Vorgehensmodelle zur Projektentwicklung in der Informationstechnik

Ausarbeitung für die mündl. PRE Matura 2015

Samuel Hammer

5. Juni 2015

HTBLVA Spengergasse

Inhaltsverzeichnis

1 Einleitung

IT-Projekte können je nach konkretem Anwendungsfall unterschiedlich groß bzw. komplex werden. Um unabhängig von diesen Faktoren die Übersichtlichkeit, Organisation und Struktur eines Projektes zu gewährleisten, gibt es verschiedene Vorgehensmodelle. Die existierenden Vorgehensmodelle lassen sich grob in zwei Kategorien unterteilen:

- Klassische Phasenmodelle
- Vorgehensmodelle zur agilen Softwareentwicklung

Abhängig vom Anwendungsfall, können verscheidene Vorgehensmodelle verschiedene Vor- und Nachteile für das Projekt haben. In der Praxis gibt es also in den seltensten Fällen ein Modell, das für die Durchführung des Projektes hundertprozentig passt.

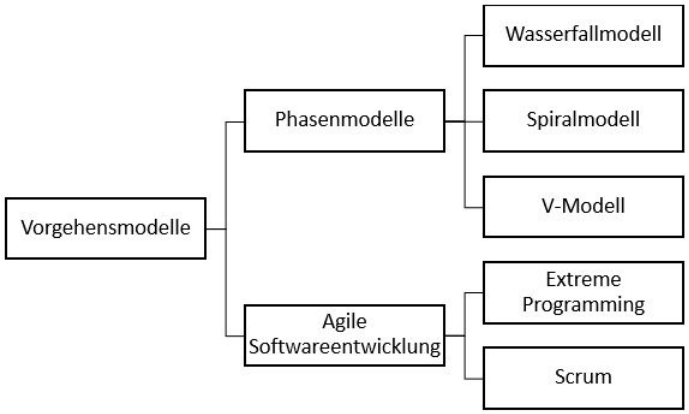

Abbildung 1.1: Übersicht über Vorgehensmodelle

In der folgenden Arbeit sollen die jeweiligen Vor- und Nachteile der verschiedenen Vorgehensmodelle erarbeitet und dokumentiert werden, sowie ein abschließender Vergleich inkl. Konklusion über das geeignetste Vorgehensmodell gezogen werden.

1.1 Phasenmodelle

Phasenmodelle sind ßtandardisierte Projektstruckturen für die Erstellung des Projektprodukts".[9] Da allerdings nicht auf jedes Projekt die selben Phasen zutreffen, gibt es verschieden Abwandlungen. Für Softwareprojekte wird in der Regel folgende Phasenstruktur verwendet:

1. Anforderungen
2. Analyse
3. Design
4. Entwicklung
5. Test

Wenn diese verschiedenen Phasen des Projektes komplett sequentiell und unabhängig voneinander stattfinden, so spricht man vom klassischen Wasserfallmodell. (Siehe Kapitel 2 In der Praxis hat sich das strikte vorgehen von oben nach unten (also wie ein Wasserfall) allerdings nicht bewährt, da dieses Modell es nicht erlaubt einzelne Phasen zu wiederholen. Für diesen Fall wurden iterative Modelle entwickelt. Hier können dann einzelne Phasen wiederholt werden falls sich beispielsweise während des Projektes die Anforderungen verändern.

1.2 Agile Softwareentwicklung

Agile Softwareentwicklung gibt es seit den frühen 1990er Jahren. Mit steigender komplexität der Softwareprojekte, wurde auch der Aufwand in Bezug auf das Projektmanagement immer größer. Da klassische Phasenmodelle aufgrund ihrer geringen Flexibilität nicht mehr praktikabel waren, wurde das sogenannte Ägile Manifest"verfasst. Das Agile Manifest ist ein Werk, in dem Agile Werte definiert werden, um Softwareprojekte einfacher und besser zu gestalten. Ein Auszug aus dem Agilen Manifest besagt:

"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:

- **Menschen 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."[3]

2 Wasserfallmodell

Das Wasserfallmodell ist eines der ältesten linearen Vorgehensmodellen. Es ist nicht iterativ, d.h. wenn man nach dem Wasserfallmodell vorgeht, ist es prinzipiell nicht möglich einzelne Projektphasen zu wiederholen. Ursprünglich stammt das Wasserfallmodell aus der Bau- und Produktionsbranche. In anbetracht der Tatsache, dass zum damaligen Zeitpunkt noch kein eigenes brauchbares Modell zur Softwareentwicklung existierte, wurde das Wasserfallmodell schlicht und einfach für IT-Projekte adaptiert.

Bei diesem Modell muss erst jede Phase abgeschlossen werden, bevor zur nächsten übergegangen werden kann. Daraus resultiert eine starke Unflexibilität, was das Wasserfallmodell vor allem bei Softwareprojekten sehr umständlich und beinahe unbrauchbar macht. Wenn beispielsweise in einem Projekt bereits mit der Programmierung begonnen wurde und der Kunde im Nachhinein eine Änderung an der Spezifikation vornehemen möchte, ist dies laut Wasserfallmodell nicht mehr möglich. Vor allem bei langläufigen

