und vollständige Definition. | Definition of Ready |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | In der Risiko-Analyse während des Sprint Planning Meetings werden alle Anforderungen auf möglichen Mißbrauch untersucht und entsprechende Maßnahmen abgeleitet. | Risk Analysis and Management |
| BC | Im Sprint Planning werden die umzusetzenden Anforderungen und deren abgeleitete Software-Einheiten durch die Risiko-Analyse auf mögliche Gefährdungen hin untersucht. | Risk Analysis and Management |
| BC | Im Sprint Planning werden die umzusetzenden Software-Einheiten durch die Risiko-Analyse auf Gefahrensituationen untersucht. | Risk Analysis and Management |
| BC | Im Sprint Planning werden Workflows und sequentielle Abfolgen von Events in der Software durch die Risiko-Analyse auf Gefahrensituationen untersucht. | Risk Analysis and Management |
| BC | Im Sprint Planning werden SOUPs und ihre bekannten Fehler/Abweichungen durch die Risiko-Analyse betrachtet. | Risk Analysis and Management |
| BC | Im Sprint Planning werden User Stories daraufhin untersucht, ob sie zu einer Gefahrensituation beitragen können, die in der Risiko-Analyse enthalten ist. | Risk Analysis and Management |

# § 7.2 - Risk control measures

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Sprint Planning wird der zu erstellenden Software-Einheit eine Sicherheitsklassifizierung zugewiesen (A, B, C), welche weitere Details im Entwicklungsprozess bestimmt. | Risk Analysis and Management |
| BC | Im Sprint Planning werden zu jeder Software-Einheit mit möglichen Gefahrensituationen entsprechende Maßnahmen identifiziert und festgehalten. | Risk Analysis and Management |
| BC | Im Sprint Planning werden Software-Einheiten, die eine Risiko-Maßnahme umsetzen, genauso behandelt wie alle anderen Anforderungen und befolgen den definierten Entwicklungsprozess. | Sprint Planning |
| BC | Im Sprint Planning Meeting wird sichergestellt, dass alle Maßnahmen aus der Risiko-Analyse einer in diesem Sprint geplanten Anforderung in das Product Backlog als Anforderung einfließen. | Risk Analysis and Management |

# § 7.3 - Verification of risk control measures

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Die Iteration Notes (Sprint Notizen) beinhalten die umgesetzten Maßnahmen aus der Risikoanalyse und deren Verifizierung im aktuellen Sprint. | Iteration Notes (Sprint Notizen) |
| BC | Mit Hilfe von verweisenden IDs kann die Traceability hergestellt werden von erkannten Gefahrensituationen über Maßnahmen über umzusetzende Software-Einheiten bis zur Verifikation dieser Software-Einheiten. | Linkable IDs |
| BC | Im Sprint Review werden umgesetzte Risiko-Maßnahmen auf neue mögliche Gefahrensituationen hin untersucht. | Sprint Review |
| BC | Im Sprint Review wird die Umsetzung aller Risiko-Maßnahmen berichtet und verifiziert. | Sprint Review |

# § 7.4 - Risk management of software changes

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| BC | Im Product Backlog Grooming Meeting durchlaufen neue und geänderte Anforderungen den Risikomanagement-Prozess, um sie mit bestehenden Risiko-Maßnahmen abzugleichen. | Product Backlog Grooming |
| ABC | Im Product Backlog Grooming Meeting durchlaufen neue und geänderte Anforderungen den Risikomanagement-Prozess, um neue Risiko-Maßnahmen abzuleiten. | Product Backlog Grooming |

# § 8 - Software configuration management process

# § 8.1 - Configuration identification

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Alle zu einem Software-System gehörenden Bestandteile und deren Versionen werden in einem Versionierungs-System abgelegt. Aus diesem Versionierungs-System kann die gültige, vollständige System-Konfiguration aller Bestandteile festgestellt werden. | Versioning System |
| ABC | Die im Software-System eingesetzten Dritt-Party-Komponenten werden in ihrem Ursprungszustand mit einem Versionierungs-System verwaltet. | Versioning System |
| ABC | Mit einem Versionierungs-System können alle zu einem Software-System gehörenden Komponenten und deren Versionen identifiziert und verwaltet werden. | Versioning System |

