Agile Scrum Handbuch

Nader K. Rad & Frank Turley

Van Haren PUBLISHING

Agile Scrum Handbuch

Andere Ausgaben bei Van Haren Publishing

Van Haren Publishing (VHP) ist auf Veröffentlichungen über Best Practices, Methoden und Standards auf dem Gebiet der folgenden Domänen spezialisiert:

- IT und IT-Management;
- Enterprise-Architektur;
- Projektmanagement und
- Businessmanagement.

Diese Veröffentlichungen sind in mehreren Sprachen verfügbar und sind Teil führender Serien, wie *Best Practice, The Open Group series, Project management* und *PM series.*

Auf der Website von Van Haren Publishing ist in der **Knowledge Base** ein großes Angebot Whitepapers, Templates, kostenlose E-Books, Dozentenmaterialien etc. zu finden. Besuchen Sie www.vanharen.net.

Van Haren Publishing ist weiterhin der Herausgeber von führenden Institutionen und Unternehmen, darunter: Agile Consortium, ASL BiSL Foundation, CA, Centre Henri Tudor, Gaming Works, IACCM, IAOP, IPMA-NL, ITSqc, NAF, KNVI, PMI-NL, PON, The Open Group, The SOX Institute.

Themen pro Domäne sind:

IT und IT-Management

ABC of ICT™
ASL®
CATS CM®
CMMI®
COBIT®
e-CF
ISO 17799
ISO 20000
ISO 27001/27002
ISPL
IT4IT
IT-CMF™
IT Service CMM
ITIL®
MOF
MSF
SABSA
SIAM

Enterprise-Architektur

ArchiMate®
GEA®
Novius Architektur Methode
TOGAF®

Businessmanagement

BABOK® Guide
BiSL® und BiSL® Next
BRMBOK™
BTF
EFQM
eSCM
IACCM
ISA-95
ISO 9000/9001
OPBOK
SixSigma
SOX
SqEME®

Projektmanagement

A4-Projektmanagement
DSDM/Atern
ICB / NCB
ISO 21500
MINCE®
M_o_R®
MSP®
P3O®
PMBOK® Guide
PRINCE2®

Für eine vollständige Übersicht aller Veröffentlichungen besuchen Sie bitte unsere Website: www.vanharen.net

Agile Scrum Handbuch

Nader K. Rad

und Frank Turley

Van Haren PUBLISHING

Impressum

Titel:	Agile Scrum Handbuch
Autoren:	Nader K. Rad und Frank Turley
Deutsche Ubersetzung:	Thomas Wuttke u.a.
Herausgeber:	Van Haren Publishing, 's-Hertogenbosch - NL, www.vanharen.net
ISBN Hardcopy:	978 94 018 0475 2
ISBN EBook (pdf):	978 94 018 0476 9
ISBN ePub:	978 94 018 0477 6
Druck:	Niederländische Version: Erste Ausgabe, erste Auflage, September 2018 Deutsche Übersetzung: Erste Ausgabe, erste Auflage, September 2020
Layout und DTP:	Coco Bookmedia, Amersfoort – NL

Inhalt

1. DAS AGILE KONZEPT

Sie möchten etwas lernen, das Sie bei Ihren Projekten unterstützt? In diesem Fall sollten Sie zwei entscheidende Punkte beachten, die meist falsch verstanden werden:

1. Die weit verbreitete Aussage „*Agile* ist eine Einstellung." ist falsch. Richtig dagegen ist, dass man für *Agile* eine bestimmte Einstellung braucht. Bezeichnet man *Agile* als Einstellung, so führt dies in der Praxis lediglich dazu, dass man ohne jegliche Einschränkung und ganz nach eigenem Gutdünken arbeitet. Man nennt es einfach *Agile*, lässt Kritik an sich abprallen und strebt nicht wirklich nach Verbesserung.
2. Wer auch nur die Spur einer Ahnung hat, wie autoritäre Systeme funktionieren, weiß, dass solche Systeme stets ein Feindbild brauchen, um Schwächen im eigenen System zu überspielen und die Massen kontrollieren zu können. Die traditionelle *Wasserfall-Methode* erfüllt für viele Anwender agiler Methoden die Funktion dieses Feindbilds. Die *Wasserfall-Methode* wird zwar niemals eindeutig definiert, aber es wird impliziert, dass es sich um etablierte Projektmanagementsysteme handelt. Erfolgreiche Projekte benötigen jedoch kein Feindbild. Bitte denken Sie stets daran, dass erfolgreiche Systeme immer auf den vorhandenen Systemen aufbauen und nicht von Grund auf neu gebaut werden und dass jede Kritik, ganz gleich wie berechtigt sie sein mag, respektvoll und konstruktiv sein muss.

Sehen wir uns also an, was *Agile* wirklich bedeutet.

1.1 PROJEKTMANAGEMENTMETHODE UND PROJEKTLEBENSZYKLUS

Bei der Softwareentwicklung werden auf die eine oder andere Weise folgende Schritte entweder für einzelne Funktionen oder für die Softwarelösung insgesamt durchgeführt:

- Analyse
- Design (Planung)
- Realisierung
- Integration
- Test

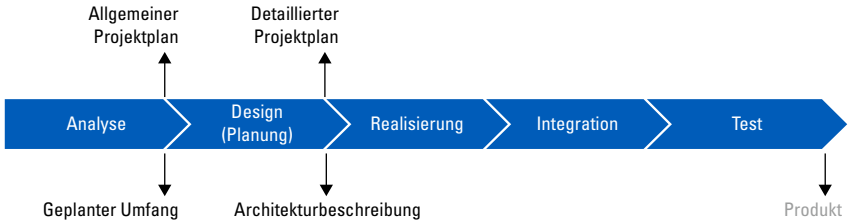
Natürlich können diese Schritte auch anders bezeichnet, in weniger Schritte zusammengefasst oder in mehrere Schritte aufgeteilt werden. Diese Schritte nennt man auch *Phasen der Softwareentwicklung*.

Die Frage ist nun, wie Sie diese Phasen organisieren und ausführen. Nehmen Sie sich kurz Zeit und überlegen Sie sich ein paar Optionen, bevor Sie den Rest des Kapitels lesen.

