Agile Softwareentwicklung

Seminar: Softwareentwicklung in der Wissenschaft

Benjamin Pöpel

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

Betreuer: Christian Hovy

27. August 2014

Inhaltsverzeichnis

1 Einleitung

Die Ausarbeitung des Seminars *Softwareentwicklung in der Wissenschaft* mit dem Thema *Agile Softwareentwicklung* verschafft zu Beginn einen Einblick in die agile Softwareentwicklung, indem das Konzept der agilen Softwareentwicklung, der Zusammenhang mit dem Projektmanagement und eine Abgrenzung zu anderen Konzepten der Softwareentwicklung dargestellt werden. Die Verfahrensweise der agilen Softwareentwicklung wird am Beispiel von *Extreme Programming (XP)* ausführlich beschrieben. Es werden dazu die grundlegenden Werte, Prinzipien, Phasen, Rollen und die Praktiken des XP verdeutlicht. Um einen Überblick über die Vielfalt der Vorgehensweisen der agilen Softwareentwicklung zu erhalten, wird eine Liste mit sämtlichen Verfahren genannt. Des Weiteren erfolgt eine kritische Auseinandersetzung mit den Chancen und Risiken der agilen Softwareentwicklung. In der abschließenden Zusammenfassung werden nochmals die wesentlichen Kriterien der agilen Softwareentwicklung hervorgehoben.

2 Agile Softwareentwicklung

In diesem Kapitel wird das Konzept der agilen Softwareentwicklung vorgestellt. Im Anschluss wird eine Einführung ins Projektmanagement mit dem Zusammenwirken der agilen Softwareentwicklung behandelt. Danach wird die agile Softwareentwicklung von anderen Konzepten der Softwareentwicklung abgegrenzt.

2.1 Konzept der agilen Softwareentwicklung

Das Konzept der agilen Softwareentwicklung ist durch das Leitbild des *agilen Manifests* geprägt. Gebildet wurde es von 17 Autoren im Jahr 2001 [1]. Es stellt die Grundlage für verschiedene Entwicklungsansätze der agilen Softwareentwicklung dar und enthält die folgenden Wertaussagen:

- *Menschen und Zusammenarbeit* sind wichtiger als Prozesse und Werkzeuge.
- *Lauffähige Software* ist wichtiger als umfangreiche Dokumentation.
- *Zusammenarbeiten mit Auftraggebern* ist wichtiger als Vertragsverhandlungen.
- *Reagieren auf Änderungen* ist wichtiger als das sture Befolgen eines Plans.

Quelle: http://www.agilemanifesto.org/

Es ist dabei stets zu berücksichtigen, dass aus den Wertaussagen alles wichtig ist, nur der kursiv hervorgehobene Teil ist noch wichtiger als der andere Teil der Aussagen.

Damit werden die wesentlichen Grundsätze festgelegt, die u. a. die beteiligten Personen der Softwareentwicklung und dessen Interaktionen untereinander hervorheben. Die Schwierigkeit besteht jedoch in der Umsetzung der Wertaussagen, denn diese enthalten

keine Angaben über die Ausprägungen der Grundsätze. Es kann z. B. die Frage aufkommen, wann genau die Dokumentation zu umfangreich wird, die nicht im agilen Manifest beantwortet wird. Daher bedarf es einer Abwägung der Beteiligten der Softwareentwicklung, in wie weit welche Maßnahmen umgesetzt werden sollen. Es ist sinnvoll eine agile Angemessenheit anzustreben, die sich flexibel auf verschiedene Umstände anwenden lässt und nicht in extremen aufwendigen Maßnahmen entartet, die möglicherweise auf Probleme nur schwer reagieren können. Es wird in der agilen Softwareentwicklung der Fokus auf die zu erreichenden Zielsetzungen sowie die Behandlungen von sozialen und technischen Problemen gelegt. Die Ergebnisorientierung der lauffähigen Software wird von der Vorgehensweise, die vorgibt so wenig wie möglich und so viel wie nötig zu realisieren, verdeutlicht. Die Einfachheit wird in den Vordergrund gestellt, damit Umstände nicht unnötig verkompliziert werden. Es sind einige weitere Prinzipien erstellt worden, die die agile Vorgehensweise in der Softwareentwicklung prägen. So wird festgehalten, dass die kontinuierliche Auslieferung von Software an den Auftraggeber schon früh in kurzen Zeitspannen stattfinden soll, denn so werden der Fortschritt und der daraus resultierende Wert der bisherigen Entwicklung früher erkennbar und nutzbar. Die Zusammenarbeit von Fachexperten, die fundierte Kenntnisse von der Domäne verfügen, mit den Softwareentwicklern soll nach Möglichkeit täglich zum Tragen kommen. Dadurch kann frühzeitig ermittelt werden, was sich konkret realisieren lässt und was nicht. Die Informationsübermittlung sollte möglichst von Angesicht zu Angesicht ausgeübt werden, da so mehr übermittelt werden kann, als nur der Inhalt. Das Umfeld sollte so bereitgestellt werden, dass die beteiligten Personen motiviert und unterstützt werden. Dabei soll die Einhaltung einer einheitlichen Arbeitsgeschwindigkeit für die Nachhaltigkeit der Entwicklung maßgeblich sein. Als wesentliche Kennzahl für den Fortschritt soll die Funktionsfähigkeit der Software dienen. Veränderungen sollen immer ermöglicht werden können, damit auf dringende Anforderungen vom Auftraggeber eingegangen werden kann. Das sollte bei der Selbstorganisation der beteiligten Personen mit berücksichtigt werden. Dabei können die Beteiligten ihre eigene Arbeitsweise identifizieren und Best Practices untereinander austauschen, wodurch alle Beteiligten motivierter an die Anforderungen herangehen können, da sie von der Wirksamkeit der Methoden und Verfahren überzeugter sein können, als durch starre vorgegebene Vorgehensweisen. Die Handlungen der Beteiligten sollen einer Selbstreflexion vollzogen werden, um dessen Potential zur Steigerung ihrer Effizienz zu ermitteln und anwenden zu können. Diese Prinzipien sind abhängig von den Gegebenheiten der Projekte, die die Randbedingungen bestimmen [2].

