Industrieseminar

Agiles Requirements Engineering

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang SWT Dual
erstellte Industrieseminar

von
Dennis Schöneborn
Matr.-Nr. 7096122

Dortmund, 23. Juni 2017

Inhaltsverzeichnis

1. Einleitung

In der Softwareentwicklung werden häufig neue Herangehensweisen und Vorgehensmodelle eingeführt oder ausprobiert. So wurden in den letzten Jahrzehnten eine große Anzahl an neuen Methoden entwickelt und fast genauso viele wurden wieder aus der Industrie und Wissenschaft verbannt. Nur wenige davon werden bis heute so oder so ähnlich eingesetzt. Dies kann durchaus daran liegen, dass die klassischen Methoden der Softwareentwicklung lediglich zum Vortäuschen eines Vorhandenseins des Managements in der Softwareentwicklung verwendet werden.

Zu dem selben Schluss kommen sowohl Studien([NA99], [Joh02]) als auch Parnas und Clements, die bereits in den 80er Jahren eine Veröffentlichung[PC86] geschrieben haben, in denen sie dieses Thema konkretisieren. Sie kritisieren in diesem Schreiben die klassischen Methoden zu imaginär und beschlossen, dass diese den Entwicklern eine zu „normative Führung in utopischen Entwicklungsumgebungen“ beiseite stellen. Daher ist es nicht verwunderlich dass viele Softwareentwickler neuen Ansätzen sehr skeptisch gegenüberstehen. Aus dieser Masse entwickelten sich die Vertreter der „Agilen Softwareentwicklungsmethoden“ und sind aktuell dabei die klassische Softwareentwicklung durch die agile zu ersetzen.

1.1 Zielsetzung

Da die Agilität immer mehr Anklang in der Softwareentwicklung findet (siehe Abbildung 1.1) und es immer wieder unstimmigkeiten zwischen Verfechtern beider Seiten, agiler und klassischer Methoden, gibt, soll diese Ausarbeitung einen genaueren Blick

auf das agile Requirements Engineering richten und die Unterschiede zwischen den traditionellen und den agilen Methoden herausarbeiten. Es soll kein bewertendes Verfahren durchgeführt werden und dem Leser hinterher zu einer der beiden Methodiken hinlenken. Vielmehr werden die Unterschide beider Herangehensweisen aufgezeigt.

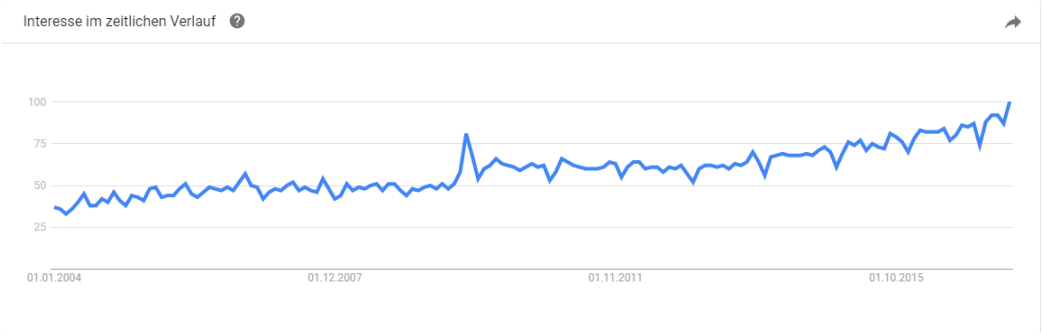

Abbildung 1.1: Suchhäufigkeit von „Agile Softwareentwicklung“ in Google

1.2 Vorgehensweise

Um die traditionelle oder auch klassische Herangehensweise des Requirements Engineerings mit der agilen Vergleichen zu können, wird das klassiche, sowie das agile Projektmanagement in ihren Grundzügen erläutert. Dies soll als Hilfe zur Verständniss- und Eingliederungshilfe beitragen.

Sobald beide Seiten des Projektmanagements der Softwareentwicklung erläutert wurden, wird direkt in das agile Requirements Engineering hinein gesprungen. Hier werden mehrere Methoden und Einsatzgebiete genannt und die Vor- bzw. Nachteile jener erläutert. Da die Erläuterung der klassischen Methoden des Requirements Engineering zu Umfangreich für diese Arbeit sind, wird davon ausgegangen, dass typische Methoden, wie die Last- und Pflichtenheft Erstellung bekannt sind.