Nun, auf wie viele Optionen sind Sie gekommen?

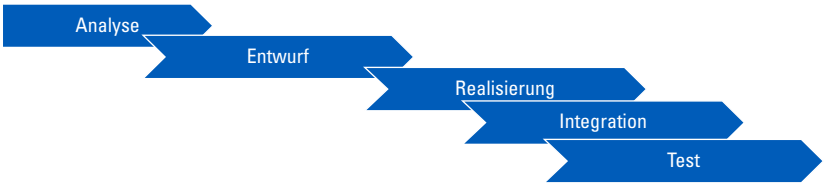
Möglicherweise können Sie sich viele verschiedene Optionen vorstellen. Im Endeffekt aber, gehören alle diese Optionen zu einer der zwei generischen Formen. Übrigens bezeichnet man diese Optionen auch als *Lebenszyklus der Softwareentwicklung*:

Ein generischer Lebenszyklus sieht etwa so aus:



In diesem Lebenszyklus wird jede Phase abgeschlossen, bevor man mit der nächsten beginnt. Mit anderen Worten, Anforderungen werden vollständig analysiert, bevor entschieden wird, was die Lösung beinhalten muss. Dann wird die Architektur der Lösung entworfen und festgelegt, wie sich die Funktionen am besten gestalten lassen. Im Anschluss daran arbeiten die Programmierer an verschiedenen Einheiten, die dann in einer Lösung zusammengeführt und als Gesamtlösung getestet werden.

Natürlich können sich die Schritte auch überlappen; so muss man beispielsweise nicht warten bis alle Einheiten vorliegen, bevor man mit der Integration und dem Testen beginnt. In einem solchen Fall könnte der Lebenszyklus wie folgt aussehen:



Dieser Lebenszyklus unterscheidet sich nicht grundlegend vom vorherigen Lebenszyklus und besteht ebenfalls aus einer sequenziellen Abfolge von Entwicklungsprozessen.

Grundvoraussetzung für diese Art von Lebenszyklus ist der anfängliche Aufwand, die Anforderungen an das zu bauende System zu verstehen. Die Spezifikation, das Design und folglich auch die Planung liegen im Vorfeld vor. Daher spricht man bei dieser Art von Lebenszyklus auch von einer **plangesteuerten Entwicklung**. Wir versuchen vorherzusagen bzw. zu prognostizieren, was wir wollen und wie sich dies realisieren lässt. Man spricht daher auch von einem **prognostizierten Lebenszyklus**.

Prognostizierte Lebenszyklen sind bei vielen Projekten, wie z. B. bei Bauvorhaben, der normale und geeignete Weg. Zuerst werden Planung und Entwurf erstellt und danach richtet sich dann die anschließende Ausführung. Für andere Projekte dagegen, ist diese Vorgehensweise nicht optimal.

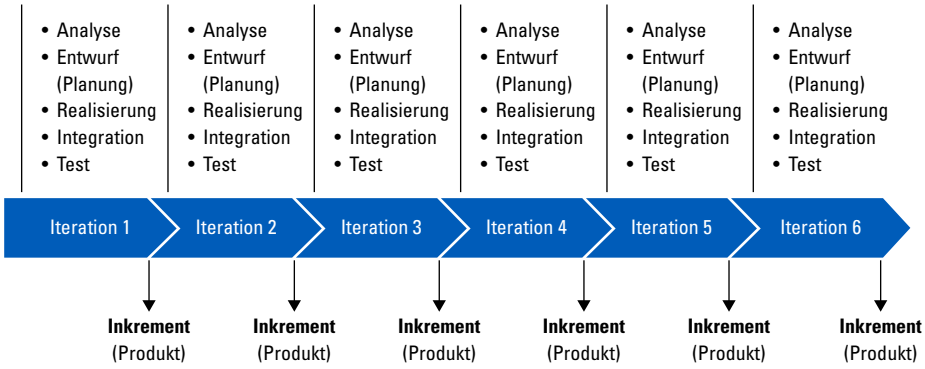
Denken Sie nur einmal an ein typisches Projekt in der IT-Entwicklung. Man investiert viel Zeit in die Spezifikation und Analyse der Anforderungen und baut dann die gesamte Entwicklung des Softwareprodukts darauf auf. Was passiert nun in der Praxis häufig? Die Kunden wünschen Änderungen. Änderungen aber sind bei diesem Lebenszyklus in dieser Phase teuer, da möglicherweise alle bisherigen Ergebnisse nochmals überarbeitet und geändert werden müssen.

Eine gängige Weisheit in der IT besagt, dass viele Kunden erst wissen, was sie wollen, wenn sie das fertige Produkt sehen. Das Produkt sehen sie aber erst gegen Ende des Projekts und damit in einer Phase, in der Änderungen mit den größten Kosten verbunden sind.

Diesem Problem können wir begegnen, indem wir den Komfort und die Struktur des *prognostizierten Lebenszyklus* opfern, zugunsten eines Lebenszyklus, bei dem das Produkt *inkrementell*, in mehreren Versionen entwickelt wird. Jede Version umfasst dabei mehr Funktionen als ihre Vorgängerversion. Diese Art der Entwicklung ist ein besonderer Luxus, den nicht alle Projekte bieten. Bei Bauvorhaben beispielsweise gibt es keine sinnvollen *Inkremente* und das Produkt kann erst nach Abschluss des Projekts genutzt werden. Die IT-Entwicklung jedoch bietet die Möglichkeit mehrerer funktionierender Versionen, bei der jede Version um weitere Funktionen ergänzt wird.

Fairerweise muss an dieser Stelle jedoch gesagt werden, dass bei einem Bauvorhaben, beispielsweise einem Bauvorhaben für ein Krankenhaus, unabhängig von der Anzahl der Änderungen, am Ende immer ein Krankenhaus und nicht zum Beispiel ein Freizeitpark entsteht. Bei der IT-Entwicklung dagegen, kann es durchaus vorkommen, dass man ein Projekt mit einem bestimmten Ziel startet und sich dieses Ziel im Laufe des Projekts massiv und grundlegend ändert.