# § 8.2 - Change control

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Die Umsetzung von Anforderungen, welche das bestehende Software-System verändern (in einem inkrementell-iterativen Prozess also alle Anforderungen), muss vom Product Owner beauftragt sein. Dies kann mit Hilfe der Definition-of-Ready sichergestellt werden. | Definition of Ready |
| ABC | Im Sprint Planning Meeting werden vom Team alle Tasks identifiziert, welche für die Umsetzung einer Änderung (erneut) durchgeführt werden müssen. | Sprint Planning |
| ABC | Eine User Story, welche auf Basis eines Problems formuliert wird, beinhaltet den zugehörigen Problembericht, um Traceability herzustellen. | User Story |
| ABC | Durch Test-Automation kann das Software-System jederzeit nach einer Änderung verifiziert werden. | Test Automation |

# § 8.3 - Configuration status accounting

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Mit Hilfe des Versionierungs-Systems kann die Historie aller Bestandteile des Software-Systems nachvollzogen werden. | Versioning System |

# § 9 - Software problem resolution process

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Einzubindende Stakeholder werden durch einen "Problem Communication"-Prozess über die Existenz eines Problems informiert. | Problem Communication |
| ABC | Im Product Backlog Grooming werden notwendige Änderungen am System diskutiert und für die Entwicklung freigegeben. | Product Backlog Grooming |
| ABC | User Stories können Fehlermeldungen, Change Requests, sowie andere Einträge enthalten, um das Systemverhalten zu ändern oder zu korrigieren. | User Story |
| ABC | Die Ergebnisse von Fehleranalysen und -bewertungen werden in der User Story bzw. dem Fehlereintrag direkt festgehalten. | User Story |
| ABC | Im Sprint Planning Meeting werden Probleme und Fehler auf sicherheitsrelevante Aspekte mit Hilfe der Risikoanalyse untersucht. | Risk Analysis and Management |
| ABC | In User Stories können Fehler- und Problemberichte beschrieben sein. | User Story |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | In der Test-Dokumentation werden folgende Informationen abgelegt: Test-Ergebnisse, gefundene Abweichungen, getestete Software-Version, Hardware- und Software-Konfigurationen, eingesetzte Test-Tools, Test-Datum, Test-Personal | Test Documentation |
| ABC | Im Sprint Planning werden die Fehler- und Problemberichte analysiert. | Sprint Planning |
| ABC | In den Iteration Notes (Sprint Notizen) sind alle Fehlerberichte aus externen Quellen enthalten, sowie deren abgeleitete Tests zur Verifikation. | Iteration Notes (Sprint Notizen) |
| ABC | Fehlerberichte werden im Sprint Backlog verwaltet, um schnellstmöglich behoben zu werden. | Sprint Development Plan = Sprint Backlog |
| ABC | Im Sprint Review werden die Problem- und Fehlermeldungen auf Trends und Tendenzen untersucht. | Sprint Review |
| ABC | Im Sprint Review wird berichtet, welche neuen Probleme und Fehler gefunden wurden. | Sprint Review |
| ABC | Im Sprint Review wird berichtet, ob Problemlösungen dazu geführt haben, nachteilige Entwicklungen umzukehren. | Sprint Review |

| SK | Herstellung der Normkonformität | Praktik/Artefakt |
|---|---|---|
| ABC | Im Sprint Review wird berichtet, welche Änderungen (Change Requests) an der Software durchgeführt wurden. | Sprint Review |
| ABC | Im Sprint Review wird berichtet, welche Fehler und Probleme behoben wurden. | Sprint Review |

# Start der agilen Reise

Die bis hierher skizzierten Methoden stellen nicht das Ende einer agilen Einführung dar, sondern vielmehr den Start der agilen Reise, die eine Organisation fortführen muss, um die tatsächlichen Vorteile von agilen Vorgehensweisen zu erfahren.