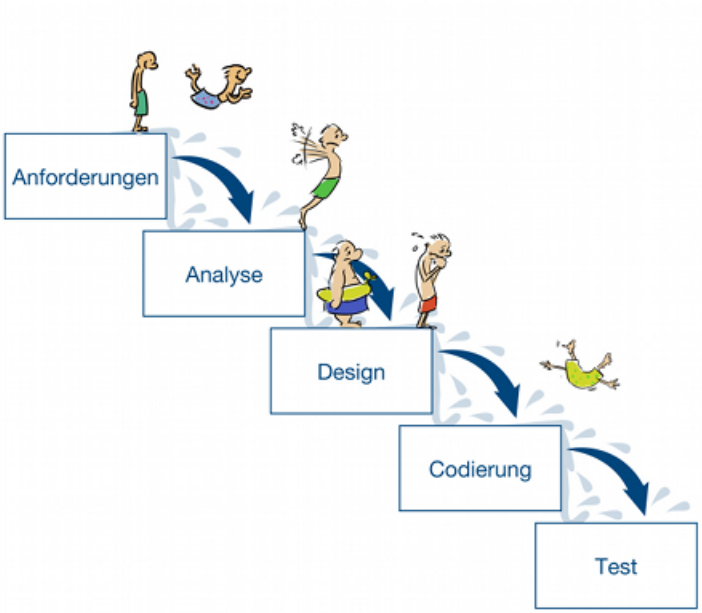

Abbildung 2.1: Das Wasserfallmodell[17]

Projekten ist der Einsatz des Wasserfallmodells sehr problematisch, da sich über lange Zeiträume oft die technischen Gegebenheiten und Umgebungsbedingungen ändern

können. Sollte bei weit fortgeschrittenem Projektverlauf ein Fehler in einer früheren Phase entdeckt werden, ist nicht selten ein Projektabbruch die einzige Möglichkeit den Schaden einigermaßen zu begrenzen.

Um die oben genannten Probleme aus der Welt zu schaffen, oder zumindest zu verbessern wurde das Erweiterte Wasserfallmodell nach Royce entwickelt. Bei dieser Version

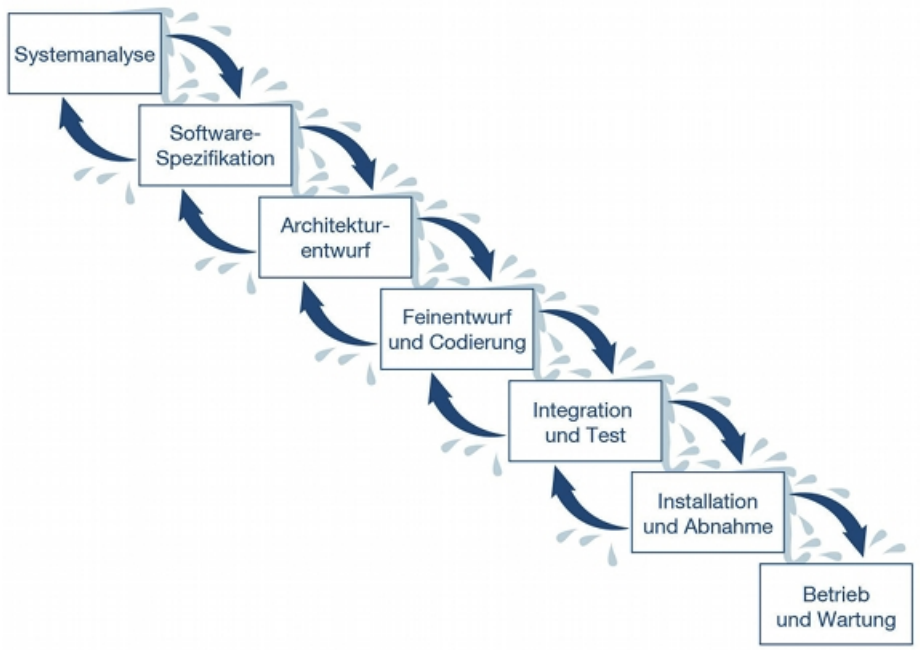

Abbildung 2.2: Das Erweiterte Wasserfallmodell nach Royce[17]

des Wasserfallmodells sind Rücksprünge erlaubt. Nichtsdestotzotz müssen nach einem Rücksprung auf eine vorhergehende Phase, alle danach folgenden Phasen wiederholt werden. Diese Änderung verbessert das Modell zwar drastisch, allerdings werden dadurch noch lange nicht alle grundlegenden Fehler ausgemerzt.

2.1 Vorteile des Wasserfallmodells

Da das Wasserfallmodell ein sehr altes und rudimentäres Vorgehensmodell ist, hat es, auf heutige Projekte bezogen, nicht viele Vorteile.
Ein entscheidender Vorteil ist allerdings die Einfachheit des Modells. Es lässt sich in der Praxis sehr leicht auf ein Projekt anwenden und vermittelt schnell und einfach den Eindruck von Übersicht und Kontrolle. Entgegen aller Kritik, kann das Wasserfallmodell also durchaus sinnvoll sein, solange sich das Projekt im sehr kleinen Rahmen bewegt.

2.2 Nachteile des Wasserfallmodells

Wie bereits in Kapitel 2 ausführlich erklärt, ist das Wasserfallmodell aufgrund der fehlenden flexibilität und der Tatsache, dass der gesamte Projektverlauf linear sein muss, in der Praxis für heutige Projekte nahezu unbrauchbar. Werden Fehler erst spät im Projektverlauf entdeckt, führt das zu sehr kostspieligen Änderungen. Des weiteren tritt durch die lange Laufzeit eines solchen Projektes, der soginannte "Return On Investment"[1]erst sehr spät nach Beginn des Projektes ein.

