Seminararbeit

Agile vs. klassische Entwicklungsmethoden

Seminar Software Engineering

WS 03/04

vorgelegt von

Michael Kauflin
Winterthur

Angefertigt am
Institut für Informatik
der Universität Zürich

Prof. Dr. M. Glinz

Betreuer: Ch. Seybold

Präsentation am 21.10.2003

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Literatur- und Quellenverzeichnis

[1] Fowler, M.: *The New Methodology*, 2003, http://martinfowler.com/articles/newMethodology.html
[2] Glinz, M.: *Software Engineering I*, 1999, Zürich
[3] Cockburn, A.: *Agile Software Development*, 2002, Addison Wesley
[4] http://xprogramming.com/xpmag/whatisXP.htm
[5] Alexander, C.: *The Timeless Way Of Building*, 1979, Oxford University Press, New York
[6] Cunningham, W.: *Episodes: A Pattern Language Of Competitive Development*. In: Vlisses, J. (Hrsg.): *Pattern Languages Of Program Design 2*, 1996, Addison-Wesley, New York
[7] Jacobsen, I., M. Christerson, P. Jonsson, G. Overgard: *Object-Oriented Software Engineering: A Use-Case Driven Approach*, 1994, Reading MA, Addison-Wesley
[8] Gilb, T.: *Principles Of Software Engineering Management*, 1988, Workingham, UK, Addison Wesley
[9] Takeuchi, H., I. Nonaka: *The New Production Development Game*, 1986, Harvard Business Review Jan/Feb.: 137-146
[10] Hunt, A., D. Thomas: *The Pragmatic Programmer*, 2000, Addison-Wesley
[11] http:// www.agilealliance.org
[12] http:// www.xpuniverse.com
[13] http:// www.agiledevelopmentconference.com
[14] http://www.agilemodeling.com/essays/proof.htm
[15] Moore, G.: *Crossing the Chasm*, 2002, HarperCollins Publishers, New York
[16] Jones, C.: *Software Assessments, Benchmarks, and Best Practices* , 2000, Addison Wesley
[17] Highsmith, J.: *Agile Software Development Ecosystems*, 2002, Addison Wesley
[18] Williams, L., R. Kessler: *Pair Programming Illuminated*, 2002, Addison Wesley;
[19] Marchesi, M. et al: *Extreme Programming Perspectives*, 2002, Addison Wesley
[20] Larman, C.: *Agile and Iterative Development: A Manager's* Guide, 2003, Addison Wesley
[21] Abrahamson, P., O. Salo, J. Ronkainen, J. Warsta: *Agile Software Development Methods - Rewiew And Analysis*, 2002, VTT Publications 478

1 Einleitung

Diese Seminararbeit beschäftigt sich mit dem Vergleich von klassischen und agilen Methoden der Softwareentwicklung. Die Arbeit ist wie folgt gegliedert: im ersten Kapitel wird ein kurzer Überblick gegeben, im zweiten Kapitel werden die klassischen Methoden aufgeführt und erläutert, im dritten die agilen. Im vierten Kapitel folgen Erläuterungen zu Konferenzen und Organisationen der agilen Welt und im fünften Kapitel folgen abschliessende Erklärungen.

1.1 Methoden der Software-Entwicklung [1]

Die Entwicklung von Software ist von Natur aus eine chaotische Angelegenheit, meist nach dem Motto „code and fix“: es existiert kein Plan, wie die Software im Endzustand aussehen soll und das Design ist ein Flickwerk von vielen kurzsichtigen Entschlüssen. Das mag für kleine Systeme funktionieren (ein Entwickler und bis ca. 300 Codezeilen), doch bei grösseren Projekten (mehrere Entwickler und mehr als 300 Codezeilen) ist dieses Vorgehen zum Scheitern verurteilt. Es wird schwierig bis unmöglich, das System mit neuer Funktionalität auszustatten, Fehler und Defekte werden zunehmend schwerer zu beheben sein. Typisch für eine Software, die nach diesem Prinzip gefertigt wurde, ist eine lange Testphase, nachdem das System „fertig“ ist. Eine solche Testphase lässt sich nicht genau planen, da die Dauer für das Testen und die darauf folgende Fehlerbehebung nicht vorhersehbar sind.