IT-Projekte ermöglichen also eine **iterative** und/oder **inkrementelle** Bereitstellung. Diese Möglichkeit machen wir uns in folgendem Lebenszyklus zu Nutze:



Bei diesem Lebenszyklus gibt es keine echten Prognosen. Anstatt das Produkt zu prognostizieren (und sich auf diese Prognose zu verlassen), werden in kurzen Zeiträumen Produktinkremente erstellt. Jedes *Inkrement* (die neueste Version des Produkts) wird dem Kunden und den Benutzern vorgestellt. Die Aktivitäten im nächsten Zeitraum richten sich dann nach dem jeweiligen Feedback. Statt einer Prognose wird das Projekt also kontinuierlich weiterentwickelt und an das Feedback angepasst bzw. adaptiert. Daher auch die Bezeichnung **adaptiver Lebenszyklus**.

Um ein *Inkrement* zu erstellen, müssen alle Entwicklungsprozesse innerhalb einer bestimmten Zeitspanne ausgeführt werden. Im nächsten Zeitraum wird das Ganze dann wiederholt bzw. *iteriert*. Man nennt diese Methode der Entwicklung deshalb auch **iterative Entwicklung**. Die Zeitspannen, in denen diese Prozesse und Handlungen wiederholt werden, bezeichnet man auch als *Iteration*. Daneben gibt es noch eine weitere Bezeichnung, auf die wir aber später noch näher eingehen werden.

1.2 PROGNOSTIZIERTE VERSUS ADAPTIVE LEBENSZYKLEN

Prognostizierte und *adaptive Lebenszyklen* haben beide Vor- und Nachteile. Die richtige Wahl hängt von vielen Faktoren ab. Der wichtigste Faktor ist jedoch die Art des zu erstellenden Produkts.

Stellen Sie sich die zwei folgenden grundlegenden Fragen, bevor Sie den für Ihr Projekt benötigten Lebenszyklus festlegen.

1. Muss ich mich flexibel anpassen? Lautet die Antwort auf diese Frage nein, dann eignet sich ein *prognostizierter Lebenszyklus* besser! *Prognostizierte Lebenszyklen* sind einfacher und strukturierter. Ein *adaptives* System ist in den Fällen erforderlich, in denen ein gewisses Risiko besteht, dass man bei Projektstart ein Krankenhaus bauen soll und zu guter Letzt eine Art Freizeitpark dabei herauskommt.

2. Kann ich mich überhaupt flexibel anpassen? Diese Frage ist sogar noch wichtiger. Wer anpassungsfähig sein möchte, muss die Möglichkeit zur *iterativen* Entwicklung und *inkrementellen* Bereitstellung haben. Denken wir noch einmal an unser Bauprojekt: Können Sie das Gebäude *iterativ* planen? Können Sie zum Beispiel nur das Fundament planen, ohne den Rest des Gebäudes zu entwerfen bzw. ohne die Lasten zu bestimmen, die das Fundament tragen muss? Die Antwort auf diese Frage ist einfach und lautet NEIN! Eine *iterative* Entwicklung ist bei Bauprojekten nicht möglich. Das Gleiche gilt, wie bereits erwähnt, für die *inkrementelle* Bereitstellung. Ein *ada* ist daher für den Bau eines Gebäudes nicht geeignet (bitte verwechseln Sie dies nicht mit Projekten in den Bereichen Innenarchitektur und -ausstattung oder gar Renovierung, für die *adaptive* Systeme durchaus in Frage kommen können).

Was ich hier vermitteln will ist, dass *prognostiziert* oder *adaptiv* keine Frage von gut oder schlecht ist.

Machen wir eine kleine Übung. Stellen Sie sich ein IT-System vor: Um für eine sehr große Organisation mit Niederlassungen an sechs Standorten eine Netzwerkinfrastruktur zu errichten, soll für das Betriebssystem der 300 Computer einer Organisation oder eines IT-Projekts ein Update durchgeführt werden. Welcher Lebenszyklus der IT-Entwicklung ist Ihrer Meinung nach für dieses Projekt besser geeignet?

1.3 AGILE VERSUS WASSERFALL

Agile ist die populäre Bezeichnung für Systeme mit *adaptiven Lebenszyklen*. Diese Definition des Begriffs *Agile* ist viel besser als zu sagen „*Agile* ist eine Einstellung“.

Sprechen die Anhänger von agilen Methoden über *prognostizierte Lebenszyklen* (englisch: Predictive Lifecycles), verwenden sie die Bezeichnung *Wasserfall*. Dies gilt vor allem für IT-Projekte; kein Mensch würde sagen: „Dieses Gebäude wurde mit Hilfe der *Wasserfallmethode* gebaut."

Um ganz sicherzustellen, dass Sie die Terminologie beherrschen, sollten Sie sich bewusst sein, dass das Wort *Wasserfall* inzwischen praktisch ein Unwort ist und dass Sie durchaus wütend und beleidigt reagieren dürfen, wenn jemand sagt, dass Sie die *Wasserfallmethode* verwenden! Ich schlage deshalb vor, dass wir uns in diesem Buch an die offizielle Bezeichnung halten und von *prognostizierten Lebenszyklen* sprechen.

1.4 IST AGILE ETWAS NEUES?

Agile wird in der Regel als neue Methode beworben. Das Wort *Agile* für *adaptive Lebenszyklen* zu verwenden ist sicherlich neu, aber wie sieht es mit dem Lebenszyklus selbst aus?

Ich weiß nicht, wie es Ihnen geht, aber ich kann mir nur schwer vorstellen, dass die vielen Projekte und Programme in der langen Geschichte der Menschheit ganz ohne *adaptive Lebenszyklen* ausgekommen sind. Fällt Ihnen vielleicht ein Beispiel ein?