Zum Schluss werden im Fazit die erlangten Erkentnisse beschrieben und möglichst objektiv zusammengefasst welche Methoden des Requirements Engineering sich für welche Einsatzgebiete am besten eignen.

2. Agilität in der Softwareentwicklung

Die agile Softwareentwicklung findet immer mehr Anhänger in der Wirtschaft. Mittlerweile ist diese sogar schon mehr die Regel als eine Ausnahme. Jedoch arbeiten immer noch eine nicht unerhebliche Menge an Unternehmen mit den klassichen Projektmanagement Methoden, denn agil ist nicht immer gleich besser.

Die bereits etwas in die Jahre gekommenen Methoden haben somit durchaus in bestimmten Einsatzgebieten ihre Berechtigung. Im Folgenden werden die klassichen und agilen Methoden des Projektmanagements kurz vorgestellt und die Einsatzbereiche, in denen die jeweiligen Methoden sinnvoll sind erläutert.

2.1 Klassische Softwareentwicklung

Die klassische Softwareentwicklung und ihre Methoden kommen ursprünglich aus der Ingineurwissenschaft und wurden eingeführt, da vor allem große Softwareprojekte scheiterten. In der Inginierwissenschaft beruht das Projektmanagement zumeist auf eine sequenzielle Abarbeitung definierter Projektphasen.

Das wohl bekannteste sequenzielle Modell ist das Wasserfallmodell (siehe Abbildung 2.1). Jede Phase erbringt ein Output, auf welches in der nächsten Phase aufgebaut wird. Eine Phase kann bei diesem Modell erst gestartet werden, wenn die vorherige Phase vollständig abgeschlossen ist. Das Wasserfallmodell erweitert das Stagewise Model durch Rückkopplungsschleifen. So ist es bei Fehlern möglich zurück in die vorherige Phase zu springen. Dies macht das Modell ein wenig flexibler und Fehler verursachen durch frühes beheben weniger wirtschaftlichen Schaden. Durch die klare Strukturierung

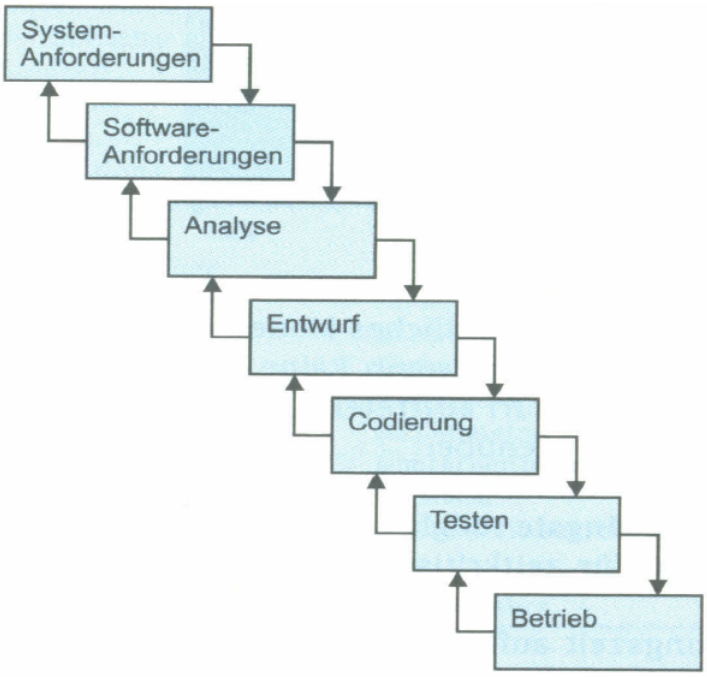

Abbildung 2.1: Wasserfallmodell [Bal08]

ist diese Methode, sowie viele weitere Methoden der klassischen Softwareentwicklung, besondern gut für Projekte mit klar definierten Anforderungen geeignet. Dies ist z.B. bei vorhandenem Lasten- und Pflichtenheft der Fall. Genau hier steckt jedoch auch die größte Schwäche vieler klassischer Methoden. Bei ungenauen oder sogar wechselnden Anforderungen, sowie bei Projekten, bei denen die Anforderungen im Laufe des Projektes erabreitet werden, sind klassische Methoden mehr als hinderlich. Ist man bereits über die Planungsphase hinaus und Anforderungen ändern sich, so muss entweder das Vorgehensmodell gelockert werden, was jedoch in den meisten Fällen die Vorteile dessen zerstört, oder die Änderungen können erst im nächsten Release implementiert werden. Im letzten Fall erhöht das die Kosten des Projektes erheblich und ist somit weder vom Projektgeber noch vom Projektnehmer erwünscht.