Man hat lange mit dieser Art Software zu entwickeln gelebt, aber es gibt ein Gegenmittel gegen diese chaotischen Zustände: Methoden. Methoden zwängen der Software-Entwicklung einen disziplinierten Prozess auf mit dem Ziel, die Entwicklung besser vorhersehbar und effizienter zu machen. Dies geschieht mit einem detaillierten Prozess, der ein Schwergewicht auf die Planung legt, inspiriert von anderen Ingenieursdisziplinen (Bauingenieure etc). Man spricht deshalb auch von Ingenieur-Methoden (engl. engineering methodology).

Ein Prozess-Modell beschreibt den organisatorischen Rahmen für die Software-Entwicklung mit

- Reihenfolge und Phasen des Arbeitsablaufes
- jeweils durchzuführende Aktivitäten
- Definition der Teilprodukte und ihrer Anforderungen
- erforderliche Inputs und ihrer Anforderungen
- notwendige Mitarbeiterqualifikationen
- Verantwortlichkeiten und Konsequenzen
- anzuwendende Standards, Richtlinien, Methoden und Werkzeuge

Ingenieur-Methoden existieren schon eine lange Zeit, doch der voll durchschlagende Erfolg blieb ihnen verwehrt. Ausserdem sind sie unpopulär, da sie sehr bürokratisch sind und einen Haufen Papierkram verursachen, der die Geschwindigkeit der Entwicklung verlangsamen lässt.

Als Reaktion auf diese Methoden, die in Kapitel zwei beschrieben werden, ist eine Reihe von neuen Methoden in den letzten Jahren aufgetaucht. Eine Zeitlang waren diese Methoden unter dem Begriff „leightweight methodologies“ (leichtgewichtige Methoden) bekannt, aber der geläufige Ausdruck ist mittlerweile „agile Methoden“.

Für viele Leute ist das Auftauchen dieser neuen Methoden eine Reaktion auf die Bürokratie der Ingenieur-Methoden. Die neuen Methoden versuchen einen brauchbaren Kompromiss zwischen keinem Prozess (Chaos) und zu viel Prozess (Ingenieur-Methoden) zu erzielen, indem sie genau den richtigen Anteil an Prozess erwischen, um einen sinnvolles Resultat zu erhalten. Die agilen Methoden werden in Kapitel drei genauer erläutert.

2 Klassische Methoden

In diesem Kapitel werden die klassischen (Ingenieur-) Methoden aufgelistet und kurz erläutert. Die Methoden werden als bekannt vorausgesetzt (siehe Software Engineering I [2])

2.1 Wasserfall-Modell

Das Wasserfall-Modell wurde von Royce entwickelt und später vor allem von Boehm propagiert. Es ist das älteste systematische Prozessmodell für die Entwicklung von Software. In diesem Modell entspricht eine Phase einer Tätigkeit.