Ich kann Ihnen ein Beispiel nennen. Denken Sie einmal an eine in der Vergangenheit äußerst gängige Praxis (bzw. ein äußerst gängiges Projekt oder Programm): Krieg führen. Kann man einen Krieg mit einem *prognostizierten* Ansatz führen? Lässt sich alles von Anfang an planen und festlegen? Ganz bestimmt nicht. Zwar gibt es wahrscheinlich einen übergeordneten Plan, eher eine Art Strategie, aber den Krieg an sich müssen Sie Schlacht um Schlacht (d. h. in *Iterationen*) führen (manchmal werden möglicherweise mehrere Schlachten gleichzeitig gekämpft). Und je nach Ausgang der einzelnen Schlachten wird dann die Strategie für die weitere Kriegsführung flexibel angepasst.

Dieses Beispiel ist zwar kein angenehmes Thema, zeigt aber ganz klar, dass *adaptive* Ansätze nicht neu sein können.

Was also ist neu?

Irgendwann in der Vergangenheit wurden der so genannte wissenschaftliche Managementansatz und der Taylorismus zur Norm und jeder andere Ansatz galt als minderwertig oder sogar falsch. Da Taylorismus voll und ganz auf *prognostizierten* Systemen beruhte, dominierten diese Systeme sozusagen die ganze Welt.

Dann kam eine Zeit, in der immer mehr Projekte in der IT-Entwicklung initiiert wurden. Die *prognostizierten* Systeme waren für das Management dieser Projekte nicht wirklich die beste Lösung. Man versuchte zwar dies zu tolerieren, aber der Druck stieg und es kam zu Demonstrationen und schließlich zur Revolution! Und, wie das bei Revolutionen so üblich ist, frisst auch diese Revolution ihre Kinder, aber dazu ein andermal.

1.5 DAS AGILE MANIFEST

Bald schon kamen die ersten *adaptiven* Systeme in der IT-Entwicklung zum Einsatz und wurden allmählich strukturiert und in reproduzierbare Managementsysteme überführt. 2001 schlossen sich dann einige Pioniere auf diesem Gebiet zusammen und erarbeiteten ein Manifest.

Beginnen wir mit dem Namen. Es wird erzählt, dass nach langen Beratungen zwei Möglichkeiten übrigblieben: *Das Agile Manifest* oder *Das Adaptive Manifest*. Leider fiel die Entscheidung auf *Das Agile Manifest*. *Das Adaptive Manifest* wäre die bessere Wahl gewesen, weil dies die Art der Herangehensweise beschreibt und viele Missverständnisse verhindert hätte.

Das Agile Manifest findet sich auf der äußerst fortschrittlichen und modernen Website AgileManifest.org und lautet wie folgt:

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	

Leider wurde das Manifest selbst, nachdem es verfasst wurde, nicht mehr überarbeitet oder angepasst.

Der letzte Satz des Manifests bleibt häufig unbeachtet. Bitte lesen Sie das Manifest noch einmal durch und denken Sie dabei an den letzten Satz.

Überprüfen wir also die vier Wertaussagen:

Wertaussage 1: Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge

Ignoriert man die Bedeutung von Individuen und ihren Interaktionen, scheitert man in der Regel sehr schnell, denn schließlich werden Projekte von Menschen durchgeführt. Einige Manager glauben, sie könnten Probleme in diesen Bereichen mittels raffinierterer Systeme lösen, aber das funktioniert nur in den seltensten Fällen.

Schon viele von uns dachten naiv optimistisch, wir könnten mit tollen Werkzeugen Probleme lösen, die durch das Ignorieren menschlicher Aspekte entstanden sind, und mussten dann enttäuscht feststellen, dass dies nicht funktioniert. Manager geben nach wie vor enorme Summen für die Implementierung und Wartung solcher Werkzeuge aus und hoffen auf deren magische Wirkung. Tatsache ist jedoch, dass Werkzeuge Systeme nur unterstützen, aber nicht ersetzen können. Positiv ist zu vermerken, dass es sich bei diesen Werkzeugen um ausgeklügelte Softwareprodukte handelt, die in jahrelanger Arbeit entwickelt und instandgehalten werden und so viele Projekte und Jobs schaffen, die uns Investitionen ermöglichen, um über die Optimierung von IT-Entwicklungsprojekten nachzudenken.

Die Erwähnung der Prozesse in dieser Wertaussage ist etwas kniffelig, denn es geht nicht um Prozesse im Allgemeinen. Gemeint sind hier Prozesse, die die Notwendigkeit menschlicher Interaktionen und Komplexitäten ersetzen sollen. Ich persönlich kenne Manager, die glauben, dass sie mit einem besseren Prozess keine hochqualifizierten Experten mehr einstellen müssten. Einer der großartigen Aspekte der derzeitigen agilen Systeme ist, dass menschliche Aspekte nicht nur aufgepfropft werden bzw. man nicht nur über die Bedeutung der menschlichen Aspekte spricht, wie das bei etablierten Projektmanagementsystemen häufig der Fall ist, sondern die menschlichen Aspekte in die Prozesse integriert.

Zusammenfassend lässt sich also Folgendes konstatieren: Prozesse, die menschliche Aspekte zu ignorieren oder zu ersetzen versuchen, sind schlecht, während Prozesse, die sich dieser Aspekte annehmen und diese in das System integrieren, gut sind.

Wertaussage 2: Funktionierende Software ist wichtiger als umfassende Dokumentation

Im Gegensatz zu der vorherigen Aussage, die für alle Projekttypen gilt, handelt es sich hierbei um eine spezifische Aussage für *adaptive* Systeme. Die Kernaussage ist, dass anstelle der Dokumentation im Vorfeld eines Projekts, die festlegt, was in einem Projekt passieren muss, funktionierende Software (*Inkremente*) erstellt und dann entsprechend angepasst wird.

Wertaussage 3: Zusammenarbeit mit dem Kunden ist wichtiger als die Vertragsverhandlung

Jedes Projekt profitiert von mehr Zusammenarbeit mit dem Kunden. Bei *adaptiven* Systemen ist die Zusammenarbeit mit dem Kunden nicht nur wichtig, sondern geradezu notwendig. Der Kunde muss, während das Team kontinuierlich neue Anforderungen spezifiziert, ständig mit dem Team zusammenarbeiten, die *Inkremente* prüfen und dem Team Feedback geben. Anderenfalls ist eine Anpassung des Produkts nicht möglich.