Als Startpunkt mag es oftmals gut genug erscheinen, ein oder wenige Teams Scrum machen zu lassen. Damit ist das Ziel aber längst nicht erreicht. Echte Agilität zeichnet sich dadurch aus, dass das System, die Prozesse, die Vorgehensweisen und die Strukturen ständig den sich ändernden Rahmenbedingungen angepasst werden.

Die Zeiten sind vorbei, in denen sich eine Organisation auf einen definierten Zielzustand hinentwickeln und es dabei belassen kann. Es gibt keine Best Practices, mit deren Umsetzung eine agile Einführung oder Transformation abgeschlossen ist. Und es gibt nicht die eine, für alle Unternehmen allgemeingültige und richtige Struktur.

Es treten immer wieder Scharlatane auf, die große Versprechen abgeben für die Einführung skalierter, agiler Methoden im Gesamtunternehmenskontext. Dort werden Frameworks verkauft mit Attributen wie „sicher" und „zuverlässig", welche letztlich nur als Marketinginstrumente das Sicherheitsbedürfnis von Unternehmen ansprechen sollen.

Hüten Sie sich vor solchen Versprechungen!

Es ist für Ihren Start in die Agilität beinahe egal, für welche Methodik Sie sich entscheiden und welchen Berater oder

Coach Sie sich aussuchen. Seien Sie sich nur stets darüber bewusst, dass es nicht damit getan ist, eine feste Menge von „agilen Regeln" aufzustellen und zu beachten.

Als agiler Coach sehe ich in Unternehmen immer wieder den Einsatz von bestimmten Praktiken, die von Teams einstudiert und perfektioniert wurden. Mit dem neutralen Blick eines Externen ist mir schnell klar, dass gewisse Praktiken überhaupt keinen Nutzen für das Team oder die Organisation schaffen, und dann stelle ich die Frage, wofür der Einsatz dieser Praktik denn gut ist. Oftmals lautet die Antwort dann: „Unser damaliger agiler Berater hat uns gesagt, wir müssen das genau so machen."

Ich möchte die Qualität der Arbeit meiner Kollegen in keiner Weise in Frage stellen. Jede eingeführte Methode mag am Anfang, in der jeweiligen Situation und im damaligen Kontext die beste und sinnvollste Möglichkeit gewesen sein, mit einer Problematik umzugehen. Oft handelt es sich einfach um eine gute, erprobte Methode, um neue Teams sehr schnell agil arbeiten zu lassen. Wir müssen uns aber genau so antrainieren, regelmäßíg die dann jeweilige Situation und den Kontext neu zu betrachten und uns die Frage immer wieder zu stellen: „Wofür ist der Einsatz dieser Praktik gut?"

Und genau so möchte ich dieses Buch verstanden wissen. Es beschreibt eine von vielen möglichen Vorgehensweisen, um agile Methoden in der Medizintechnik einzuführen und zu implementieren. Es wird Ihnen jedoch nicht weiterhelfen, wenn Sie genau an den hier beschriebenen Stukturen und Vorgehensweisen festhalten. Bereiten Sie sich darauf vor, dass sich in Ihrem Team und in Ihrem Unternehmen bereits nach kurzer Zeit genau diese Strukturen und Vorgehensweisen geändert haben werden - immer getrieben durch die

Erkenntnisse der tatsächlich erlebten Erfahrungen. Das ist echte, empirische Prozesskontrolle.

Wer Sicherheit und Zuverlässigkeit in der Produktentwicklung und Projektdurchführung dadurch sucht, dass Strukturen und Vorgehensweisen definiert und fixiert werden, der möge bei traditionellen Managementmethoden bleiben. Wer jedoch in der Lage sein möchte, sein Unternehmen, seine Strukturen und seine Vorgehensweisen ständig weiter zu entwickeln und den sich verändernden Rahmenbedingungen anzupassen, der ist mit dem konsequenten Einsatz agiler Methoden auf dem richtigen Weg.