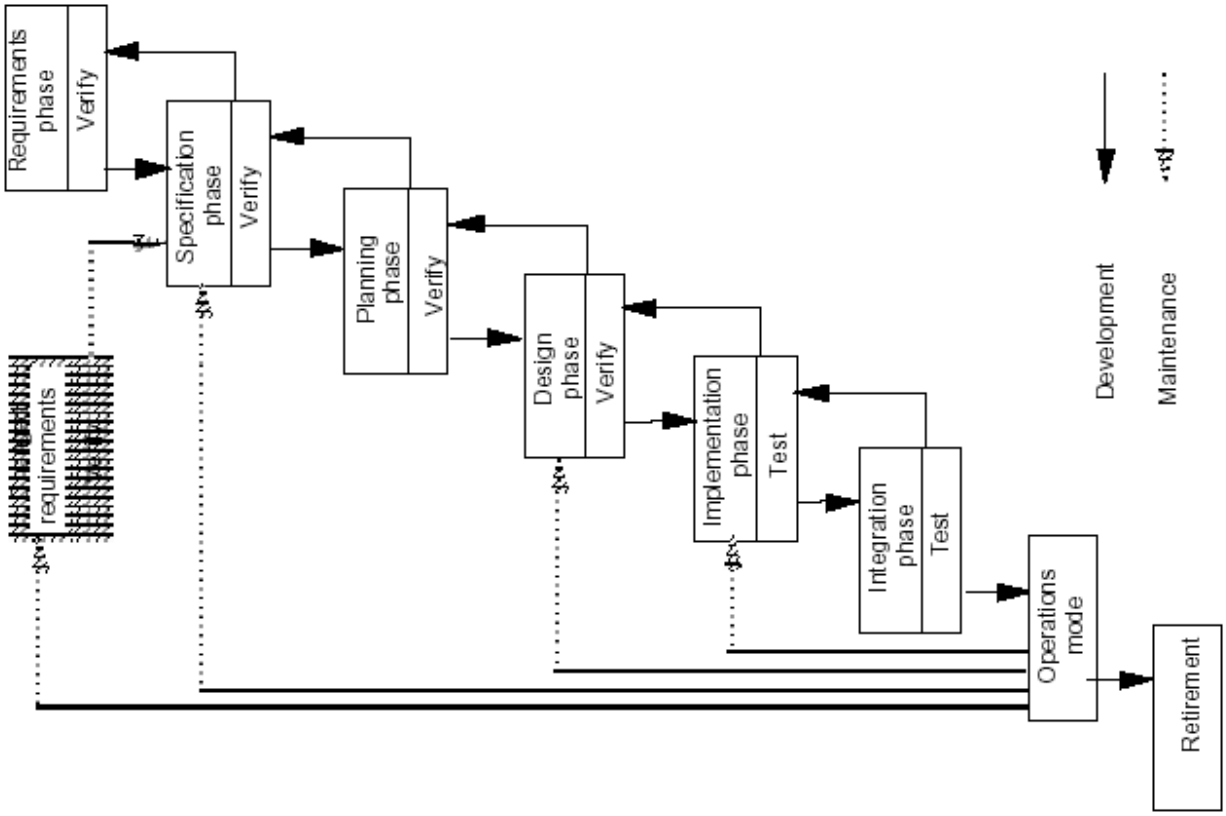

Bild 1: Wasserfallmodell

Vorteile	Nachteile
Einfachheit	Nichtbeachtung der Software-Evolution
Modellierung des natürlichen Lebenslaufes	erkannte Fehler werden nicht nur in der gegenwärtigen und der vorangegangenen Phase gemacht → Iterationen über mehrere Phasen → erschwerte Projektführung

Tabelle 1: Vor- und Nachteile des Wasserfallmodells

2.2 Ergebnisorientiertes Phasenmodell

Das ergebnisorientierte Phasenmodell hat wie das Wasserfallmodell den Software-Lebenslauf als Grundlage. Eine Phase ist jedoch keine Tätigkeit, sondern eine Zeitintervall.

Vorteile	Nachteile
Leicht verständlich	für grosse Systeme ungeeignet; können nicht konstruiert werden, sondern wachsen evolutionär
Modellierung des natürlichen Lebenslaufes	lauffähige Systemteile entstehen erst sehr spät
Förderung eines planvollen Vorgehens	System muss in einem Schritt komplett in Betrieb genommen werden

Tabelle 2: Vor- und Nachteile des ergebnisorientierten Phasenmodells

2.3 Wachstums-Modell

Wachstumsmodelle stellen den Gedanken der Software-Evolution in den Mittelpunkt. Ein System wird nicht konstruiert, es wächst in einer Reihe aufeindanderfolgender Schritte.
Es ist ein Modell für den Software-Entwicklungsprozess, das die Entwicklung in eine Folge von Iterationen unterteilt. In jeder Iteration wird ein vollständiges Teilergebnis mit betriebsfähiger Software erarbeitet und ausgeliefert.
Jede dieser Iterationen wird als weitgehend autonomes Teilergebnis organisiert. Bei der Definition des Umfanges und der Reihenfolge der Lieferungen wird immer ein schrittweiser Ausbau des Systems bis zum gewünschten Endzustand geplant. Dabei können verschiedene Merkmale schrittweise ausgebaut werden, z. B. Funktionalität, Bedienungskomfort, Leistung oder Wiederverwendbarkeit.

Vorteile	Nachteile
Modelliert das natürliche Verhalten der meisten grossen Systeme	Gefahr, dass viele nicht ganz zusammenpassende Teilsysteme entstehen
Lauffähige Teile entstehen sehr schnell	Konzepte und Strukturen können durch Ergänzungen und Änderungen bis zur Unkenntlichkeit entstellt werden
Sanfte Einführung des Systems möglich	

Tabelle 3: Vor- und Nachteile des Wachstummodells

2.4 Spiral-Modell

Das Spiralmodell ist eine Weiterentwicklung des Wasserfallmodells. Es ist vor allem für die Entwicklung grosser, risikoreicher Systeme gedacht.