2.2 Agile Softwareentwicklung und Projektmanagement

Damit eine einheitliche Vorstellung von dem Begriff Projekt, auf dessen Basis das Projektmanagement und die agile Softwareentwicklung zusammenwirken, etabliert werden kann, wird dieser zunächst kurz erläutert.

Ein Projekt enthält klare Anforderungen, die zur Erreichung einer Zielsetzung dienen. Es ist vom operativen Geschäft eines Unternehmens abgrenzbar, wodurch seine Einmaligkeit und Interdisziplinarität verdeutlicht werden. Durch einen Start- und Endtermin

eines Projektes existiert eine zeitliche Restriktion, die durch weitere Restriktionen, wie bspw. monetärer und personeller Restriktionen, ergänzt werden. Ein Projekt verkörpert einen Erfolg für das Unternehmen und enthält stets Risiken.

Das Projektmanagement ist für die konkrete Planung, Koordination, Steuerung und Überwachung eines Projektes zuständig, damit ein Projekt effizient realisiert werden kann. Dafür gibt es drei Sichten innerhalb des Projektmanagements, die funktionale, institutionelle und instrumentelle Sicht. Die funktionale Sicht behandelt die Gesamtheit aller Planungs-, Koordinations-, Steuerungs- und Überwachungsaufgaben eines Projektes. Dagegen betrachtet die institutionelle Sicht des Projektmanagements die Rollen der beteiligten Personen am Projekt und dessen Zusammenwirken. Der instrumentellen Sicht unterliegt eine Menge von konkreten Instrumenten, die zur Unterstützung der anfallenden Aufgaben der funktionalen Sicht dienen sollen. Dabei ist das Projektmanagement stets von einem Konkurrenzverhältnis der Restriktionen des Projekts umgeben. Es sind die drei Restriktionen der Zeit, der Qualität und die des Ressourceneinsatzes. Die Zeit beschränkt das Projekt bezüglich der Termineinhaltung. Die Qualität verkörpert die Ergebniserreichung und der Ressourceneinsatz verursacht Kosten, unter dem ebenfalls die beteiligten Personen des Projekts mitberücksichtigt werden. Dieses Konkurrenzverhältnis wird als *Magisches Dreieck* bezeichnet [3].

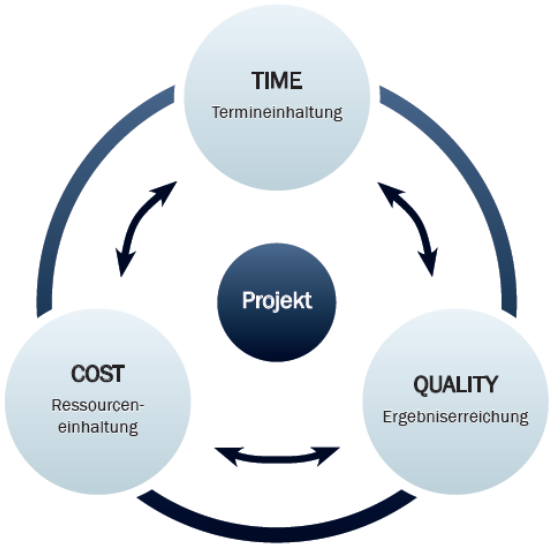

Abbildung 1: Magisches Dreieck
Quelle: http://www.projektmanagementhandbuch.de/projektinitiierung/zieldefinition/

Ein Projekt durchläuft verschiedene Phasen während seiner Umsetzung. Die typischen Phasen sind die *Analyse-*, *Entwurfs-*, *Realisierungs-* und *Einführungsphase*. Es gibt meh-

rere unterschiedliche Modelle in der Softwareentwicklung, die sogenannten *Vorgehensmodelle*, die eine konkrete Vorgehensweise dieser Phasen darstellen und u. a. auch noch diese Phasen weiter differenzieren. Die agilen Vorgehensmodelle sind jedoch iterativ und durchlaufen die Phasen zyklisch. Dabei werden die Prinzipien des agilen Manifests umgesetzt (siehe Kapitel 2.1.). Jedoch werden in der agilen Softwareentwicklung keine Vorgehensmodelle mit einer strikten und starren Vorgehensweise befürwortet, denn nachdem agilen Manifest ist es notwendig stets auf Veränderungen eingehen zu können. Die agile Softwareentwicklung verwendet einen Projektmanagementansatz, in dem die kooperative Arbeit im Vordergrund steht, wodurch die Komplexität reduziert und die Kommunikation der beteiligten Personen des Projektes ausgeprägt werden soll. Daher ist es sinnvoll eine Balance zwischen der Struktur der Vorgehensmodelle und der Flexibilität, die Änderungen erfordern, zu erzielen [3].