Auch ist es beim Wasserfallmodell oft schwierig, die einzelnen Phasen voneinander abzugrenzen. Übergänge die in der Theorie klar definiert sind, können in der Praxis oftmals eher fließend sein.

[1] "Der Begriff Return on Investment (...) ist eine betriebswirtschaftliche Kennzahl zur Messung der Rendite einer unternehmerischen Tätigkeit, gemessen am Gewinn im Verhältnis zum eingesetzten Kapital."[10]

3 Spiralmodell

Das Spiralmodell wurde 1986 von Barry W. Boehm entwickelt. Es ist ein risikogetriebenes Vorgehensmodell, welches auf dem Wasserfallmodell basiert, sich allerdings der Idee der Iteration bedient. Im Gegensatz zum Wasserfallmodell, werden alle Phasen immer wieder wiederholt, was eine kontinuierliche Verbesserung des Projektes erlaubt. Durch die Spiralform des Modells, wird ein wiederholtes Eingreifen durch das Management in jedem Zyklus gewährleistet. Durch die Anwendung dieses Vorgehensmodells, kann ein Projekt auch erfolgreich abgeschlossen werden, wenn sich Ziele oder Spezifikation währenddessen verändern.

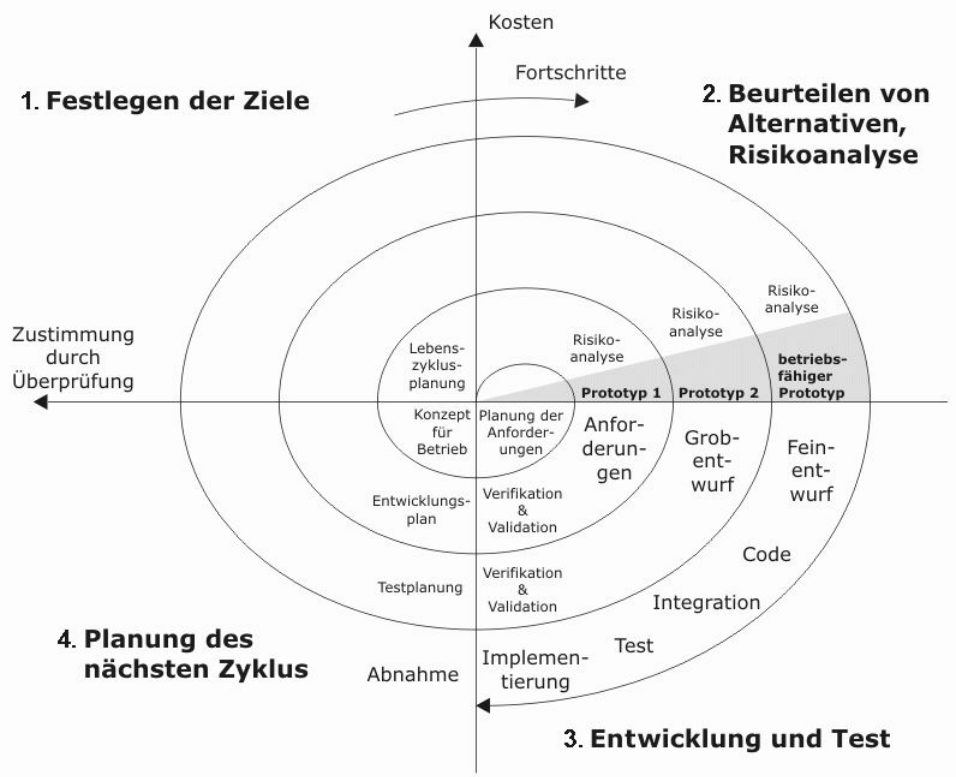

Abbildung 3.1: Das Spiralmodell[13]

Wie in Abbildung 6.1 zu sehen ist, bewegt sich die Linie (Zeitfortschritt) vom Ursprung aus nach außen. Jede Umdrehung um den Koordinatenursprung weist dabei einen vollendeten Zyklus auf. Während jedes Zyklus wiederholen sich immer die selben 4 Quadrandten:

1. Festlegung der Ziele
2. Beurteilen von Alternativen und Risikoanalyse
3. Entwicklung und Test
4. Planung des nächsten Zyklus

Ein Punkt, in welchem sich das Spiralmodell wesentlich vom Wasserfallmodell unterscheidet, ist die Risikobetrachtung. Zunächst werden Risiken analysiert und bewertet. In weiterer Folge wird versucht das gefährlichste Risiko zu identifizieren und zu beseitigen. Durch die Spiralform, wird diese Risikoanalyse zyklisch wiederholt. Daher können neu auftretende Risiken in der nächsten Iteration wieder erkannt und beseitigt werden. Scheitert die Risikobeseitigung in irgendeinem Zyklus, so gilt das Projekt als gescheitert. Sind am Ende des Projektes alle Risiken beseitigt und das Produkt einwandfrei, so wird das Projekt erfolgreich abgeschlossen betrachtet.

Eine zentrale Rolle beim Spiralmodell spielt auch die Erstellung von Prototypen. Durch die wiederholte Erstellung bzw. Erneuerung des Prototypen, können Risiken frühzeitig erkannt werden. Der Prototyp wird im Laufe des Projektes kontinuierlich weiterentwickelt, sodass dieser bis zum finalen Produkt immer weiter verfeinert wird.