Und Vertragsverhandlungen lieben wir natürlich alle ☺. Im Idealfall erfordert ein *Agile*-Projekt, das nach Aufwand abgerechnet wird und bei dem der Kunde nicht davon ausgeht, dass alle Anbieter Verbrecher sind, nicht allzu viele Vertragsverhandlungen. Die beiden Parteien erstellen gemeinsam ein Produkt, das einen gewissen Mehrwert bietet. Aber wir alle wissen auch, dass der Idealfall sehr selten ist und dass Kunden in der Regel doch Verträge mit einem festen Umfang und Festpreisen wünschen. Diese Art von Verträgen steht in grundlegendem Widerspruch zu den *adaptiven* Methoden und führt daher, ähnlich wie bei *prognostizierten* Projekten, zu endlosen Vertragsverhandlungen.

Wertaussage 4: Reagieren auf Veränderungen ist wichtiger als das Befolgen eines Plans

Diese Wertaussage gilt, ähnlich wie die Wertaussage 2, spezifisch für *adaptive* Systeme. Bei *Agile* gibt es im Vorfeld keinen Plan, der den Weg weist und gleichzeitig festlegt. Agile Systeme leben von der Anpassung. Letzteres bezeichnet man bei *Agile* normalerweise als *Change (Änderung)*, möglicherweise, weil den Kunden die Vorstellung gefällt, jederzeit alles ändern zu können. Streng genommen jedoch spricht man nur dann von *Change*, wenn etwas von der ursprünglichen Baseline-Planung abweicht. Bei *adaptiven* Systemen gibt es aber gar keine Baseline-Planung. Technisch gesehen handelt es sich um einen kontinuierlichen Fluss neuer Ideen. Bleiben wir jedoch ruhig bei der Bezeichnung *Change (Änderung)*, wenn das für unsere Kunden wichtig ist.

- Ihre Prüfung wird sehr wahrscheinlich auch Fragen zum Agilen Manifest enthalten. Lesen Sie daher das *Agile Manifest* mehrmals durch und prägen Sie sich die vier Wertaussagen gut ein.

1.6 DIE AGILEN PRINZIPIEN

Das *Agile Manifest* ist angenehm kurz. Da die Autoren die neu benannten Ideen der *Agile*-Methode jedoch näher ausführen wollten, haben sie ergänzend folgende zwölf Prinzipien formuliert:

Prinzip 1: Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufriedenzustellen.

Wir sind wirtschaftlich tätig und brauchen zufriedene Kunden. Das ist offensichtlich. Heutzutage nimmt man gerne die Benutzerzufriedenheit als ultimativen Maßstab, da sie Gewinne für den Kunden generiert und der Kunde so früher oder später nachhaltig zufriedengestellt wird. Klingt das zu idealistisch?

Wie also können wir die Kunden zufriedenstellen? Dies tun wir über die von uns entwickelte Software, die das Potenzial hat, Wert (z. B. Geld) zu generieren. Liefern wir früh und kontinuierlich aus, können wir die Wertschöpfung beschleunigen. Darüber hinaus können wir Anpassungen vornehmen und so ein Produkt nach den Wünschen des Markts entwickeln, für das der Markt auch bereit ist zu bezahlen, und nicht ein Produkt, von dem wir nur annehmen, dass es auf dem Markt nachgefragt wird.

Prinzip 2: Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.

Schlagen wir doch mehr Marketingkapital aus den Begriffen *Änderung* oder *Change*. Kunden lieben diese Begriffe. ☺

Prinzip 3: Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.

Erinnern Sie sich an die *Iterationen*, über die wir gesprochen haben – diese Zeiträume, in denen wir Entwicklungsprozesse wiederholen, um Produktinkremente zu erzeugen? Diese *Inkremente* sollen laut diesem Grundsatz maximal ein paar Monate dauern. Das Maximum bei Scrum ist ein Monat. Über diesen Punkt werden wir in diesem Buch noch häufig sprechen.

Der Vorschlag, funktionierende Software innerhalb weniger Wochen auszuliefern, also die Idee, innerhalb weniger Wochen über ein neues Produktinkrement zu verfügen, wurde seinerzeit mit Gelächter aufgenommen. Inzwischen gibt es aber sogar Projekte mit kürzeren *Iterationen*.

Prinzip 4: Fachexperten und Entwickler müssen während des Projekts täglich zusammenarbeiten.

Dies steht im Widerspruch zu der Idee, die Fachexperten (ob Kunden oder andere Experten) von den technischen Mitarbeitern zu trennen, die bei Projekten nach wie vor für Probleme sorgt. Techniker und Fachexperten betrachten die jeweils andere Partei manchmal als Gegner, was für das Projekt alles andere als optimal ist.

Darüber hinaus sind Produktanpassungen nur möglich, wenn die Fachexperten kontinuierlich verfügbar sind. Denken Sie nur einmal an die kontinuierliche Analyse von neuen Funktionen und das Prüfen von fertiggestellten Einheiten. Außerdem macht es mehr Spaß, den erfolgreichen Abschluss einer *Iteration* gemeinsam mit vielen Mitstreitern zu feiern.

Prinzip 5: Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.

Wir werden schon bald über weitere Aspekte *adaptiver Systeme* sprechen. Einer dieser Aspekt ist, dass die Projektmitarbeiter motiviert, eigenverantwortlich und selbstorganisiert arbeiten. Motiviertes, eigenverantwortliches und selbstorganisiertes Arbeiten ist nicht nur wichtig, weil es prinzipiell eine gute Sache ist, sondern vor allem, weil *adaptive Lebenszyklen* diese Art der Arbeit brauchen. Sie können sich ja, während wir die weiteren Prinzipien besprechen, schon einmal überlegen, warum das so ist.

Prinzip 6: Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist das Gespräch von Angesicht zu Angesicht.

Persönliche Gespräche sind effizienter und effektiver als E-Mails! Bitte beachten Sie, dass dies das prüfungsrelevanteste Prinzip aller Zeiten ist.

Auf dieses Prinzip werden wir nochmals zuruckkommen in Abschnitt 2.3.8, wenn wir über die so genannte *osmotische Kommunikation* sprechen.

Prinzip 7: Funktionierende Software ist das wichtigste Fortschrittsmaß.