2.3 Abgrenzung zu anderen Konzepten der Softwareentwicklung

Die agile Softwareentwicklung ist durch das agile Manifest stark geprägt und gibt mehrere Prinzipien vor, die während der Entwicklung berücksichtigt werden sollen. In anderen Konzepten der Softwareentwicklung gibt es kein Manifest mit derartigen Prinzipien. Es stehen lediglich der Projekterfolg und vor allem der monetäre Erfolg im Vordergrund. Es gilt das Prinzip, dass eine Planung aufgestellt wird, die sämtliche zukünftigen Ereignisse erfassen soll, von der nach Möglichkeit Abweichungen ferngehalten werden sollen, damit die geplante Ordnung und Struktur des Projektes bewahrt werden können. Das steht im direkten Gegensatz zur agilen Softwareentwicklung in der Veränderungen stets berücksichtigt und gewünscht werden.

Ein Beispiel für den Ablauf der Phasen der Softwareentwicklung, die nicht dem Konzept der agilen Softwareentwicklung folgt, ist das Vorgehensmodell des *Wasserfallmodells* [3]. Es unterteilt den Projektablauf in sieben Phasen, die sequentiell und einmalig durchlaufen werden. Es sind die Phasen der *Systemanforderungen*, *Softwareanforderungen*, *Anforderungsanalyse*, *Design*, *Implementierung*, *Testen* und der *Einführung*.

In der ersten Phase der Systemanforderungen werden die nicht-funktionalen Anforderungen festgelegt, die die Software umfassen, wie z. B. den Preis, die Verfügbarkeit, die Dokumentation u. Ä. In der zweiten Phase der Softwareanforderungen werden die funktionalen Anforderungen aufgestellt. Das Resultat der ersten beiden Phasen wird in dem Lastenheft festgehalten. Die darauffolgende Anforderungsanalyse setzt sich mit dem Lastenheft im Hinblick auf die Vollständigkeit, Machbarkeit, Notwendigkeit sowie einer Priorisierung der Anforderungen auseinander. Daraus entsteht das Pflichtenheft, das ebenfalls um die Abnahmekriterien der Software ergänzt wird. In der Phase des Designs werden Entscheidungen bezüglich der Architektur und der einzusetzenden Technologien getroffen, die damit die ersten Details für die Implementierung definieren. Diese Details werden in der folgenden Phase der Implementierung realisiert. Ist die Implementierung abgeschlossen, erfolgt das Testen der entstandenen Software. Danach wird die Software vom Auftraggeber anhand der Abnahmekriterien aus dem Pflichtenheft abgenommen.

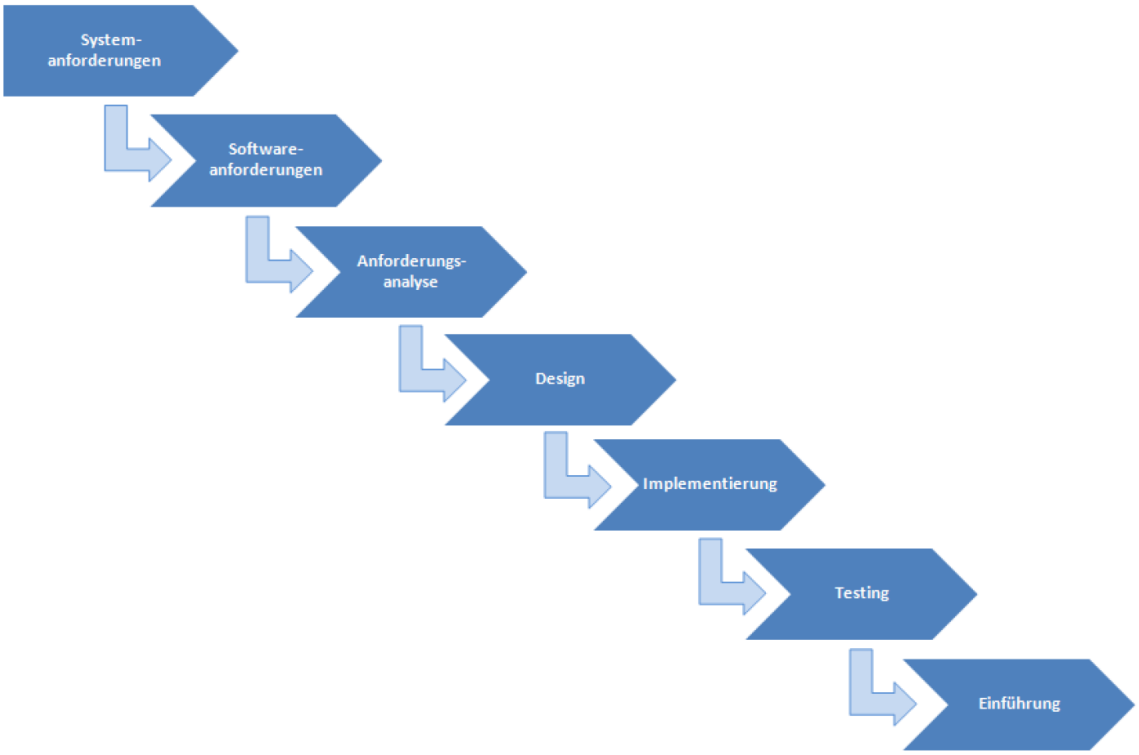

Abbildung 2: Wasserfallmodell
Quelle:
http://www.fernuni-hagen.de/imperia/md/content/ps/masterarbeit-harwardt.pdf

War die Abnahme erfolgreich, geht es zur letzten Phase der Einführung über, in der die Inbetriebnahme der Software beim Auftraggeber erfolgt. Darüber hinaus sind Wartungsarbeiten notwendig, die aber nicht mehr der Softwareentwicklung zuzuordnen sind.