3.1 Vorteile des Spiralmodells

Ein wesentlicher Vorteil des Spiralmodells im Vergleich mit anderen Vorgehensmodellen, ist die Tatsache, dass es risikogetrieben ist. Durch die regelmäßigen Risikoanalysen können etwaige Fehler schon im Vorhinein vermieden werden, was im Endeffekt sowohl Kosten-, als auch Zeitsparend ist. Auch die Tatsache, dass man durch das Prototyping schon sehr früh ein vorzeigbares Produkt zur Verfügung hat, macht das Spiralmodell zu einem brauchbaren Werkzeug für die Entwicklung von Softwareprojekten.

3.2 Nachteile des Spiralmodells

Einer der zentralen Aspekte, welche das Spiralmodell ausmachen, ist gleichzeitig ein großer Nachteil; der Prototyp. Werden beim ersten Prototyp Fehler im Design gemacht, so ist es im Nachhinein schwierig diese noch zu beheben und es besteht die Gefahr, dass das Endprodukt nicht zufriedenstellend ist. Durch die Komplexität und, immer wiederkehrenden Risikoanalysen und den fortlaufenden Prototypen, ist das Spiralmodell auch mit einem verhälnismäßig sehr hohen Managementaufwand verbunden.

4 V-Modell

Das V-modell ist ein weiteres, auf dem Wasserfallmodell aufbauendes, Vorgehensmodell in der Softwareentwicklung. Das Projekt wird bei diesem Modell in Phasen aufgeteilt, daher zählt es ebenfalls zu den oben beschriebenen Phasenmodellen. (Siehe Kapitel 1.1) Da das V-Modell allerdings neben Entwicklungsphasen auch Phasen zur Qualitätssicherung enthält, unterscheided es sich deutlich von anderen Phasenmodellen, wie dem Wasserfall- oder Spiralmodell.

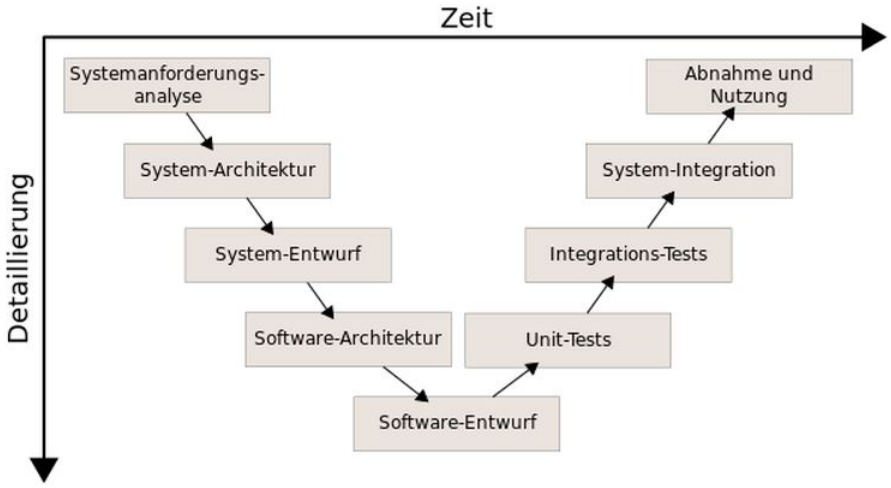

Abbildung 4.1: Das V-Modell[14]

Wie beim Wasserfallmodell, ist das Ergebnis einer Phase eine bindende Vorgabe für die nächsttiefere bzw. nächsthöhere Projektphase. Den Entwicklungsphasen auf der linken Seite stehen die jeweils zugehörigen Testphasen auf der rechten Seite gegenüber. Durch diese Gegenüberstellung wird eine sehr hohe Testabdeckung gewährleistet, da zu jeder Entwicklungsphase genau eine Testphase gehört.

4.1 Vorteile des V-Modells

Das V-Modell ist ein äußerst umfassendes Vorgehensmodell. Es integriert viele Aspekte des Entwicklungsprozesses und ist daher leicht erweiterbar und anpassbar. Durch die klare Struktur der Phasen ist das Modell leicht verständlich und lässt sich sehr einfach auf ein Softwareprojekt anwenden.

4.2 Nachteile des V-Modells

Ein entscheidender Nachteil des V-Modells ist, dass es zu allgemein und generisch ist, um es auf hochkomplexe Softwareprojekte anzuwenden. Obwohl es sehr einfach verständlich ist, ist es nicht für Projekte im kleinen Rahmen geeignet, da diese meist nicht über die notwendigen Ressourcen verfügen die vorgegebenen Phasen auch einzuhalten.

5 Extreme Programming

Extreme Programming[1] ist ein agiles Vorgehensmodell in der Softwareentwicklung, bei dem das Lösen von konkreten programmiertechnischen Aufgaben im Vordergrund steht. Dabei kommt der Formalen Komponente des Projektes weniger Bedeutung zu, was XP zu einer eher umstrittenen Methode macht.

Das Ziel von XP ist es, das Produkt nicht vollständig zu einem späteren Zeitpunkt in der Zukunft auszuliefern, sondern viel mehr die Features genau dann auszuliefern, wenn der Kunde sie benötigt. Es werden von Anfang an Tests durhgeführt, was ein konstantes Feedback gewährleistet und dem Team so ermöglicht Fehler früh zu erkennen und zu beheben. Auch die Kommunikation mit dem Kunden spielt eine zentrale Rolle bei XP, so soll möglichst schnell auf Kundenwünsche oder etwaige Change Requests reagiert werden.

XP definiert fünf zentrale Werte, welche zur erfolgreichen Ausführung eines Softwareprojektes führen sollen:

- Einfachheit
- Kommunikation
- Feedback
- Respekt
- Mut

Das sind abstrakte Werte, die als Orientierung für alle Beteiligten am Projekt gelten.