So werden zumeist Hybridlösungen erschaffen, welche versuchen klassiche Softwareentwicklungsmethoden mit einem agilen Anteil zu füllen. Wie bereits erwähnt verliert hier jedoch die klassiche Softwareentwicklung ihren größen Vorteil. Die Berechenbar- und Planbarkeit.

2.2 Agile Softwareentwicklung

Die agile Softwareentwicklung hat einen anderen Ansatz gesucht und gefunden. Doch was für ein Ansatz ist das und wie wird eine Agile Softwareentwicklung überhaupt definiert?

Christian Fonden hat sich diese Frage in einer Ausarbeitung von 2008 gestellt und versucht eine präzise Definition für eine Agile Softwareentwicklungsmethode niederzuschreiben. „Dies ist der Fall, falls ein Entwicklungsprozess inkrementell, also mit kurzen Release-Zyklen, kooperativ, also in der Situation, dass der Kunde und Entwickler ständig Zusammenarbeiten und kurze Kommunikationswege gewährlseitet sind, einfach zu erlernen, d.h. die Methode an sich ist einfach erlernbar, einfach veränderbar und gut dokumentiert, und adaptiv, also an die aktuelle Entwicklungssituation anpassar, ist.“ [Fon08]. Diese Definition ist meiner Meinung nach sehr gut gelungen. Das Einzige, was hier noch zu ergänzen ist, ist dass der Hauptfaktor für den Erfolg eines Projektes nicht der Prozess, sondern die Menschen, die an dem Projekt arbeiten, sind[Coc01].

Laut einer Studie der GPM[fPeK15] (Deutsche Gesellschaft für Projektmanagement e.V.) ist Scrum die am weitesten verbreitete Ausprägung der Agilen Softwareentwicklung (siehe Abbildung 2.2). Scrum erfüllt alle, in der Definition aufgezählten, Merkmale und hat sich durch seine einfache und flexible Handhabung etabliert.

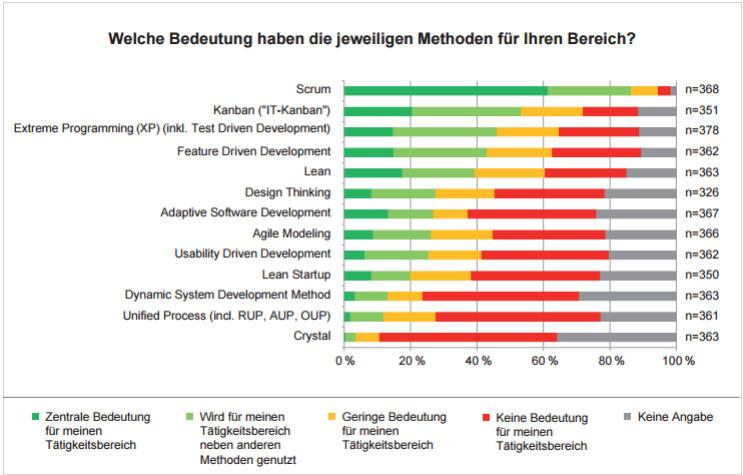

Abbildung 2.2: Bedeutung der angewendeten agilen Methoden [fPeK15]

Methoden wie Scrum sind grade dann sehr hilfreich und den klassichen Projektmanagement Methoden überlegen, wenn Anforderungen entweder nur sehr grob oder auch gar nicht definiert werden können. Dies kann z.B. bei innovativen Softwareprojekten ohne Erfahrungsschatz oder bei vielen Kundenprojekten sein. Denn Kunden wissen oft nicht was Softwaretechnisch umgesetzt werden kann. Somit muss der Projektnehmer im Projektverlauf evtl. Schritt für Schritt mit dem Projektgeber die Anforderungen analysieren und aufstellen.

3. Agiles Requirements Engineering

Das klassische Anforderungsmanagement, wie es z.B. mit einem Lasten- und Pflichtenheft gelebt wird, kann im agilen Projektmanagement nicht genutzt werden. Grund dafür sind die sich ändernden oder evtl. noch nicht vorhandenen Anforderungen. Diese müssen in agilen Requirements Engineering Methoden, häufig mit dem Projektgeber, erarbeitet werden.