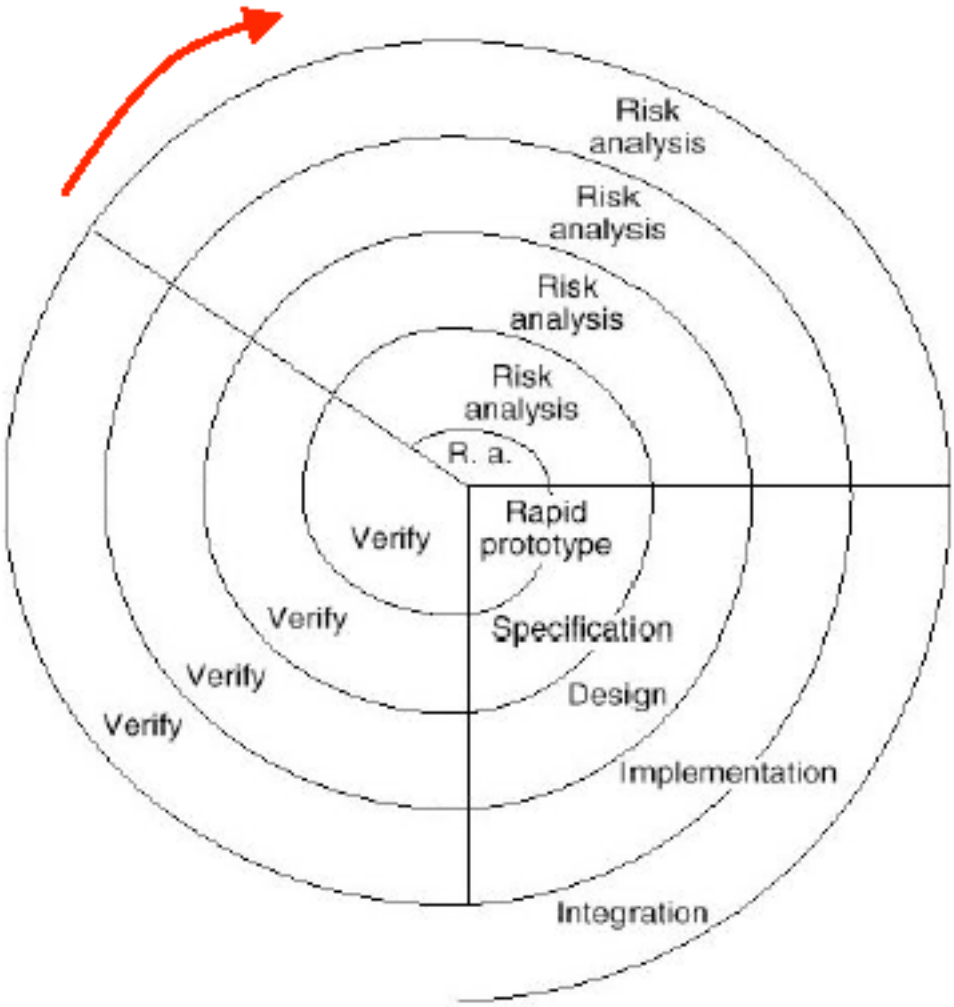

Bild 2: Spiralmodell

Das Spiral-Modell arbeitet mit Zyklen, von denen jeder im Wesentlichen einer Phase des Wasserfall-Modells entspricht. In jedem Zyklus werden die folgenden Phasen durchlaufen:

- Klärung der Ziele, Alternativen und Randbedingungen
- Gegebenfalls Einsatz von Prototyping zur Klärung der Ziele, Vorstellungen, Machbarkeit
- Ablauf geeigneter Schritte, wie im Wasserfall-Modell
- Review der abgelaufenen Phase und Planung der folgenden

Ein Zyklus im Spiral-Modell entspricht einer Phase im Wasserfall-Modell. Diese wird aber durch einleitende Risikoanalyse und abschließenden Review dynamischer als im Wasserfall-Modell an den Projektstand angepasst. Nicht berücksichtigt werden Iterationen und parallel ablaufende Aktivitäten.