Des Weiteren gibt es 14 Prinzipien, die eine Brücke zwischen den Abstrakten Werten von XP und konkret annwendbaren Praktiken bilden sollen. Diese sind:

Menschlichkeit, Wirtschaftlichkeit, Beidseitiger Vorteil, Selbstgleichheit, Verbesserungen, Vielfältigkeit, Reflexion, Lauf, Gelegenheiten wahrnehmen, Redundanzen vermeiden, Fehlschläge hinnehmen, Qualität, Kleine Schritte und Akzeptierte Verantwortung.[2]

Auf diese Prinzipien wird bei der Verwendung von XP viel Wert gelegt. Sie müssen auf jedenfall beachtet und bei der Entwicklung berücksichtigt werden.

[1] In weiterer Folge nur noch als XP bezeichnet
[2] Alle sinngemäß übersetzt

All diese abstrakten Werte und Prinzipien bringen allerdings wenig, wenn sie nicht in konkreten Praktiken umgesetzt werden. Daher wurden für XP einige Praktiken, sozusagen "best practices", definiert, die in der Realität der Softwareentwicklung angewendet werden können.

5.1 XP Praktiken

5.1.1 Pair-Programming

Beim Pair-Programming arbeiten immer zwei Programmierer gleichzeitig an einem konkreten Problem. Dabei ist einer derjenige der tatsächlich Programmiert (der "Driver") während der Andere eine Kontrollfunktion innehat bzw. einen gesamtüberblick über den geschriebenen Code haben sollte (der "Partner"). Dadurch wird der Wissenstransfer gefördert und Anfänger können sich leichter in ein Projekt einarbeiten. Des Weiteren wird die Fehlerdichte erheblich verringert, da ein ständiges Code Review stattfindet.

5.1.2 Kollektives Eigentum

Es soll verhindert werden, dass einzelne Programmierer einen Teil des Projekts als ihr Eigentum ansehen. Daher werden konkrete Aufgabenstellungen zuerst an das gesamte Team gestellt. Durch Pair-Programming wechselnde Aufgabengebiete, soll dies verhindert werden.

5.1.3 Permanente Integration

Es sollen einzelne Komponenten, and denen unter Umständen separat voneinander entwickelt wird, regelmäßig und kontinuierlich zu einem Großen Gesamtsystem zusammengefügt werden. Dadurch sollen Fehler früh erkannt werden. Dieser Schritt geschieht bei XP in der Regel täglich.

5.1.4 Test-Driven-Development (TDD)

Bei testgetriebener Entwicklung geht es darum, die Unit-Tests für eine bestimmte Funktion VOR der eigentlichen Implementierung zu schreiben. Mit Hilfe dieser Technik soll verhindert werden, dass der Entwickler das eigentliche Ziel aus den Augen verliert.

5.1.5 Kundenbeziehungen

XP basiert in seiner Gesamtheit auf einer engen Beziehung zwischen dem Kunden und dem Projektteam. Der Kunde das Iterationsziel in Form von User-Storys vor und führt nach der Fertigstellung selbst Akzeptanztests durch. Eine Vorraussetzung für diese enge Zusammenarbeit ist, dass der Kunde immer für das Projektteam erreichbar sein muss und umgekehrt.

5.1.6 Refactoring

Der Source Code soll laufend verbessert werden. Das geschieht durch wiederkehrende Code Reviews und laufendes Refactoring. XP distanziert sich vom Anspruch an Code, welcher von Anfang an perfekt ist. Dies wird allerdings durch laufende Fehlersuche und Fehlerbehebungen ausgeglichen.

5.1.7 Keine Überstudnen

Überstunden sollen vermieden und stattdessen die normale 40-stunden-Woche eingehalten werden, da bei Überstunden auf Dauer die Konzentrationsfähigkeit und Freude an der Arbeit verloren gehen.

5.1.8 Iterationen

Iterationen sollen kurz und in regelmäßigen Abständen fertiggestellt werden. Dadurch kann dem Kunden auf regelmäßiger Basis ein lauffähiges Produkt vorgezeigt werden. Dieses Vorgehen erlaubt schnelle Feedbackschleifen und direktes reagieren auf Kundenwünsche bzw. Fehler.

5.1.9 Metapher

Bei traditionell ausgeführten Softtwareprojekten gibt es oft eine hohe Diskrepanz zwischen der Fachsprache des Auftraggebers und der des AUftragnehmers. Um Missverständnissen vorzubeugen werden deshalb die Anforderungen des Kunden von diesem selbst als sogenannte User-Storys beschrieben. In diesen User-Storys werden die Anforderungen als verständliche, mit Metaphern versehene Alltagsgeschichten definiert.

5.1.10 Coding-Standards

Alle Programmierer in einem Team sollten sich an die gleichen Coding-Standards halten. Durch das Einsetzen von vielen Programmierern in immer wechselnden Bereichen, ist es nur durch solche Standars möglich eine produktive Zusammenarbeit zu erreichen.

5.1.11 Einfaches Design

Nach dem Motto "Keep it small and simple"wird das Design des Softwareprojektes bewusst einfach gehalten. Es wird im Zuge dessen auch auf vorbereitende Maßnahmen für eventuelle spätere Funktionen verzichtet.