Nur wer bereit ist, die eigenen Prozesse und Strukturen auf allen Ebenen immer wieder zu hinterfragen und entsprechend zu verändern, der wird auf genau diesen Ebenen langfristig und nachhaltig erfolgreich sein.

Agilität ist nichts, was man einfach so machen kann, es ist vielmehr eine Haltung, die man an den Tag legen muss:

**„Don‘t do agile, be agile.“**
*(Mache nicht agil, sei agil.)*

# Literatur- und Quellenverzeichnis

## Normen


- IEC 62304 - Edition 1.0 - 2006-05
  Medical device software - Software life cycle processes

- IEC 62366 - Edition 1.0 - 2007-10
  Medical devices - Application of usability engineering to medical devices

- ISO 13485:2003
  Medical devices - Quality management systems - Requirements for regulatory purposes

- ISO 14971:2007
  Medical devices - Application of risk management to medical devices

## Bücher


- Beck, Kent - Test-Driven Development
  Addison-Wesley, 2002
- Cohn, Mike - User Stories Applied
  Addison-Wesley, 2004
- Crispin, Lisa; Gregory, Janet - Agile Testing
  Addison Wesley, 2008
- Derby, Esther; Larsen, Diana - Agile Retrospectives
  Pragmatic Programmers, 2006
- Duvall, Paul M.; Matyas, Steve; Glover, Andrew - Continuous Integration: Improving Software Quality and Reducing Risk
  Addison Wesley, 2007
- Freeman, Steve; Pryce, Nat - Growing Object-Oriented Software, Guided by Tests
  Addison Wesley, 2009
- Gärtner, Markus - ATDD in der Praxis
  dpunkt.verlag, 2013
- Humble, Jez; Farley, David - Continuous Delivery: Reliable Software Releaes through Build, Test, and Deployment Automation
  Addison Wesley, 2010
- Kerth, Norman - Project Retrospectives
  Computer Bookshops, 2001

- Kniberg, Henrik - Scrum and XP from the Trenches
  Lulu.com, 2007

- Kua, Patrick - The Retrospectives Handbook
  Leanpub, 2012
  ➪ leanpub.com/the-retrospective-handbook

- Löffler, Marc - Retrospektiven in der Praxis
  dpunkt.verlag, 2014

- Martin, Robert C. - Clean Code
  Prentice Hall, 2008

- Osherove, Roy - The Art of Unit Testing
  mitp, 2010

- Pichler, Roman - Agiles Produktmanagement mit Scrum:
  Erfolgreich als Product Owner arbeiten
  dpunkt.verlag, 2013

- Pichler, Roman - Scrum
  dpunkt.verlag, 2007

- Schelle, Ottmann, Pfeiffer - ProjektManager
  (beinhaltet das gesamte Projektmanagement-Kompendium der GPM/IPMA)
  GPM Deutsche Gesellschaft für Projektmanagement, 2008

- Schwaber, Ken - Agiles Projektmanagement mit Scrum
  Microsoft Press Deutschland, 2007

- Shore, James; Warden, Shane - The Art of Agile Development (Kapitel 13.2)
  O'Reilly Media, 2007

## Weblinks

- Agile Alliance - Acceptance Testing
  ➪ guide.agilealliance.org/guide/acceptance.html
- Agile Alliance - Backlog
  ➪ guide.agilealliance.org/guide/backlog.html
- Agile Alliance - Backlog Grooming
  ➪ guide.agilealliance.org/guide/grooming.html
- Agile Alliance - Continuous Integration
  ➪ guide.agilealliance.org/guide/ci.html
- Agile Alliance - Definition of Done
  ➪ guide.agilealliance.org/guide/sashimi.html
- Agile Alliance - Definition of Ready
  ➪ guide.agilealliance.org/guide/ready.html
- Agile Alliance - Exploratory Testing
  ➪ guide.agilealliance.org/guide/exploratory.html
- Agile Alliance - Heartbeat Retrospective
  ➪ guide.agilealliance.org/guide/heartbeatretro.html
- Agile Alliance - Integration
  ➪ guide.agilealliance.org/guide/integration.html