2.5 V-Modell

Das V-Modell wurde ursprünglich im Auftrag des Bundesministeriums für Verteidigung entwickelt.

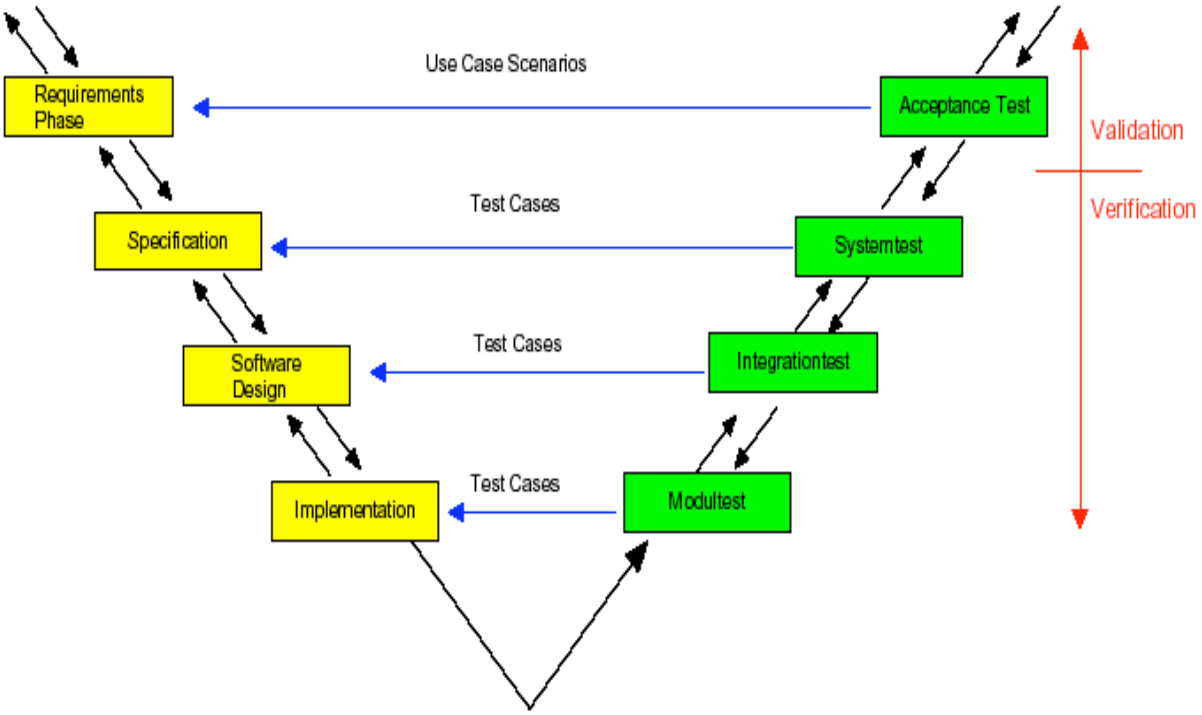

Bild 3: V-Modell

Es beschreibt Aktivitäten und Produkte und regelt die Gesamtheit der Aktivitäten und Produkte sowie die Produktzustände und die Abhängigkeiten zwischen Aktivitäten und Produkten. Das V-Modell beschreibt den Software-Entwicklungsprozess nur aus funktionaler Sicht, d.h. es werden keine Aussagen über die konkrete organisatorische Umsetzung getroffen.

2.6 Eigenschaften

Die klassischen Methoden versuchen, einen grossen Teil des Software-Prozesses für ein grosse Zeitspann genau zu planen. Das funktioniert nur solange, bis sich etwas ändert. Dann müssen alle Pläne wieder angepasst werden.

Das Ziel der klassischen Methoden ist es, einen Prozess zu definieren, der funktioniert, egal wer ihn ausführt.

Es lassen sich folgende Nachteile feststellen:

- grosse Zeitspanne, hoher Detaillierungsgrad
- Prozess im Zentrum

3 Agile Methoden

Die agilen Methoden stellen allesamt den Mensch ins Zentrum der Überlegungen. Es ist ihnen gemeinsam, dass sie die Zeit und die Ressourcen konstant halten und die Funktionalität variieren.