Es ist erkennbar, dass nach diesem Vorgehensmodell jede Phase für sich als abgeschlossene Einheit betrachtet werden kann, wodurch keine zyklische Vorgehensweise vorhanden ist. Des Weiteren ist der Auftraggeber nur in den ersten und in den letzten Phasen der Softwareentwicklung involviert, was eine umfassende Spezifikation der Software zu Beginn der Softwareentwicklung erfordert [4]. Änderungen werden damit weitestgehend ausgeschlossen. Da das Testen erst nach der Phase der Implementierung erfolgt, werden Fehler ziemlich spät identifiziert, was zu hohen Kosten führen kann. Es wird auch erst spät in dieser Vorgehensweise eine lauffähige Version der Software erstellt, wodurch der Auftraggeber nicht frühzeitig ein Feedback geben kann. Dadurch kann ebenfalls die Gefahr bestehen, dass die fertig entwickelte Software u. U. nicht dem Wunsch des Auftraggebers entspricht. Die agile Softwareentwicklung versucht das durch seine zyklische Vorgehensweise zu vermeiden, damit auf Fehler und Änderungen frühzeitig eingegangen werden kann [3].

3 Extreme Programming (XP)

In diesem Kapitel wird die Verfahrensweise der agilen Softwareentwicklung konkret am Beispiel von dem agilen Vorgehensmodell *Extreme Programming* dargestellt. Dabei werden die Wertvorstellungen und Prinzipien des agilen Manifests wiedererkennbar sein. Im Anschluss wird eine Liste weiterer Vorgehensmodelle der agilen Softwareentwicklung genannt.

Extreme Programming, auch als XP bekannt, ist ein Verfahren der agilen Softwareentwicklung, das 5 Werte, 14 Prinzipien, 13 Primär- und 11 Folgepraktiken definiert. Es wurde von Kent Beck im Jahr 2000 entwickelt [5]. Im Gegensatz zu anderen Verfahren der agilen Softwareentwicklung ist XP wesentlich umfassender strukturierter und gibt eine Menge von Praktiken vor, die im Projekt verwendet werden sollen [3].

3.1 Fünf Werte

Die fünf Werte sind *Kommunikation*, *Einfachheit*, *Rückkopplung*, *Mut* und *Respekt*. Sie bilden die Grundlage für die 14 Prinzipien [3]. Die Kommunikation gilt als entscheidend für den Projekterfolg und lässt sich in der Art mit dem Auftraggeber und der Art der internen Kommunikation innerhalb des Entwicklerteams unterscheiden. Der Auftraggeber soll Einblicke in den Projektablauf mit den Erfolgen sowie Misserfolgen erhalten können und die interne Kommunikation ist für die Selbstorganisation des Teams wichtig. Die Einfachheit steht für die Komplexitätsreduktion des Projektes, die durch die Beschränkung auf das Notwendige realisiert wird. Rückkopplungen stellen das Feedback dar, das zum einen vom System und zum anderen vom Auftraggeber geliefert wird. Vom System wird mit dem Durchlaufen von Tests ein Feedback über die Funktionalität gegeben und der Auftraggeber liefert regelmäßig ein Feedback über die Resultate und äußert ggf. Änderungsvorstellungen. Für die Einfachheit, Rückkopplungen und Kommunikation sowie für die Notwendigkeit einer Umstrukturierung des Designs, Codes oder gar der Vorgehensweise wird stets Mut benötigt. Es wird für die erfolgreiche Umsetzung eines Projektes Vertrauen zwischen allen beteiligten Personen benötigt, das nur durch einen respektvollen Umgang generiert werden kann [6].

3.2 14 Prinzipien

Die fünf Werte sind jedoch sehr abstrakt und werden daher von 14 Prinzipien konkretisiert. Es sind die folgenden Prinzipien:

1. **Menschlichkeit**
   Die Menschlichkeit spielt für die Software eine wichtige Rolle, da Software von Menschen für Menschen entwickelt wird, die alle Bedürfnisse haben, auf die Rücksicht genommen werden sollen [3].

2. **Wirtschaftlichkeit**
   Die Entwicklung der Software macht nur Sinn, wenn sie wirtschaftlich ist und einen monetären sowie erfahrungstechnischen Mehrwert bringt [3].

3. **Gegenseitiger Vorteil**
   Ein gegenseitiger Vorteil soll erzielt werden, in dem jeder Beteiligte auf seinen aktuellen Vorteil sowie auf den künftigen Vorteil anderer hinarbeitet, die eine zukünftige Bearbeitung der Software erleichtern kann [3].

4. **Selbstähnlichkeit**
   Die Selbstähnlichkeit verkörpert die Wiederverwendbarkeit von bereits Bestehenden, die auf das gesamte Projekt und nicht nur in der Implementierung, eine Anwendung finden soll [3].

5. **Verbesserung**
   Um Verbesserungen im Rahmen des Konzepts von XP sollen sich alle beteiligten Personen für sämtliche Bereiche des Projektes bemühen [3].

6. **Mannigfaltigkeit**
   Damit ein Team Umstände aus verschiedenen Perspektiven betrachten kann, ist ein mannigfaltiges Team notwendig. So können Probleme sowie Konflikte aufgedeckt werden und dadurch optimale und innovative Lösungen kreiert werden [3].

7. **Reflexion**
   Sämtlichen Handlungen aller Beteiligten sollen dauerhaft einer Reflexion unterliegen [3].

8. **Fluss**
   Das Prinzip vom Fluss in XP verkörpert die Auslieferung von nützlicher Software, die kontinuierlich während der Softwareentwicklung vollzogen werden soll [3].