5.1.12 Planning Game

Neue Versionen der Software werden in einem Planning-Game, auch als Planning-Poker bekannt, spezifiziert und der Aufwand zu deren Umsetzung abgeschätzt. An diesem iterativen Prozess sind sowohl Entwicklungsmannschaft als auch Kunde beteiligt.[4]

6 Scrum

Scrum wird per Definition nicht als Vorgehensmodell, sondern als Framework bezeichnet. Das kommt daher, dass Scrum nur einige wenige Regeln definiert und nicht eine komplette Vorgehensweise. Scrum basiert auf der Annahme, dass die meisten komplexen Softwareprojekte zu kompliziert und unvorhersehbar sind, um sie von Vorhinein in einen kompletten Plan gefasst zu werden. Weiters beruht Scrum darauf, dass nicht nur das Produkt kontinuierlich weiterentwickelt wird, sondern auch die langfristige Planung.[1] Detailpläne werden nur für den jeweiligen nächsten Zyklus[2] erstellt. Durch diese Vorgehensweise soll die Projektplanung auf das wesentliche fokussiert werden. Das Scrum

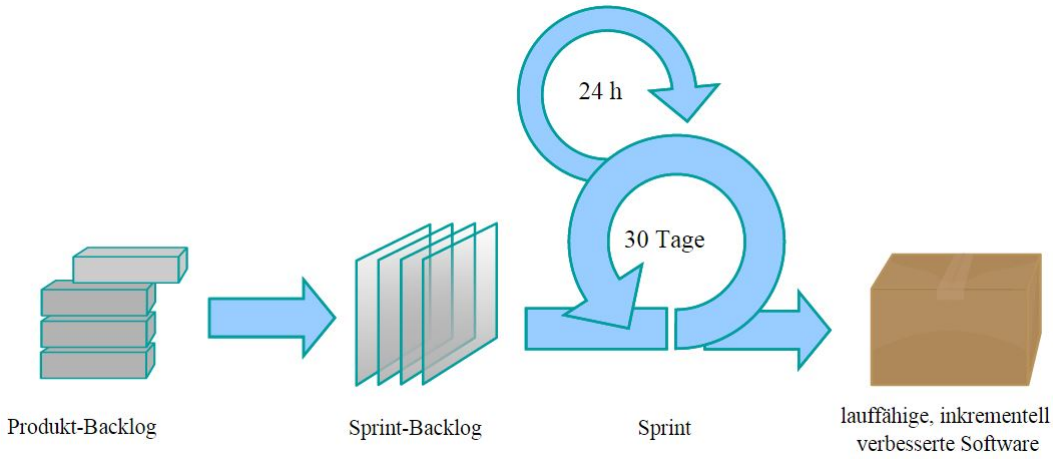

Abbildung 6.1: Der Scrum Prozess[11]

Framework definiert 3 Rollen, 3 Artefakte und 5 Aktivitäten.

6.1 Scrum Rollen

Es gibt 3 Scrum Rollen: Product Owner, Entwicklungsteam und Scrum Master. Die Gesamtheit dieser drei Rollen bezeichnet man als Scrum Team.

6.1.1 Product Owner

Der Product Owner ist sozusagen der Koordinator des gesamten Projekts. Er ist für den wirtschaftlichen erfolg des Projekts und die Ausführung im generellen verantwort-

[1] Das sogenannte "Product Backlog"
[2] Den Sprint

lich. Er legt fest welche Funktionen im jeweiligen Spring implementiert werden sollen und priorisiert diese. Der Product Owner ist außerdem für das Product Backlog verantwortlich und aktualisiert dieses auf regelmäßiger Basis. Außerdem ist er die zentrale Kommunikationsschnittstelle zwischen Auftraggeber und Auftragnehmer.

6.1.2 Entwicklungsteam

Das Entwicklungsteam ist dafür verantwortlich, die im Produkt Backlog bzw. in weiterer Folge im Sprint Backlog spezifizierten Funktionen umzusetzen. Das Entwicklungsteam organisiert sich dabei eigenständig. Die Entscheidung wie und in welcher Reihenfolge die Backlogeinträge umgesetzt werden obliegt einzig und allein den Mitgliedern des Entwicklungsteams. Entwicklungsteams haben meist eine Größe von drei bis 9 Mitgliedern, wobei allerdings darauf geachtet wird das Team möglichst klein zu halten, da sonst der Koordinationsaufwand steigt.

6.1.3 Scrum Master

Der Scrum Master kümmert sich um die fehlerfreie Ausführung des gesamten Scrum-Prozesses. Er arbeitet eng mit dem Entwicklungsteam zusammen, gehört diesem jedoch selbst (meistens) nicht an. Der Scrum Master ist nicht der Vorgesetzte des Entwicklungsteam, er versteht sich vielmehr als Coach, welcher dem Team bei Problemen diverser Art beisteht und zur Behebung dieser beiträgt. Zu den Aufgaben des Scrum Masters gehören auch das moderieren von Treffen und die Überprüfung der Einhaltung der Scrum Regeln.

6.2 Artefakte

6.2.1 Product Backlog

Unter Product Backlog wird die Auflistung aller vom Kunden gestellten Anforderungen an das Projekt verstanden. Es ist dynamisch, wird also nicht nur einmal erstellt und dann abgearbeitet sondern wird auf regelmäßiger Basis weiterentwickelt. Der Product Owner ist für das Product Backlog der alleinige verantwortliche. Er kümmert sich unter Anderem auch um die Priorisierung der Backlogeinträge. Die Kundenanforderungen im Product Backlog werden soweit möglich nicht in technischner Fachsprache, sondern anwendungsorientiert und fachlich als User Stories definiert.