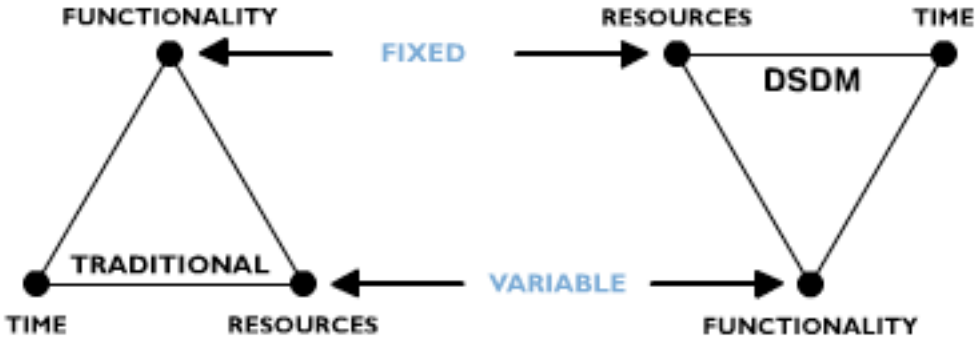

Bild 4: Vergleich klassische versus agile Methoden

Was charakterisiert eine agile Entwicklungsmethode? Die Entwicklung der Software erfolgt

- inkrementell (kleine Releases in kurzen Abständen)
- kooperativ (Kunde und Entwickler arbeiten die ganze Zeit mit enger Kommunikation zusammen)
- geradlinig (die Methode selbst ist einfach zu lernen und zu ändern)
- adaptiv (fähig, auch Änderungen im letzten Moment zu vollziehen)

Cockburn [3] definiert das Herz der agilen Software-Entwicklungsmethoden als Nutzung von leichten, aber genügenden Projektverhaltensregeln und die Anwendung von menschen- und kommunikationsorientierten Regeln. Der agile Prozess ist sowohl leicht als auch genügend. Leichtigkeit ist ein Mittel, um beweglich zu bleiben. Genügend ist eine Sache des-im-Spiel-bleibens. Er listet folgende Punkte auf für ein erfolgreiches Projekt:

- zwei bis acht Leute in einem Raum
- Anwender vor Ort
- kurze inkrementelle Phasen
- voll automatisierte Regressionstests
- erfahrene Entwickler

In den folgenden Unterkapiteln werden die einzelnen Methoden aufgeführt und kurz erläutert.

3.1 Exteme Programming (XP)

Die Begründer von Extreme Programming heissen Kent Beck, Ward Cunningham und Ron Jeffries.

Was ist Extreme Programming [4]: „Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.“

XP ist entstanden als Reaktion auf die Probleme, die Projekte mit langen Entwicklungszyklen auf Grund von traditionellen Entwicklungsmethoden verursachen. Am Anfang war es die Idee, wie man die Arbeit erledigen konnte, mit Hilfe von Praktiken, die man als effektiv ansah und sich in den vorangehenden Jahren bewährt hatten.

3.1.1 Der Prozess

Der Lebenszyklus beseht aus fünf Phasen:
- Erforschung
- Planung
- Iterationen bis zum Release
- Produktion
- Unterhalt
- Tod

3.1.2 Praktiken

XP ist eine Kollektion von Ideen und Praktiken, die aus bereits vorhandenen Methoden zusammengetragen wurde:
- Die Entscheidungsstruktur, in der der Kunde über Geschäftsdinge und die Entwickler über technische Dinge entscheiden, stammt von Alexander [5]
- Die rasche Art der Entwicklung hat seinen Ursprung in Scrum (siehe 3.2) und der Patternsprache [6]
- Die Art, Projekte auf Grund von Storycards zu planen, hat seine Wurzeln in den Use Cases [7]
- Die evolutionäre Lieferung wurde von Gilb übernommen [8]
- Auch das Spiralmodell hatte seinen Einfluss

3.2 Scrum

Die Begründer von Scrum heissen Ken Schwaber, Jeff Sutherland und Mike Beedle.
Scrum tauchte 1986 zum ersten Mal auf, als ein adaptiver, schneller und selbst organisierender Produktprozess aus Japan präsentiert wurde [9].
Die Bedeutung des Wortes „Scrum“ hat seinen Ursprung im Rugby; es ist die Strategie, einen Ball wieder ins Spiel zu bringen.