9. **Gelegenheit**
   Auftretende potentielle Probleme sollen als Gegebenheiten betrachtet werden, die Chancen für neue Erkenntnisse und erweiterte Lösungen bieten können [3].

10. **Redundanz**
    Damit Probleme effizienter identifiziert und behandelt werden können, wirkt eine Redundanz der Kompetenzen der beteiligten Personen am Projekt unterstützend [3].

11. **Fehlschlag**
    Auch das Akzeptieren von Fehlschlägen gehört zu einem Projekt dazu, wodurch ebenfalls Lerneffekte erzielt werden können [3].

12. **Qualität**
    Die Qualität ist ein wesentlicher Bestandteil eines Projekts. Es kann durch hohe Qualität Nachhaltigkeit geschaffen werden, die in der Zukunft Aufwand reduzieren kann [3].

13. **Babyschritte**
    Eine hohe Qualität erfordert jedoch selbst viel Aufwand. Daher besagt das Prinzip der Babyschritte, dass es sinnvoll wäre die Qualität in kleinen Schritten zu

erhöhen, als sie einmalig mit einem großen Aufwand zu erzeugen. Damit kann das Risiko ebenfalls verringert werden, denn somit können Fehlschläge eher weniger weitreichende Konsequenzen haben, als wenn diese in großen umfassenden Schritten stattfinden würden [3].

14. **Akzeptieren von Verantwortlichkeiten**
    Das Prinzip Akzeptieren von Verantwortlichkeiten verdeutlicht, dass Verantwortlichkeiten nur akzeptiert werden, wenn die verantwortliche Person selbst von ihrer Verantwortung überzeugt ist und nicht diese von einer anderen Person einfach auferlegt bekommt [3].

3.3 Phasen

Die Phasen im Extreme Programming sind die vier Phasen *Planung*, *Design*, *Kodieren* und das *Testen*. Diesen Phasen werden XP-Praktiken zugeordnet. Dabei gibt XP selbst keine klare Reihenfolge dieser Phasen vor. Diese Phasen werden in Iterationen ausgeführt und erzeugen iterativ Inkremente die schrittweise die Software bilden. Dabei kann jedes Inkrement vom Auftraggeber stets betrachtet werden, damit er ein Feedback geben kann, wie es in den Prinzipien des XP gefordert wird [6]. Die Iterationen sollen eine Dauer von ca. zwei Wochen haben, können aber auch länger sein, wenn das Team das für sinnvoll erachtet. Durch diese zyklische Vorgehensweise, erfolgt in jeder Iteration eine Planung, in der Schätzungen bezüglich der Kosten und Risiken für die zu entwickelnden Komponenten erstellt werden. In der Planung sollen immer die Erfahrungen aus den vorigen Iterationen mit einfließen, an der das ganze Entwicklerteam beteiligt ist. Durch die Beteiligung können sich alle mit der Iterationsplanung identifizieren, womit ihre Motivation gefördert werden kann. Auf dieser Grundlage sowie der Werthaltigkeit der Komponenten für den Auftraggeber, wählt dieser die Komponenten aus, die in der nächsten Iteration entwickelt werden sollen. Sind die Komponenten bestimmt worden, werden Akzeptanztests vom Auftraggeber definiert, die als Maßstab für den Erfolg dienen sollen. Beim Design sollen nur die Aspekte der aktuellen Iteration berücksichtigt werden, sodass das Design möglichst einfach gehalten wird und trotzdem alle Tests erfüllt werden [2]. In der Phase Kodieren wird mit der eigentlichen Implementierung begonnen, in der der Auftraggeber dauerhaft integriert sein soll. So soll gewährleistet werden, dass der Auftraggeber sich schnell an Entscheidungsprozessen beteiligen kann und eine rapide Reaktion auf ändernde Anforderungen des Auftraggebers ermöglicht werden kann. Tests finden während des gesamten Projekts eine Anwendung. Dabei stehen die Akzeptanztests eines Inkrements am Ende einer Iteration [6].

3.4 Rollen

In Extreme Programming werden zentralen Rollen definiert. Es existiert somit die Rolle des Auftraggebers, der auch als Kunde bezeichnet wird, der Entwickler und die des XP-Coachs. Die Besonderheit der Rolle des Auftraggebers ist, das er oder ein kundiger Vertreter permanent in der räumlichen Nähe des Entwicklerteams anwesend sein soll. Er

soll somit für geschäftliche Fragestellungen und zur Ergänzung von Fachkonzepten dienlich sein. Die Entwickler kooperieren mit dem Auftraggeber und entwickeln die Software. Dabei eignet sich XP für kleine und mittelgroße Entwicklerteams bis zu 15 Personen. Der XP-Coach dient zur Unterstützung des Entwicklerteams in Bezug auf methodische und koordinative Fragestellungen. Somit schafft er Anreize für Entscheidungsprozesse, trifft aber selbst keine Entscheidungen, sondern überlässt diese den Entwicklern [2].

3.5 Praktiken

In den Phasen sollen verschiedene Praktiken situationsabhängig zum Einsatz gebracht werden. Es werden daher in XP 13 Primär- und 11 Folgepraktiken erwähnt, die auf etablierte Vorgehensweisen zurückgreifen, die auch als *Best Practices* bekannt sind [3].

3.5.1 Primärpraktiken

Die Primärpraktiken können kontextunabhängig in der Softwareentwicklung verwendet werden. Die Primärpraktiken sind die Folgenden:

1. **Räumliche Zusammenarbeit**
   Alle beteiligten Personen des Projekts sollen zusammen in einem Raum arbeiten, damit u. a. die Kommunikation effektiver gestaltet werden kann [3].