3.1 Einführung

Das Requirements Engineering kann in vier Unterbereiche unterteilt werden (siehe Abbildung 3.1). Dies erleichtert die Klassifizierungen und Einordnungen der Agilen Methoden. Eine Methode muss in einer, kann jedoch auch in mehreren dieser Unterbereiche angesiedelt sein. In dieser Arbeit werden lediglich fünf Methoden, welche zu einem agilen Requirements Engineering Prozess gehören können, vorgestellt. Zu beachten ist, dass es viele weitere andere Methoden gibt und lediglich einige der bekanntesten erläutert werden. Zudem wurde darauf geachtet, dass mit diesen fünf Methoden alle Unterbereiche des Requirements Engineering abgedeckt wurden.

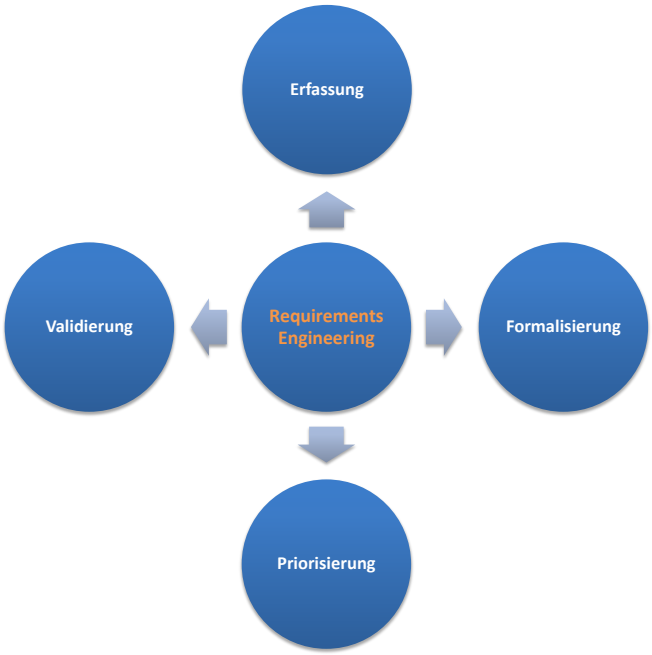

Abbildung 3.1: Unterbereiche des Requirements Engineering

3.2 Methodenkatalog

3.2.1 Brainstorming

Das Brainstorming ist lediglich in den Unterbereich „Erfassung“ eingegliedert. Hierbei handelt es sich um eine Kreativtechnik und ist in zwei Phasen aufgeteilt. Das Ziel der ersten Phase, die Kreativ-Phase, ist es viele Ideen zu sammeln und Parallelen bzw. Zusammenhänge zwischen den Ideen festzustellen. Anschließend folgt die Bewertungsphase, in der die Ideen auf Qualität hin bewertet werden [OJ89]. Kriterien zur Bewertung sind beispielsweise die Sinnhaftigkeit, technische Machbarkeit und die Projektrelevanz.

Teilnehmer des Brainstormings sind sowohl der Kunde, als auch das Entwicklerteam. Hierbei werden Synergieeffekte aus der Gruppenbildung genutzt. Damit Diskussionen nicht zu langwierig werden hat der Projektleiter die Rolle des Moderators einzunehmen.

Diese Methode ist sehr gut bei innovativen Projekten, sowie bei Projekten bei denen der Kunde noch keine genaue Vorstellung vom Endergebnis hat.

3.2.2 Story Cards

Die Story Cards lassen sich in den Unterbereichen „Erfassung“, „Formalisierung“ und „Priorisierung“ eingruppieren. Hierbei ist jedoch zu beachten, dass Ersteres und Letzteres in bestimmten, folgend erläuterten, Konstellationen wegfallen können.

Diese Methode kommt ursprünglich aus dem Extreme Programming [BF01] und wird zur unstrukturierten Dokumentation von Anforderungen verwendet. Story Cards sind im agilen Projektmanagement eine weitverbreitete Methode und sind in vielen agilen Frameworks wie z.B. Scrum zu finden.

Die Story Cards sind entweder Karteikarten oder eine digitale Abwandlung dessen. Auf ihnen stehen die Anforderungen aus der Nutzersicht. In der Regel ist das Format „Als *x* kann ich *y* tun, um *z* zu erreichen.“ zu verwenden. Somit stellt jede Story Card eine Anforderung dar.