Bei den meisten Projekten wird das Falsche gemessen. Dies ist ein grundlegendes Problem, denn das, was man misst, ist das, was man bekommt. Wenn Sie messen, wie viele Codezeilen erstellt werden, erhalten Sie mehr Codezeilen. Wenn Sie messen, wie beschäftigt die Entwickler sind, bekommen Sie stärker beschäftigte Entwickler. Wenn Sie die Velocity (Geschwindigkeit) messen, wird die Velocity gesteigert. Die ist aber nicht das Ziel.

Was soll also gemessen werden?

Das wichtigste Maß ist die Wertschöpfung. Diese ist jedoch schwer zu messen. Der nächstbeste Maßstab ist daher die funktionierende Software, die die Fähigkeit zur Wertschöpfung begründet.

Prinzip 8: Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

Keine übermäßigen Überstunden vor Releases. Ziel ist die langfristige Wertmaximierung nicht ein kurzfristiger Nutzen, der möglicherweise Produktivitäts- und Qualitätsverluste zur Folge hat.

Prinzip 9: Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

Bei *adaptiven Systemen* besteht die Gefahr eines schlechten Designs, da dieses nach und nach, parallel zum Projektverlauf und nicht vorab umgesetzt wird. Um dieses Problem zu lösen, stehen bestimmte Praktiken zur Verfügung.

Prinzip 10: Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.

Dies klingt relativ kompliziert, bedeutet aber ganz einfach, dass mehr Funktionen nicht automatisch besser sind.

In der Einfachheit liegt die Kunst. Wir sollten versuchen, uns bei einer Lösung auf die wirklich nützlichen Funktionen zu beschränken. Dies spart Zeit und Geld (das für andere Projekte eingesetzt werden kann) und senkt die Wartungskosten.

Prinzip 11: Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

Selbstorganisation bedeutet in diesem Zusammenhang, dass das Team aus motiviert, eigenverantwortlich und selbstbestimmt arbeitenden Mitgliedern besteht, die sich aktiv an Entscheidungen beteiligen. Dies ist in der Regel eine gute Idee.

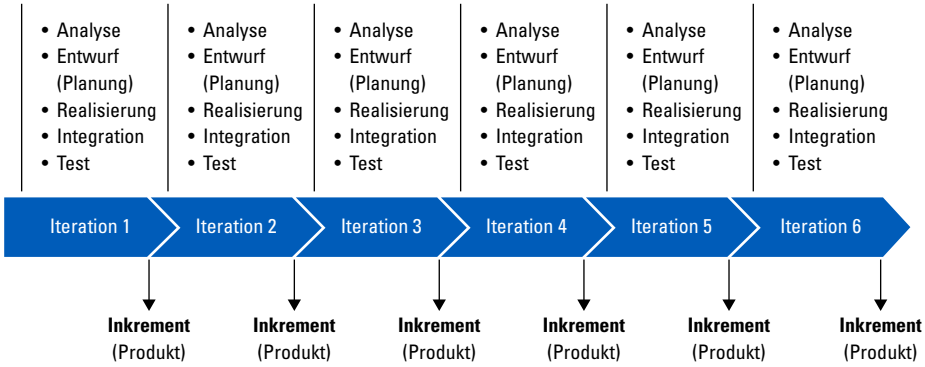
Prinzip 12: In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann, und passt sein Verhalten entsprechend an.

Wir müssen akzeptieren, dass unsere Arbeitsweise nicht perfekt ist, haben aber die Möglichkeit, sie schrittweise zu verbessern. Und bitte, nehmen Sie sich hier kein Beispiel an der Verbesserung des *Agilen Manifests* oder der Prinzipien von *Agile*. Lassen Sie in diesem Fall ausnahmsweise die Worte, nicht die Taten sprechen.

- Okay, damit haben wir die Prinzipien von *Agile* abgeschlossen. Bitte denken Sie daran, dass diese sehr häufig in Prüfungen abgefragt werden.

1.7 PRAKTISCHE ÜBERLEGUNGEN ZU ADAPTIVEN LEBENSZYKLEN

Erinnern Sie sich, wie der *adaptive Lebenszyklus* funktioniert? Hier noch mal das Bild, damit das Buch auch umfangreich genug wird:



Bei diesem Thema gibt es ein paar Punkte, über die wir sprechen müssen. Zunächst wählen wir für jede *Iteration* eine Reihe von Funktionen aus. Unser Ziel ist, bis zum Ende der *Iteration* ein funktionierendes *Inkrement* zu erstellen, das hoffentlich alle Funktionen enthält. Jetzt ist Ihre Meinung gefragt. Was halten Sie für besser, eine *Iteration* mit festem Umfang oder eine *Iteration* mit fester Dauer?

1.7.1 Iterationen mit festem Umfang versus Iterationen mit fester Dauer

Theoretisch ist beides möglich, aber in der Praxis haben sich *Iterationen* mit fester Dauer bewährt. Die Gründe hierfür sind, dass man bei *Iterationen* mit festem Umfang…

- …mehr Zeit benötigt, um den Umfang abzuschließen. Dadurch erhält man weniger Rückmeldungen und somit weniger Möglichkeiten zur Anpassung des Produkts.
- … zu viel Zeit für die einzelnen Funktionen aufwendet und zu viel Schnickschnack hinzufügt. Bei einer festen Dauer muss man sich stets auf die wichtigsten (wertschöpfenden) Punkte konzentrieren.

Aus diesem Grund nutzen fast alle *Agile*-Methoden *Iterationen* mit fester Dauer (so genannte *Timeboxes*) und bestehen meist darauf, diese einzuhalten. Eine **Timebox** ist eine Zeitspanne mit einer maximalen (oder festen) Zeitdauer, die unter keinen Umständen verlängert werden darf (verlängert man sie einmal, dann wird dies gerne zur Gewohnheit).

1.7.2 Länge der Iterationen

Wir haben also festgestellt, dass *Iterationen* zeitlich beschränkt, d. h. *timeboxed*, sind. Welche Dauer sollte eine solche *Iteration* demnach haben? In den *Agile*-Prinzipien wurde dies bereits erwähnt. Erinnern Sie sich?