2. **Komplettes Team**
   Das komplette Team soll alle notwendigen Kompetenzen verfügen, die ein erfolgreicher Projektabschluss benötigt. Damit sollen Abhängigkeiten von externen Ressourcen vermieden und eine schnelle flexible Umsetzung ermöglicht werden [3].

3. **Informative Arbeitsumgebung**
   Alle Beteiligten sollen sich schnell und einfach einen Überblick über die wichtigsten Informationen verschaffen können, wie bspw. über den derzeitigen Stand des Projektes. Dazu ist eine informative Arbeitsumgebung erforderlich, die u. a. mit Whiteboards gestaltet verwendet werden kann [3].

4. **Energiegeladene Arbeit**
   Eine energiegeladene Arbeit ist nur dann möglich, wenn sich jeder konzentrieren kann. Deshalb sollte es im Interesse von jedem Einzelnen sein, so zu arbeiten, dass jeder seine Aufgaben engagiert erledigen kann [3].

5. **Pair-Programming**
   In der Praktik Pair-Programming sitzen zwei Entwickler gemeinsam vor einem Computer und bearbeiten dieselbe Aufgabenstellung. Es entsteht eine gegenseitige Kontrolle der beiden Entwickler, die zur Folge hat, dass Fehler vermieden werden können und sich damit die Qualität der Software erhöhen kann [3].

6. **User Stories**
   In den User Stories werden Anforderungen an die Anwendungsfälle informell be-

schrieben. Diese werden oft auf Karteikarten festgehalten, die auch als *Story Cards* bezeichnet werden [3].

7. **Wochenzyklus**
   Die Länge einer Iteration soll auf die Dauer von einer Woche festgelegt werden. So soll am Anfang einer Woche mit der Planung der Iteration begonnen werden, gefolgt von der Erstellung des Designs, damit der Rest der Woche für die Implementierung und das Testen zur Verfügung stehen kann [6].

8. **Quartalszyklus**
   Der Quartalszyklus ist die feste Vorgabe für einen Release, in dem größere Einheiten der Software vom Auftraggeber anhand von den definierten Akzeptanztests abgenommen werden. Es findet ebenfalls im Quartalszyklus einmal eine Reflexionssitzung des Teams statt, in der Probleme und Lösungen identifiziert werden sollen [6].

9. **Freiraum**
   Durch einen Freiraum kann jedes Teammitglied für sich selbst eigene Aktivitäten entwickeln, die mit seinen Aufgaben oder seinen Interessen bezüglich des Projekts zu tun haben. So können neue wertvolle Ideen kreiert werden [6].

10. **Zehn-Minuten-Build**
    Ein Build der Software mit allen Tests soll nicht länger als zehn Minuten Zeit in Anspruch nehmen. Wenn ein Build mehr Zeit benötigen sollte, besteht die Gefahr, dass sich die Anzahl der Builds reduzieren, wodurch Probleme häufiger auftreten können [6].

11. **Kontinuierliche Integration**
    Damit eine aktuelle Codebasis entstehen kann, ist es notwendig, dass kontinuierlich eine Integration vom Quellcode in kurzen Zeitabständen stattfinden kann [6].

12. **Testgetriebene Entwicklung**
    Auf Grund der zu Beginn definierten Akzeptanztests mit dem Auftraggeber, wird in XP eine testgetriebene Entwicklung gefordert. Die Tests dienen damit zur Beschreibung der Anforderungen und sollen vor der eigentlichen Implementierung entwickelt werden. Es werden natürlich noch weitere Tests der Software später erstellt, in denen u. a. das Umfeld der Software getestet werden kann [6].

13. **Inkrementeller Entwurf**
    Durch Veränderungen der Anforderungen kann der bisherige Entwurf, der auch als Design bezeichnet wird, nicht mehr aktuell sein und bedarf einer Überarbeitung. Daher ist ein inkrementeller Entwurf notwendig, der den Entwurf nachdem aktuellen Stand des Projekts anpassen kann [6].

3.5.2 Folgepraktiken

Die Folgepraktiken sollen erst nach einem erfolgreichen Einsatz von den Primärpraktiken eingesetzt werden. Die Folgepraktiken sind die Folgenden:

1. Echte Kundeneinbindung
2. Inkrementelle Ausbreitung
3. Teamkontinuität
4. Schrumpfende Teams
5. Ursachenanalyse
6. Gemeinsamer Quelltext
7. Quelltext und Tests
8. Eine Quelltextbasis
9. Tägliche Ausbreitung
10. Vertrag mit verhandelbarem Ausgang
11. Bezahlung pro Benutzung

[3]

3.6 Weitere Verfahren

Es gibt noch weitere Verfahren der agilen Softwareentwicklung, die ebenfalls auf dem agilen Manifest aufbauen. Zur Übersicht und Verdeutlichung der Vielfalt an agilen Vorgehensmodellen wird hier eine Liste mit den Namen der weiteren agilen Verfahren aufgeführt, die zum Teil noch unterschiedliche Ausprägungsformen haben können.