- Agile Alliance - Iteration
  ➪ guide.agilealliance.org/guide/iteration.html
- Agile Alliance - Test-Driven Development
  ➪ guide.agilealliance.org/guide/tdd.html
- Agile Alliance - Unit Testing
  ➪ guide.agilealliance.org/guide/unittest.html
- Agile Alliance - Usability Testing
  ➪ guide.agilealliance.org/guide/usability.html
- Agile Alliance - User Stories
  ➪ guide.agilealliance.org/guide/stories.html
- Amoussou, Joel - Software Architecture Documentation in Agile Projects
  ➪ efasoft.blogspot.de/2010/10/software-architecture-documentation-in.html
- Bach, James - Exploratory Testing Explained
  ➪ www.satisfice.com/articles/et-article.pdf
- Baker, Simon - Release Planning
  ➪ www.energizedwork.com/weblog/2006/04/release-planning.html
- Bradley, Charles - Tips for a Good Sprint Review
  ➪ www.scrumcrazy.com/Tips+for+a+Good+Sprint+Review
- Bunio, Terry - Agile Development Team Charter
  ➪ www.infoq.com/articles/agile-development-team-charter

- Clean Coders - Training Videos. With Personality. For Software Professionals
  ➪ www.cleancoders.com
- Cohn, Mike - Scrum Product Backlog
  ➪ www.mountaingoatsoftware.com/scrum/product-backlog
- Cohn, Mike - Sprint Backlog
  ➪ www.mountaingoatsoftware.com/scrum/sprint-backlog
- Cohn, Mike - Sprint Planning Meeting
  ➪ www.mountaingoatsoftware.com/scrum/sprint-planning-meeting
- Cohn, Mike - Sprint Review Meeting
  ➪ www.mountaingoatsoftware.com/scrum/sprint-review-meeting
- Cohn, Mike - User Stories
  ➪ www.mountaingoatsoftware.com/topics/user-stories
- Cunningham, Ward - Integration Tests
  ➪ c2.com/cgi/wiki?IntegrationTest
- Cunningham, Ward - Test Driven Development
  ➪ c2.com/cgi/wiki?TestDrivenDevelopment
- Cunningham, Ward - Unit Test
  ➪ c2.com/cgi/wiki?UnitTest
- Derby, Esther - Norms, Values, Working Agreements, Simple Rules
  ➪ agile.dzone.com/news/norms-values-working

- Druckman, Angela - How to Hold an Effective Backlog Grooming Session
  - ➪ www.scrumalliance.org/community/articles/2011/march/how-to-hold-an-effective-backlog-grooming-session
- Feggins, Reedy - Using Agile Practices to Support Software Maintenance
  - ➪ www.ibm.com/developerworks/mydeveloperworks/blogs/c914709e-8097-4537-92ef-8982fc416138/entry/march_10_2012_11_27_am8?lang=en
- Felix, Luciano - 9 Tips for Creating a Good Sprint Backlog
  - ➪ www.scrumalliance.org/community/articles/2009/march/9-tips-for-creating-a-good-sprint-backlog
- Fowler, Martin - Continuous Integration
  - ➪ www.martinfowler.com/articles/continuousIntegration.html
- Hazrati, Vikas - The Holy Grail of Zero Defect Systems
  - ➪ www.infoq.com/news/2011/02/zero-defect-systems
- Hendrickson, Elisabeth - Exploratory Testing in an Agile Context
  - ➪ de.slideshare.net/codecentric/exploratory-testing-inagileoverviewmeettheexpertselisabethhendrickson
- Huether, Derek - Simple Cheat Sheet to Sprint Planning Meeting
  - ➪ www.leadingagile.com/2012/08/simple-cheat-sheet-to-sprint-planning-meeting/