Ausgehend von diesen Prinzipien sollte eine *Iteration* maximal zwei Monate dauern. Bei Scrum beträgt das Maximum einen Monat.

Diese Zeit reicht aus, um einige neue Funktionen zu entwickeln und diese Kunden und Benutzern vorzustellen, um Feedback zu erzeugen. Ist die Dauer der *Timebox* zu lang, besteht die Gefahr, dass wir am Ende zu wenig Feedback haben.

1.7.3 Gleiche oder unterschiedliche Dauer für Iterationen?

Was meinen Sie? Ist es besser, wenn alle *Iterationen* die gleiche Dauer haben oder sollte die Dauer flexibel sein?

Iterationen von gleicher Dauer sind disziplinierter und haben den Vorteil, dass man sich nicht ständig für eine neue Dauer entscheiden muss.

Bei Scrum müssen alle *Iterationen* die gleiche Länge haben, wobei diese Länge flexibel angepasst werden kann. Sie können ein Projekt beispielsweise mit zweiwöchigen *Sprints* (so nennt man bei Scrum eine *Iteration*) starten. Stellen Sie nach einer Weile fest, dass Sie in zwei Wochen nicht genügend Funktionen erstellen können, dann können Sie Ihre *Sprints* künftig auf drei Wochen verlängern. Das geht! Was bei Scrum dagegen nicht geht ist, dass Sie vor jedem *Sprint* fragen: „Okay, wie lange ist unser *Sprint* dieses Mal?“.

Nicht alle Methoden gehen hier gleich vor. Bei DSDM, einer anderen agilen Methode, planen Sie die Dauer der *Timeboxes* basierend auf ihrem Umfang (bei DSDM werden *Iterationen* als *Timeboxes* bezeichnet).

1.7.4 Was ist, wenn einige Funktionen nicht fertig werden?

Wir wählen also ein paar Funktionen für eine *zeitlich beschränkte (timeboxed) Iteration* aus. Was passiert nun, wenn wir es nicht schaffen, bis zum Ende der *Iteration* alles fertigzustellen?

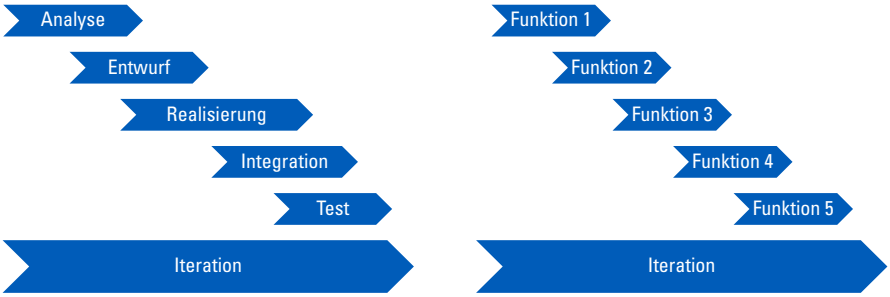
Das ist überhaupt kein Problem! Unser Ziel ist die Erstellung eines Software-*Inkrements*, um Feedback für unsere Anpassungen zu erzeugen und später, bei der Einführung in die Produktivumgebung, maximale Wertschöpfung zu ermöglichen. Unser Ziel ist es NICHT, möglichst viele Funktionen zu entwickeln.

Diese Aussage stützt sich auf drei Prinzipien von Agile. Wissen Sie welche?

Antwort: 1, 7, 10

1.7.5 Was passiert in den Iterationen?

Wie sollte Ihrer Meinung nach der Entwicklungsprozess in den einzelnen *Iterationen* ablaufen? Hier gibt es zwei Möglichkeiten:



Entweder Sie nehmen, wie auf der linken Seite dargestellt, alle Funktionen einer *Iteration* als Gesamtheit und führen für diese Gesamtheit einen Entwicklungsprozess nach dem anderen durch (in einer Art Mini-*Wasserfall*).

Oder Sie konzentrieren sich auf eine oder mehrere Funktionen und führen für diese Funktion(en), wie auf der rechten Seite dargestellt, alle Entwicklungsprozesse durch. Dies ist die bessere Option. Können Sie erklären, warum?

Da wir uns für *zeitlich beschränkte (timeboxed) Iterationen* entschieden haben, besteht immer die Gefahr, dass wir nicht mit allem fertig werden. In einem solchen Fall hätten Sie bei der Option mit dem Mini-*Wasserfall* keine einzige neue Funktion vollständig vorliegen und könnten somit auch kein Feedback erzeugen. Bei der

anderen Herangehensweise dagegen, haben Sie immer ein paar Funktionen, die Sie Ihrem Kunden präsentieren und für die nächste *Iteration* anpassen können.

1.7.6 Empowerment[1]

Einige Entscheidungen werden von Führungskräften, andere vom Projektteam getroffen. In welchem Verhältnis dies geschieht wird teilweise durch das Vorgehen im Rahmen des Entwicklungsprojekts vorgegeben.

Wann müssen die meisten nichttechnischen Entscheidungen getroffen werden?

Die meisten nicht-technischen Entscheidungen fallen in der Analysephase, in der wir versuchen, die Anforderungen festzulegen, und in der finalen Testphase, in der wir prüfen, ob die Funktionen unsere Anforderungen erfüllen. Blättern Sie bitte noch einmal an den Anfang des Kapitels und sehen Sie sich noch einmal an, wie sich die Entscheidungen bei *prognostizierten* und *adaptiven* Ansätzen verteilen.

Im *prognostizierten Lebenszyklus* konzentrieren sich viele Entscheidungen auf den Anfang und das Ende des Lebenszyklus, so dass wir die meisten Entscheidungen den Führungskräften überlassen können. Wie sieht es aber bei *adaptiven* Systemen aus? Bei diesen Systemen müssen Entscheidungen über den gesamten Lebenszyklus verteilt, fast täglich getroffen werden. Würde man auch hier die Entscheidungen immer an die Führungskräfte eskalieren, würde das Projekt wahrscheinlich niemals fertig werden.