Scrum ist ein agiler, leichtgewichtiger Prozess, der benutzt werden kann, um Software- und Produktentwicklung zu managen. Indem Scrum existierende Praktiken beinhaltet, Extreme Programming eingeschlossen, kombiniert es die Vorteile der agilen Entwicklung mit den Vorteilen einer simplen Implementierung. Scrum konzentriert sich darauf, wie die Teammitglieder arbeiten sollen, um ein System flexibel in einer sich konstant verändernden Umgebung zu produzieren.

Scrum hat drei Phasen:
- Vorspiel
- Entwicklung
- Nachspiel

3.3 Crystal

Der Begründer von Crystal ist Alistair Cockburn.
Crystal ist eine Familie von menschengetriebenen, adaptiven, ultraleichten Software-Entwicklungsmethoden.
- Menschengetrieben: der Fokus bei der Erreichung eines Projekterfolges liegt auf Arbeitssteigerung der involvierten Menschen.
- Ultraleicht: Egal welche Grösse oder Prioritäten das Projekt hat, Crystal reduziert den Aufwand an Papierarbeit und Bürokratie auf ein minimales Level, das für dieses spezifische Projekt genügt

Es gibt Gemeinsamkeiten, die alle Crystal Methoden benutzen:
- Inkrementelle Entwicklungszyklen mit einer Dauer von einem bis drei Monaten
- Kommunikation und Kooperation

Drei Crystal Methoden wurden konstruiert:
- Crystal Clear
- Crystal Orange
- Crystal Orange Web

3.4 Feature Driven Development (FDD)

Die Begründer von FDD sind John deLuca und Peter Coad.
Indem man Informationen aus der Modellierungsaktivitäten und aus allen anderen Anforderungsaktivität nimmt, generieren die Entwickler eine Eigenschaftsliste. Als nächstes wird ein Grobplan erstellt und Verantwortlichkeiten werden zugewiesen. Danach werden kleine dynamische Teams gebildet, welche die Eigenschaften implementieren in Zeiteinheiten von maximal zwei Wochen.

3.5 Rational Unified Process (RUP)

Rational Unified Process (RUP) wurde von Philippe Kruchten, Ivan Jacobsen und anderen bei der Firma Rational entwickelt, um UML zu vervollständigen.
RUP ist ein inkrementeller Ansatz für objektorientierte Systeme; es verwendet Use Cases, um Anforderungen zu modellieren und das Fundament für das System zu legen.

Die Lebenszeit von RUP ist in vier Phasen aufgeteilt:
- Anfang
- Erarbeitung
- Konstruktion
- Übergang

3.6 Dynamic Sytems Development Method (DSDM)

DSDM ist ein Framework, das von einem Konsortium von siebzehn Firmen 1994 erstellt wurde (u.a. Oracle, British Airways, American Express). Das Framework gehört (Copyright und geistiges Eigentum) dem DSDM-Konsortium, eine NPO.
Wenn man DSDM anwenden will, sprich eine Lizenz dafür haben möchte, muss man dem DSDM-Konsortium beitreten. Das Konsortium steckt alle Einnahmen der Lizenzen in die Weiterentwicklung des Frameworks.
Die fundamentale Idee hinter DSDM ist die, dass die Zeit und Ressourcen fixiert werden und die Funktionalität danach gerichtet wird.

3.6.1 Lebenszyklus

Der Produktprozess hat sieben Phasen, welche während des Projektes wiederholt werden; DSDM ist eine iterative und inkrementelle Methode.

Phase	Name	Beschreibung
1	Vor-Projekt-Phase	Stellt sicher, dass nur die richtigen Projekte gestartet werden und dass sie korrekt aufgesetzt werden
2	Machbarkeitsstudie	Beurteilung, ob DSDM der richtige Ansatz für das Projekt ist und Definition des Problems, Beurteilung der Kosten und der technischen Machbarkeit der Lieferung des Systems
3	Geschäftsstudie	Hauptaugenmerk gilt den Geschäftsprozessen, die betroffen sind und deren Informationsbedürfnis.