- ActiF
- Adaptive Software Development (ASD)
- Agile Enterprise (ehemals X Breed)
- Agile Model Driven Development (AMDD)
- Behavior Driven Development (BDD)
- Crystal
- Design Driven Development (D3)
- Dynamic System Development Method (DSDM)

- Eclipse Way Process
- Evolutionary Process For Integrating Cots-Based Systems (EPIC)
- Evolutionary Project Management & Product Development (EVO)
- Feature Driven Development (FDD)
- Iconix
- Internet-Speed Development
- Lean Software Development (LSD)
- Microsoft Solutions Framework For Agile Software Development (MSF4ASD)
- Mobile-D
- Rapid Application Development (RAD)
- Scrum
- Test Driven Development (TDD)
- Unified Process (UP)
- Agile Unified Process (AUP)
- Essential Unified Process (EssUP)
- Open Unified Process (OpenUP)
- Usability Driven Development (UDD)

[7]

4 Chancen & Risiken

In diesem Kapitel werden die Chancen und Risiken der agilen Softwareentwicklung im Allgemeinen sowie auf Aspekte des Verfahrens von Extreme Programming dargestellt.

4.1 Chancen

Die agile Softwareentwicklung kann die typischen Probleme der Softwareentwicklung beseitigen. Es wird die Kommunikation zwischen den Entwicklern und den Auftraggebern sowie unter den Entwicklern selbst gefördert. Hat der Auftraggeber noch keine konkreten Vorstellungen von dem Projekt [3], oder kennt er nicht die vollständige Spezifikation der Software, so kann er diese zusammen mit den Entwicklern im Laufe der Projektdurchführung aufstellen. Dadurch wird gewährleistet, dass die benötigten und unnötigen

Anforderungen von den Entwicklern sowie vom Auftraggeber identifiziert werden können [8]. Somit entstehen ständig neue Anforderungen und Änderungen von bestehenden Anforderungen, die durch die Flexibilität des Extreme Programming mit der iterativen Planungsweise berücksichtigt werden können. Dagegen wäre in anderen Konzepten, wie dem Vorgehensmodell des Wasserfallmodells, eine komplette umfangreiche Neuplanung erforderlich [9]. Die Erzeugnisse entsprechen damit den Erwartungen des Auftraggebers, da sämtliche Änderungen umgesetzt werden können [10]. Die überflüssige und veraltete Dokumentation kann durch die zyklische Vorgehensweise und die Fokussierung auf lauffähiger Software ebenfalls vermieden werden [3]. Durch die kurzen Phasen wird eine schnellere Auslieferung von Komponenten ermöglicht, die Risiken frühzeitig aufdecken können. Es wird damit ein permanenter Soll-Ist-Vergleich geschaffen, der die Qualität der Software erhöht [10]. Damit werden ebenfalls mehr Erfolgserlebnisse der Entwickler erzielt. Eine weitere Risikominimierung ist die frühe Möglichkeit des Auftraggebers ein Feedback geben zu können [11]. Das Risiko einer Mitarbeiterfluktuation reduziert sich durch die hohe Identifikation der Mitarbeiter mit dem Projekt, da bspw. die Mitarbeiter in sämtlichen Phasen der Entwicklung miteinbezogen werden und einen Freiraum für die Anwendung von eigenen Methoden haben [10]. Aber selbst wenn eine Fluktuation stattfinden sollte, können negative Auswirkungen durch die Wissensweitergabe innerhalb des Teams gesenkt werden. Des Weiteren hat die Software stets den größtmöglichen Wert für den Auftraggeber, da dieser in jeder Iteration die wichtigsten und wertvollsten zu entwickelnden Komponenten auswählt [11], sodass durch deren frühzeitigen Auslieferung eine Inbetriebnahme schneller ermöglicht werden kann und damit u. U. schon Kosten während der Projektlaufzeit gedeckt werden können [12].

4.2 Risiken

Allerdings birgt der Einsatz der agilen Softwareentwicklung nicht nur Chancen sondern auch Risiken. Die agile Softwareentwicklung geht von einer optimalen personellen Besetzung aus, wie z. B. die Anforderung im Extreme Programming, dass der Auftraggeber dauerhaft vor Ort zur Klärung von Fragen zur Verfügung stehen soll. So etwas lässt sich nicht immer realisieren. Gerade in langen Projekten wäre es unrealistisch, wenn der Auftraggeber von seinem eigentlichen Tagesgeschäft dauerhaft fehlen würde, um im Projektgeschehen mitwirken zu können [3]. Darüber hinaus kann es auch am Willen des Auftraggebers mangeln, am Projekt, wie in der von XP geforderten Weise, mitwirken zu wollen. Für die agile Vorgehensweise wird stets Vertrauen des Auftraggebers an das Entwicklerteam benötigt [8]. Durch die Fokussierung des agilen Manifests auf den Menschen sind die negativen Beeinflussungsfaktoren des Menschen mit zu berücksichtigen. Es können z. B. zwischenmenschliche Probleme entstehen, wenn alle beteiligten Personen des Projekts dauerhaft in einem Raum gemeinsam arbeiten. Hierfür bietet XP keine expliziten Praktiken an, falls solche Probleme eintreten sollten. Es kann auch eintreten, dass die Entwickler keinen permanenten Kontakt zum Auftraggeber aufrechterhalten wollen oder Angst vor Verantwortungen haben [3]. Durch die hohen Anforderungen an den Entwicklern hängen die Ergebnisse stark von den deren individuellen Kompetenzen und Motivationen ab. Es werden hochqualifizierte Entwickler benötigt, um die agile