Die (digitalen) Karteikarten sollten vom Kunden alleine erstellt werden. Ist dies auf Grund mangelnder technischer oder methodischer Kenntnisse nicht möglich kann ein technischer Berater, wie z.B. ein Entwickler, unterstützen. Dies gilt genauso bei der Priorisierung der Anforderungen.

Story Cards können jedoch auch nachgelagert nach einer anderen Erfassungs- und/oder Priorisierungsmethode eingesetzt werden. Somit dient diese Methode nur noch den Zweck der „Formalisierung“ der Anforderungen.

In der Regel müssen vor Projektbeginn noch nicht alle Anforderungen auf Story Cards niedergeschrieben sein. Diese können nach jeder Abarbeitung eines Iterationszyklus aufgestockt oder auch geändert werden.

3.2.3 Product Backlog

Das Product Backlog stammt aus dem Scrum Framework und enthält die gesamte Menge an Anforderungen/Features, die sich der Kunde wünscht. Diese Methode kann

in den Unterbereich „Formalisierung“ eingeordnet werden, wobei es auch priorisierende Elemente enthält.

Das Product Backlog muss bei Projektbeginn nicht voll ausgeprägt sein. Es reicht, wenn genügend Anforderungen für die erste Iteration im Projektzyklus vorhanden sind. Im laufe des Projektes erlangen Kunde und Entwickler mehr Wissen über das Produkt und weitere Anforderungen können dem Produkt Backlog hinzugefügt werden.

Zwischen den Anforderungen werden in dieser Methode sowohl technische, als auch nicht-technsiche Abhängigkeiten gepflegt. Dies ermöglicht die Erstellung eines groben Ablaufplanes.

Für das Produkt Backlog ist keine Struktur vorgegeben, was die Gefahr darstellt, dass verschiedene Anforderungen verschieden formuliert sein können. Dies bringt Inkonsistenzen mit sich und kann durch eine Verbindung mit Story Cards verhindert werden. Somit enthält ein Product Backlog eine Menge an Story Cards.

Die Priorisierung der einzelnen Anforderungen geschieht folgend. Zuerst werden vom Entwicklungsteam die Anforderungen auf Stundenbasis geschätzt. Anschließend bekommt der Kunde das Ergebnis der Schätzung mittgeteilt und priorisiert die Anforderungen nach seinem Ermessen. Das Entwicklungsteam stimmt darauf hin ab, welche Anforderungen in die nächste Iteration kommen.

3.2.4 Sprint Backlog

Auch der Sprint Backlog kommt aus dem Scrum Framework. Anders als das Product Backlog enthält es nicht alle Anforderungen, sondern nur jene, die in der Interation (Sprint) abgearbeitet werden sollen. Diese Methode ist lediglich im Unterbereich „Formalisierung“ einzuordnen.

Zu Beginn einer Iteration wird festgelegt wie viele Stunden Arbeit diese maximal enthalten darf, damit das Team ohne Mehrarbeit den Sprint an einem Tag x beenden kann. Anschließend werden der Iteration passend die vorher geschätzten Anforderungen zugeteilt. Zu beachten ist, dass während eines Sprints keine weiteren Anforderungen mit aufgenommen werden. Ist am Ende des Sprints mehr Zeit über als geplant wird der Sprint verfrüht beendet und ein Neuer geplant.

Wünschenswert währe in dieser Methode eine Validierung der Anforderungen. Einige Scrum Ausprägungen realisieren dies durch sogenannte Akzeptanzkriterien. Dies ist eine Checkliste für eine Anforderung. Erst wenn alle Punkte auf der Checkliste lauffähig sind (z.B. „Das laden der Seite dauert nicht länger als *x* Sekunden.“), wird die Anforderung als erledigt gekennzeichnet. Es wäre auch denkbar das Sprint Backlog mit Unit Tests zu kombinieren. So wird für jede Anforderung ein Unit Test geschrieben. Erst wenn dieser nicht fehlschlägt ist die Anforderung erledigt.

3.2.5 Unit Tests

Unit Tests werden oft nicht zum Requirements Engineering gezählt, da viele den Aufgabenbereich nur bis zur fertigen Anforderung sehen. Die Validierung und Abgleichung des Ergebnisses mit der Anforderung ist jedoch ein wichtiger Teil des Requirements Engineering.