- Jeffries, Ron - Automating „All“ Tests
  - ➪ xprogramming.com/articles/automatedtesting/
- Jeffries, Ron - Automating Story Tests
  - ➪ xprogramming.com/blog/automating-story-tests/
- Johnston, Courtney - User Stories: A Beginner's Guide to Acceptance Criteria
  - ➪ www.boost.co.nz/blog/agile/acceptance-criteria/
- Luke, Monica - How Early Integration Testing Enables Agile Development
  - ➪ www.ibm.com/developerworks/rational/library/early-integration-testing-enables-agile-development/
- Martin, Robert C. - The Three Laws of TDD
  - ➪ butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd
- MindTools - Team Charters (mittlerweile kostenpflichtiger Artikel)
  - ➪ www.mindtools.com/pages/article/newTMM_95.htm
- Osherove, Roy - The Art of Unit Testing - Official Book Site
  - ➪ artofunittesting.com
- Panchal, Dhaval - What is Definition of Done (DoD)?
  - ➪ www.scrumalliance.org/community/articles/2008/september/what-is-definition-of-done-(dod)

- Pichler, Roman - Grooming the Product Backlog
  - ➪ www.romanpichler.com/blog/product-backlog/grooming-the-product-backlog/
- Pichler, Roman - The Definition of Ready
  - ➪ www.romanpichler.com/blog/product-backlog/the-definition-of-ready/
- Radcliff, Deb - Zero Tolerance for Bugs
  - ➪ sdt.bz/32548
- scrum.org - Definition of Done
  - ➪ www.scrum.org/Resources/Scrum-Glossary/Definition-of-Done
- scrum.org - Scrum Guide
  - ➪ www.scrum.org/Scrum-Guides
- scrum.org - What is Scrum?
  - ➪ www.scrum.org/Resources/What-is-Scrum/
- Spolsky, Joel - The Joel Test: 12 Steps to Better Code
  - ➪ www.joelonsoftware.com/articles/fog0000000043.html (Punkt 5)
- Wells, Don - Release Planning
  - ➪ www.extremeprogramming.org/rules/planninggame.html
- Wells, Don - Unit Tests
  - ➪ www.extremeprogramming.org/rules/unittests.html
- Wells, Don - User Stories
  - ➪ www.extremeprogramming.org/rules/userstories.html

- Westphal, Ralf; Lieser, Stefan - SOLID
  ➪ www.clean-code-developer.de/SOLID.ashx
- Wikipedia - Acceptance Testing
  ➪ en.wikipedia.org/wiki/Acceptance_testing
- Wikipedia - Clean Code
  ➪ de.wikipedia.org/wiki/Clean_Code
- Wikipedia - Continuous Integration
  ➪ en.wikipedia.org/wiki/Continuous_integration
- Wikipedia - Exploratory Testing
  ➪ en.wikipedia.org/wiki/Exploratory_testing
- Wikipedia - Integration Testing
  ➪ en.wikipedia.org/wiki/Integration_testing
- Wikipedia - ISO 13485
  ➪ de.wikipedia.org/wiki/ISO_13485
- Wikipedia - Unit Testing
  ➪ en.wikipedia.org/wiki/Unit_testing
- Zörner, Stefan - „Frag das Team!“ für Architekturdokumentation?
  ➪ www.oose.de/teamblog/team/frag-das-team-fur-architekturdokumentation/

# Feedback und Kontakt

Nach der Lektüre dieses Buches und der Anwendung seiner Inhalte auf Ihr konkretes Umfeld in Teams und Unternehmen bin ich sehr an Ihren Erfahrungen interessiert.

- Wie konnten Sie Scrum in Ihrem Umfeld umsetzen?
- An welchen Stellen hatten oder haben Sie Schwierigkeiten mit der Umsetzung?
- Was hat nicht funktioniert?
- Was hat gut funktioniert?
- Wo haben Sie Veränderungen vorgenommen?
- Mit welchen Change Management Methoden haben Sie diese Veränderungen vorgenommen?
- Welche Fragen sind unbeantwortet geblieben?

Feedback und konstruktive Kritik zu diesem Buch sind jederzeit willkommen. Sie können mich über meine Webseite www.scrum-und-die-iec62304.de kontaktieren. Ich werde versuchen, Ihre Kommentare aufzunehmen und Ihre Fragen zu beantworten.

Vielen Dank für das Lesen dieses Buches.