6.2.2 Sprint Backlog

Der Sprint Backlog ist, ähnlich wie der Product Backlog, eine Auflistung der Anforderungen. Im Sprint Backlog werden allerdings nur einige wenige Einträge des Product Backlogs übernommen, die für den aktuellen Sprint anfallen.

6.2.3 Product Increment

"Das Inkrement ist die Summe aller Product-Backlog-Einträge, die während des aktuellen und allen vorangegangenen Sprints fertiggestellt wurden. Am Ende eines Sprints muss das neue Inkrement in einem nutzbarem Zustand sein und der Definition of Done entsprechen."[11]

6.3 Aktivitäten

6.3.1 Sprint Planning

Im Sprint Planning spielen zwei essentielle Fragen eine Rolle:

- Was kann im kommenden Sprint entwickelt werden?
- Wie wird die Arbeit im kommenden Sprint erledigt?

Das Sprint Planning dauert in Summe maximal 2 stunden pro Sprint Woche. Der Product Owner stellt dem Entwicklungsteam die definierten einträge des Product Backlogs vor. Zunächst arbeitet das gesamte Team daran, die festgelegten Backlogeinträge zu verstehen und die Arbeit einzuteilen. Es werden auch die Akzeptanzkriterien festgelegt, welche am Ende des Sprints darüber entscheiden ob das neue Inkrement der "Definition of Doneëntspricht, also fertig ist. In weiterer Folge wird der Aufwand abgeschätzt und die Anzahl der Product Backlog Einträge, welche in diesem Sprint erledigt werden können wird festgelegt. Über die Reihenfolge und Priorisierung der Einträge entscheidet dabei der Product Owner alleine. In weiterer Folge wird dann im Detail geplant, welche Tasks zur Erfüllung der Product Backlog Einträge notwendig sind. Das Ergebnis des Sprint Plannings ist der Sprint Backlog

6.3.2 Daily Scrum

Der Sinn des Daily Scrum ist, dass sich alle Teammitglieder einmal am Tag zu einem kurzen, maximal 15 Minuten langen, Meeting zusammenfinden. Dabei sollen allerdings nicht Probleme und auftretende Bugs besprochen werden, sondern ein aktiver Informationsaustausch stattfinden. Jedes Teammitglied gibt kurz und bündig eine Übersicht über den Status der aktuell bearbeiteten Tasks. so weiß jedes Mitglied des Entwicklungsteams immer genau über den Status der anderen Mitglieder bescheid. Product Owner und Scrum Master sind beim Daily Scrum zwar häufig anwesend, beteiligen sich aber nicht aktiv am Gespräch.

6.3.3 Sprint Review

Das Sprint Review findet am Ende eines jeden Sprints statt. Dabei wird überprüft ob das Team die zu Beginn im Sprint Planning festgelegten Ziele erreicht wurden. Am Sprint Review nehmen zusätzlich zum Scrum Team auch die Steakholder Teil und es wird besprochen wie die weitere Vorgehensweise aussieht. Weiters ist es von Vorteil

wenn auch Kunden und Anwender an diesen Besprechungen teilnehmen, da diese die Funktionalität bzw. bedienbarkeit des fertigen Inkrements am besten beurteilen können.

6.3.4 Sprint Retrospektive

Die Sprint Retrospektive findet, ebenso wie das Sprint Review, am Ende eines Sprints statt. Im Gegensatz zum Sprint Review, nehmen daran allerdings nur Mitglieder des Scrum Teams teil. Steakholder können in manchen fällen nur auf Einladung dazu kommen. Das Team überprüft seine Arbeitsweise auf kritische Art und Weise, daher müssen auch unangenehmen Wahrheiten offen geäußert werden. In weiterer Folge werden dann Verbesserungsvorschläge festgelegt und dokumentiert.

6.3.5 Product Backlog Refinement

Als Product Backlog Refinement bezeichnet man den laufenden Prozess, bei dem der Product Owner zusammen mit dem entwicklungsteam den Product Backlog akutalisiert bzw. weiterentwickelt. Es gibt eine Reihe von Aktivitäten, welche zum PBR gehören, unter Anderem:

- Ordnen von Einträgen
- Löschen von einträgen die nicht mer benötigt werden
- Hinzufügen von neuen Einträgen
- Planung von Releases
- Detaillieren von Einträgen

Da die Steakholder Wertvolle Informationen und Anmerkungen für das Produkt bzw. das Product Backlog liefern können, finden meistens regelmäßig auch sogenannte Product-Backlog-Refinement-Meetings zusammen mit ausgewählten Stakeholdern statt.

7 Feature Driven Development

Feature Driven Development[1] ist ein Vorgehensmodell, dass auch zur agilen Softwareentwicklung. Es ist ein Set an Arbeitstechniken, Strukturen, Rollen und Methoden für das Projektmanagement von Software Projekten.

FDD wurde ursprünglich als schlanke, einfache Methode ein zeitkritisches Softwareprojekt durchzuführen. Dabei soll der Begriff "Featureïm Mittelpunkt. Jedes Feature das entwickelt wird, stellt einen Mehrwert für das Projekt und infolgedessen auch für den Kunden dar. Die Planung findet anhand eines Feature-Plans statt.

Bei FDD gibt es 2 wichtige Rollen:

- **Der Chefarchitekt (Chief Architect)**
  Er behält den Überblick über die gesamtarchitektur des Projekts und die fachlichen Kernmodelle.
- **Der Chefprogrammierer (Chief Programmer)**
  Er führt das Entwicklerteam an und behält die Übersicht über die Aufgaben der einzelnen Programmierer. Bei größeren Projekten können auch mehrere Entwicklerteams mit je einem Chefprogrammierern.