Deshalb braucht man bei *adaptiven Lebenszyklen* motivierte, zur Selbstverantwortung und Selbstbestimmung ermächtigte und befähigte Teammitglieder, die die meisten Entscheidungen selbst treffen können und diese nicht an Führungskräfte eskalieren müssen.

1.8 IST AGILE NUR ETWAS FÜR IT-PROJEKTE?

Wie Sie bemerkt haben, gehe ich davon aus, dass alle agilen Projekte im Bereich der IT-Entwicklung angesiedelt sind. Sie haben auch festgestellt, dass das *Agile Manifest* und die Prinzipien von *Agile* von Software sprechen. Beschränken sich agile Methoden demnach auf Projekte in der IT-Entwicklung?

Meine Antwort auf diese Frage lautet wie folgt:

- Ich persönlich glaube, dass *Agile* sich nicht für alle Projekte eignet. Diese Meinung wird nicht von allen geteilt. Es gibt Experten (in der Regel Experten, die keine

1 Empowerment steht für eine Unternehmenskultur, bei der die Mitarbeiter zur Selbstverantwortung und Selbstbestimmung ermächtigt und befähigt werden.

Erfahrung mit anderen Projekttypen haben), die der Meinung sind, *Agile* eigne sich für alle Projekttypen.
- *Agile* lässt sich mit Abstand am besten in der IT-Entwicklung anwenden.
- Es mag möglich sein, *Agile* bei einigen anderen Projekttypen anzuwenden, aber meiner Erfahrung nach ist dies mit einigen Schwierigkeiten verbunden. Möglicherweise lassen sich jedoch bei einem Großprojekt, vorausgesetzt die Rahmenbedingungen stimmen, einige Teilprojekte erfolgreich mit *Agile* bearbeiten.
- Was ich bisher gesagt habe, betrifft Projekte. Wie aber verhält es sich bei Programmen[2]? Programme sollten generell und ohne Ausnahme mit *adaptiven* Methoden ausgeführt werden, wobei sich die *adaptiven* Methoden, die man auf ein Programm anwenden kann, von den *adaptiven* Methoden unterscheiden, die man für Projekte verwendet (z. B. Scrum).

Diese Ausführungen sind etwas für die Praxis und damit Sie etwas fürs Leben lernen. Wird Ihnen diese Frage jedoch in einer Prüfung zum Thema *Agile* gestellt, dann lautet die richtige Antwort: „*Agile* (oder die jeweilige agile Methode, die Gegenstand der Prüfung ist) ist *nicht* auf IT-Projekte beschränkt", wahrscheinlich sollten Sie sogar antworten: „Das Einzige, was unsere Welt noch retten kann, ist *Agile*".

1.9 IST AGILE SCHNELLER?

Der Begriff *Agile* impliziert, dass diese Methoden schneller sind. Diese Hypothese zu belegen oder zu widerlegen ist außerordentlich schwierig. Die Schnelligkeit agiler Projekte wird durch zwei Faktoren begünstigt:
- *Change (Änderung)*: Die Durchführung von Änderungen in einem *prognostizierten* Projekt ist bei weitem zeit- und kostenintensiver als bei einem agilen Projekt.
- Umfang: Bei *prognostizierten* Projekten wird der Umfang im Vorfeld festgelegt. Die Verantwortlichen neigen dazu, bei der Festlegung des Umfangs äußerst kreativ zu sein. Häufig werden Funktionen hinzugefügt, die nie oder nur selten verwendet werden. Dies gilt, einigen Studien zufolge, bei einem durchschnittlichen Software-Produkt für mehr als die Hälfte der Funktionen. Bei agilen Projekten entwickelt sich der Umfang im Laufe des Projekts. Dadurch verringert sich dieses Problem zu einem gewissen Grad, was wiederum für ein einfacheres und kürzeres Projekt sorgt.

2 Projekt: „ein Vorhaben, das im Wesentlichen durch die Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z. B.: Zielvorgabe, zeitliche, finanzielle, personelle oder andere Bedingungen, Abgrenzungen gegenüber anderen Vorhaben und projektspezifische Organisation." (Definition nach DIN 69901)

Programm: „ein großes und komplexes Vorhaben, das aus mehreren inhaltlich zusammengehörenden Projekten besteht." (Definition frei nach https://erfolgreich-projekte-leiten.de/projekt-programm-multiprojekt-projektportfoliomanagement/)

2. SCRUM

Agile ist ein übergeordnetes Konzept und steht für die Anwendung eines *adaptiven Lebenszyklus*. Für die Projektdurchführung benötigen wir jedoch eine praktische Vorgehensweise, die auf diesem Konzept beruht und hier kommen die Methodiken (und Frameworks) ins Spiel.

Zwar gibt es einige agile Methoden, aber nur eine dieser Methoden ist weitverbreitet und bekannt: Scrum. Tatsächlich dominiert Scrum den Markt so stark, dass es schwerfällt über *Agile* zu sprechen, ohne sich konkret auf Scrum zu beziehen.

Eine Beschreibung des zentralen Rahmenwerks von Scrum findet sich im Scrum Guide (scrumguides.org). Dieser bietet eine kurze Erläuterung des Rahmenwerks, ist aber bei weitem nicht so einfach zu lesen wie dieses Buch ☺.

- Etwa 80 % der Fragen in Ihrer Prüfung beziehen sich direkt oder indirekt auf Scrum.

2.1 METHODIK VERSUS FRAMEWORK

Scrum ist keine Methodik, es ist ein *Framework* (Rahmenwerk). Der Unterschied zwischen einer Methodik und einem *Framework* ist nicht einfach zu erklären. Allgemein lässt sich jedoch sagen, dass es sich bei einer Methodik eher um ein ausgefeiltes Verfahren für komplexe Situationen handelt, das bei weniger komplexen Projekten entsprechend vereinfacht werden muss. Ein *Framework* dagegen beschreibt das absolute Minimum, das für ein Projekt notwendig ist und muss bei komplexeren Projekten möglicherweise ergänzt werden.

Ich habe unter uns gesagt den Eindruck, dass viele das Wort Methodik nicht mögen, weil es traditionell und nach *Wasserfallmethode* klingt, aber vielleicht irre ich mich da auch.