In der agilen Welt der Softwareentwicklung sind Unit Tests jedoch ein zweischneidiges Schwert. Sie verfolgen nämlich nicht das Prinzip der „simplen Veränderung“. Dies liegt daran, dass wenn eine Anforderung sich ändert, nicht nur das Produkt modifiziert werden muss sondern, um die Konsistenz zu wahren, im Zweifel auch Unit Tests angepasst werden müssen. Dies ist erheblicher Mehraufwand.

Zum Anderen legen einige agile Methoden das Hauptaugenmerk auf das sogenannte „Test-First-Programming“. Hier werden zuerst die Tests geschrieben und anschließend gegen jene Programmiert. Dies hat u.a. motiviertere Entwickler, durch reduzierte Kopplung ein besseres Design und einen stärkeren Zusammenhalt zur Folge [LL10].

Unit Tests prüfen nach dem IO(Input/Output) Prinzip. Sie gleichen für einen fest definierten Input Wert den Ist-Output mit dem fest definierten Soll-Output ab. Der Test schlägt nur dann nicht fehl, wenn Ist- und Soll-Output exakt den selben Wert haben.

4. Fazit

Durch die Erläuterung und Erklärung einiger agilen Methoden, die z.B. in Frameworks wie Scrum und XP vorkommen oder auch ganz unabhängig für sich selbst stehen, kann man das agile Requirements Engineering gut vom klassischen trennen.

Das klassiche Requirements Engineering wird zumeist dann verwendet, wenn Anforderungen zu Projektbeginn bereits so ausführlich definiert sind, dass keine Änderungen abzusehen sind. Dies ist meist bei übersichtlichen Projekten wie kleine bis mittlere Webseiten oder bei großen Ausschreibungen (staatlich) der Fall. Hier können entweder alle Anforderungen bereits vor Projektbeginn unkompliziert ermittelt werden oder der Kunde hat bereits im Vorfeld genügend Zeit in die Anforderungsanalyse investiert so, dass die Ausarbeitung fertig ist und Bestandteil eines Vertrages werden kann.

Da viele der vorgestellten Methoden des agilen Requirements Engineering mit Iterationen oder sogenannten Sprints arbeiten fallen diese in ihrer ursprünglich geplanten Form aus dem klassischen Projektmanagement raus. Natürlich kann man diese in abgewandelter Form trotzdem in klassische Methoden integrieren und somit ein hybrides Verfahren erschaffen. Dies ist jedoch aufgrund der Aufhebung der Vor- und Nachteile aus gegenseitigen Synergieeffekten des klassichen und agilen Projektmanagements und Requirements Engineerings meist nicht erwünscht.

Werden diese Methoden jedoch in iterativen Projektzyklen eingesetzt kommen ihre Vorteile heraus. So ermöglichen sie es Projekte mit ungenauen Anforderungen nach und nach immer weiter und genauer zu definieren. Sie können bei Innovationsprojekten durch Kreativtechniken angereichert werden und somit technische Erfahrung des

Projektnehmers mit der geschäftlichen Erfahrung des Projektgebers kombinieren. Zudem ist herauszustellen, dass Kombinationen aus mehreren agilen Methoden zu extrem mächtigen, aber zugleich übersichtlichen Werkzeugen werden können. Die hier vorgestellten Methoden wurden dadurch immer weiter verbessert, und es ist noch nicht abzusehen wann dieser Prozess endet.

Literaturverzeichnis

[Bal08] Helmut Balzert. Lehrbuch der softwaretechnik: Softwaremanagement, 2008.

[BF01] Kent Beck and Martin Fowler. Planning extreme programming, 2001.

[Coc01] Alistair Cockburn. Agile software development: Software through people, 2001.

[Fon08] Christian Fonden. Agile requirements engineering. 2008.

[fPeK15] GPM Deutsche Gesellschaft für Projektmanagement e.V. and Hochschule Koblenz. Status quo agile. 2015.

[Joh02] Jim Johnson. Xp 2002 keynote speech (adressing 'chaos report'). 2002.

[LL10] Jochen Ludewig and Horst Lichter. Software engineering - grundlagen, menschen, prozesse, techniken, 2010.

[NA99] Joe Nandhakumar and David Avison. The fiction of methodological development. 1999.

[OJ89] Olaf and Johannes. Brainstorming: How to create successful ideas, 1989.

[PC86] David Lorge Parnas and Paul C. Clements. A rational design process: How and why to fake it. 1986.