7.1 Vorgehensweise

FDD definiert füf zentrale Prozesse: Die Prozesse 1-3 werden innerhalb weniger Tage

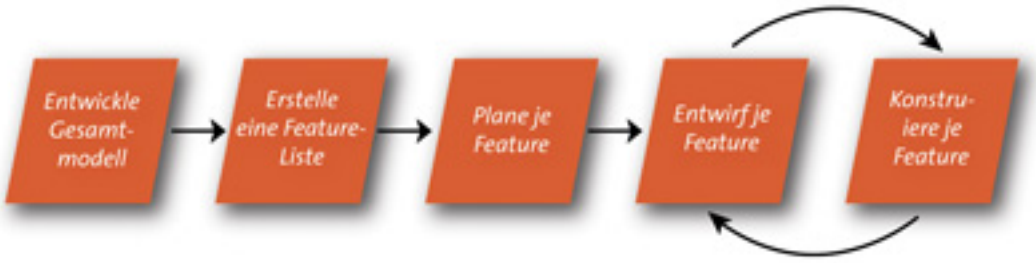

Abbildung 7.1: Die FDD Prozesse[5]

durchgeführt, 4 und 5 werden allerdings ständig wiederholt

[1] Abk.: FDD

8 Simultaneous Engineering

9 Rational Unified Process

10 aufgabenstellungen

10.1 Frage 1

Ihre Firma möchte ein Softwareprojekt im kleinen Rahmen beginnen, als ranghoher Mitarbeiter mit Erfahrung im Projektmanagement, wurden Sie damit beauftragt ein passendes Vorgehensmodell auszuwählen. Geben Sie einen Überblick über die gängigsten Vorgehensmodelle und stellen Sie insbesondere den Nachteil klassischer Phasenmodelle dar.

10.2 Frage 2

Sie arbeiten als Projektleiter an einem Softwareprojekt, welches zum jetzigen Zeitpunkt mit dem klassischen Wasserfallmodell durchgeführt wird. Sie empfehlen schon seit längerem den Umstieg auf Scrum, Ihr vorgesetzter weigert sich jedoch diesen Aufwand umzusetzen. Stellen Sie den Vorteil von agiler Softwareentwicklung in Form von Scrum im Gegensatz zum Wasserfallmodell dar.

10.3 Frage 3

Literatur

[1] *Agile Softwareentwicklung*. Abgerufen am: 27.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Agile_Softwareentwicklung.

[2] Roland Baldenhofer. *Kleine Erklärung zum Spiralmodell*. Abgerufen am: 29.05.2015. Guggat Emol Blog. 2009. URL: http://www.baldenhofer.eu/blog/it/kleine-erklaerung-zum-spiralmodell/.

[3] Kent Beck u. a. *Manifesto for Agile Software Development*. 2001. URL: http://www.agilemanifesto.org/.

[4] *Extreme Programming*. Abgerufen am: 31.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Extreme_Programming.

[5] *Feature Driven Development*. Abgerufen am: 05.06.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Feature_Driven_Development.

[6] Marco Kuhrmann. *Spiralmodell*. Abgerufen am: 29.05.2015. Enzyklopädie der Wirtschaftsinformatik. URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Vorgehensmodell/Spiralmodell.

[7] Peter Lorenz. *Agile Softwareentwicklung*. HTBLVA Spengergasse, 2008.

[8] Peter Lorenz. *Prozessmodelle*. HTBLVA Spengergasse, 2003.

[9] *Projektphase*. Abgerufen am: 27.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Projektphase.

[10] *Return On Investment*. Abgerufen am: 28.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Return_on_Investment.

[11] *Scrum*. Abgerufen am: 02.06.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Scrum.

[12] *Spiral-Modell in der Softwareentwicklung*. Abgerufen am: 29.05.2015. Informatik Forum Simon. 2014. URL: http://www.infforum.de/themen/anwendungsentwicklung/se-spiral-modell.htm.

[13] *Spiralmodell*. Abgerufen am: 28.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Spiralmodell.

[14] *V-Modell*. Abgerufen am: 29.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/V-Modell.

[15] *Vorgehensmodell zur Softwareentwicklung*. Abgerufen am: 27.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Vorgehensmodell_zur_Softwareentwicklung.

[16] *Vorgehensmodelle und standardisierte Vorgehensweisen*. Abgerufen am: 29.05.2015. techSphere. URL: http://www.techsphere.de/pageID=pm03.html.

[17] *Wasserfall-Modell*. Abgerufen am: 28.05.2015. Scrum-kompakt.de. URL: http://www.scrum-kompakt.de/grundlagen-des-projektmanagements/wasserfall-modell/.

[18] *Wasserfallmodell*. Abgerufen am: 28.05.2015. Wikipedia. URL: http://de.wikipedia.org/wiki/Wasserfallmodell.

[19] Don Wells. *Extreme Programming: A gentle introduction*. Abgerufen am: 31.05.2015. 2013. URL: http://www.extremeprogramming.org/.

[20] Frank Westphal. *Extreme Programming*. Abgerufen am: 31.05.2015. 2001. URL: http://www.frankwestphal.de/ExtremeProgramming.html.

[21] Harald Wittek. *Projektentwicklung 5. Semester Projektphasenmodelle*. HTBLVA Spengergasse.

Abbildungsverzeichnis