Vorgehensweise effizient umsetzen zu können [13]. Somit wäre eine hohe Mitarbeiterfluktuation sehr erfolgskritisch zu betrachten [3]. Eine missverständliche Kommunikation sowie verspätete Entscheidungen können auch hier einen Mehraufwand verursachen [13]. Die Unterstützung für große und verteilte Teams ist in XP kaum konsequent durchsetzbar. Der Koordinationsaufwand würde steigen und damit die Flexibilität der agilen Vorgehensweise mindern [3]. Daher ist eine Anwendung auf große und sicherheitskritische Projekte fraglich [13]. Die Überwachung des Projekts lässt sich schwieriger handhaben, da es keine konkreten abgeschlossenen Phasen gibt, wie z. B. im Wasserfallmodell. Durch das dauerhafte Zulassen von Änderungen der Anforderungen wird eine fixe Festsetzung von Inkrementen mit einem Liefertermin und einen Lieferumfang schwer zu gewährleisten [3]. Somit kann es passieren, dass Entwickler dazu neigen Software zu schnell fertigstellen zu wollen, wodurch Tests vernachlässigt werden könnten, die damit die Qualität der Software reduzieren würde [8]. Durch die iterative Fertigstellung von Inkrementen und den Änderungen der Anforderungen, die nach der agilen Vorgehensweise stets mit berücksichtigt werden sollen, lässt sich die Planung eines Gesamtpreises nicht einfach gestalten [3].

5 Zusammenfassung

Das agile Manifest bildet die Grundlage für die agilen Vorgehensmodelle, wie dem Extreme Programming. In XP werden dazu Werte, Prinzipien und Praktiken definiert, die die Vorgehensweise explizit festlegen. Dabei werden die Rollen der beteiligten Personen an einem Projekt stets im Zusammenhang betrachtet und deren Interaktionen hervorgehoben. Das Besondere an der agilen Softwareentwicklung ist die iterative Vorgehensweise der Phasen, die ein Projekt durchläuft. Es werden dadurch Chancen realisiert, die Problematiken des Umgangs mit dem Auftraggeber vermeiden können. Hat der Auftraggeber noch keine konkrete Vorstellung von der Software, kann er in regelmäßigen Abständen Inkremente begutachten und Änderungswünsche äußern, falls ihm etwas nicht gefällt. Durch die Hervorhebung der Kommunikation mit allen beteiligten Personen des Projekts, wie in Extreme Programming gefordert, kann das gewährleistet werden. Somit wird eine Flexibilität aufrecht gehalten, die Änderungen erfordern und eine hohe Identifikation der Entwickler mit dem Projekt gefördert. Dem stehen die Risiken gegenüber, wie der Erfordernis der hochqualifizierten Entwicklern und der Problematik der Festsetzung von Lieferterminen und Lieferumfängen der Inkremente, die jedoch geklärt werden müssen. Würde während der Vertragsverhandlungen ein vernünftiger Umgang mit neuen oder geänderten Anforderungen geklärt werden, sodass jeder weiß, was neue oder geänderte Anforderungen für alle Beteiligen bedeuten, kann die agile Softwareentwicklung gute Resultate erzielen.

6 Literaturverzeichnis

[1] Mike Beedle; Martin Fowler; et al. Manifesto for Agile Software Development. http://agilemanifesto.org/, 2001. [Online; Stand 14. August 2014].

[2] Peter Hruschka; Chris Rupp; Starke Gernot. *Agility kompakt.* Spektrum Akademischer Verlag, 2009.

[3] Tobias Trepper. *Agil-systemisches Softwareprojektmanagement.* Springer Gabler Verlag, 2012.

[4] Mark Harwardt. Wasserfallmodell versus Scrum. http://www.fernuni-hagen.de/imperia/md/content/ps/masterarbeit-harwardt.pdf. [Online; Stand 12. Juni 2014].

[5] Kent Beck; Martin Fowler. *Planning Extreme Programming.* Addison-Wesley Professional, 2000.

[6] Eckhart Hanser. *Agile Prozesse: Von XP über Scrum bis MAP.* Springer-Verlag, 2010.

[7] Computerwoche. Agile Methoden im Vergleich. http://www.computerwoche.de/a/agile-methoden-im-vergleich,2352712, 2013. [Online; Stand 14. Juni 2014].

[8] Martin Seibert. Extreme Programming: Vorgehensmodell zur Software-Entwicklung bei //SEIBERT/MEDIA. http://blog.seibert-media.net/blog/2005/05/01/extreme-programming-vorgehensmodell-zur-software-entwicklung-bei-seibertmedia/, 2005. [Online; Stand 21. Juni 2014].

[9] Benedikt Franz. eXtreme Programming. `http://dme.rwth-aachen.de/en/system/files/file_upload/course/12/proseminar-methoden-und-werkzeuge/cameraready-extremeprogramming.pdf`. [Online; Stand 21. Juni 2014].

[10] E-Business Management. Agile Softwareentwicklung im Vergleich zu traditionellen Softwareentwicklungsprozessen. http://vfhebm.wordpress.com/2011/12/16/agile-softwareentwicklung-im-vergleich-zu-traditionellen-softwareentwicklungsprozessen/, 2011. [Online; Stand 21. Juni 2014].

[11] Hermann Götz; Sergij Paholchak. eXtreme Programming (XP). `http://www3.informatik.uni-wuerzburg.de/courses/vorl_06_ss/projman/daten/referate_new/eXtreme_Programming__XP_.pdf`. [Online; Stand 21. Juni 2014].

[12] it-agile GmbH. eXtreme Programming (XP). http://www.it-agile.de/wissen/methoden/extreme-programming/. [Online; Stand 21. Juni 2014].

[13] Computerwoche. Entwicklung: Agile - das Aus für Wasserfall? http://www.computerwoche.de/i/detail/artikel/2352228/1/943975/d2e274-media/, 2013. [Online; Stand 21. Juni 2014].