Phase	Name	Beschreibung
4	Funktional-Modell-Iteration	Verfeinerung der geschäftsbasierten Aspekte des Systems
5	Entwurf- und Bau-Iteration	Errichtung des Systems
6	Implementation	Übergang von der Entwicklungsumgebung zur operationellen Umgebung
7	Nach-Projekt	Effektivitätserhaltung des Systems

Tabelle 4: Lebenszyklus des DSDM

3.6.2 Zu Grunde liegende Prinzipien

- Klare Definition der Rollen und Verantwortlichkeiten im Projekt
- Aktiver Einbezug der Anwender ist zwingend bei vier definierten Geschäftsrollen
- Das Team muss ermutigt werden, Entscheidungen zu treffen
- Der Fokus liegt auf häufigen Lieferungen des Produktes, welche definiert sind nach Zweck und Qualität und wer verantwortlich ist für die Lieferungen
- Anpassung an Geschäftszwecke ist das essentielle Kriterium für die Akzeptanz der Lieferung
- Iterative und inkrementelle Entwicklung ist notwendig, damit Entwicklung und eine angemessene Geschäftslösung konvergieren
- Alle Änderungen während der Entwicklung sind umkehrbar
- Anforderungen werden auf einem hohen Level gewonnen (Requirements are baselined at a high level)
- Testen ist in den Lebenszyklus integriert
- Zusammenarbeit und Kooperation zwischen allen Beteiligten ist essentiell

3.7 Adaptive Software Development (ASD)

Der Begründer von ASD ist Jim Highsmith.
ASD behandelt hauptsächlich Probleme, die bei der Entwicklung grosser und komplexer Systeme entstehen. Die Methode beruht auf inkrementeller und iterativer Entwicklung, mit Einbezug von konstantem Prototyping. ASD „balanciert auf der Kante des Chaos“ – das Ziel ist ein Framework anzubieten, das Projekte gerade noch vor dem Chaos schützt, aber das Entstehen und die Kreativität nicht unterdrücken.
Ein ASD-Projekt wird in drei Phasen-Zyklen ausgeführt:

- **Spekulieren** wird an Stelle von Planung verwendet, weil bei einem Plan Unsicherheit als Schwäche angesehen wird und von dem aus Abweichungen einen Misserfolg darstellen.
- **Zusammenarbeiten** hebt die Bedeutung von Teamwork hervor, um sich ständig verändernde Systeme zu entwickeln.
- **Lernen** hebt die Notwendigkeit hervor, Fehler zu erkennen und darauf zu reagieren und die Tatsache, dass Anforderungen während der Entwicklung ändern können.

3.8 XBreed

(Mix aus Scrum, XP und Alexandrinischen Ideen) seit 2000

Die Kombination Scrum/XP war sehr natürlich: Scrum liefert ein solides Management-Framework und XP bietet einen grundlegenden und kompletten Satz von Entwicklungstechnologien. Das Resultat ist eine schlanke aber sehr effektive Art, Software zu entwickeln.

3.9 Agile Modelling (AM)

Der Begründer von AM ist Scott Ambler. Es entstand 2002.
AM ist eine praxisbasierte Methode, um Softwaresysteme effektiv zu modellieren und dokumentieren.
Die vier zentralen Werte sind die gleichen wie bei XP: Kommunikation, Einfachheit, Feedback, Mut. Zusätzlich wird Menschlichkeit gross geschrieben, indem die Tatsache hervorgehoben wird, dass verschiedene Leute unterschiedliche Erfahrungen haben. Diese Erfahrungen sollen ausgenutzt werden, indem Kooperation gefördert wird und die besten Leute für eine Aufgabe gesucht werden.

3.10 Pragmatic Programming (PP)

PP ist keine eigentliche Technik; es gibt keine Methode die „pragmatic programming“ heisst. PP kommt von einem Buch [10], das einen interessanten Satz von Programmierpraktiken enthält. Beide Autoren dieses Buches sind in die agile Entwicklung involviert (Agile Manifesto). Ausserdem sind die Methoden, die im Buch diskutiert werden, Bestandteile anderer agiler Methoden.
PP hat weder einen Prozess, Phasen, unterschiedliche Rollen noch ein Produkt. Die Philosophie kann auf alle Phasen der Software-Entwicklung angewandt werden. Diese Philosophie kann in sechs Punkten dargelegt werden:

- Übernimm Verantwortung für das, was du tust. Erfinde Lösungen statt Entschuldigungen.
- Gib dich nicht zufrieden mit schlechtem Design oder Code. Behebe Inkonsistenzen sobald du sie siehst oder plane, sie schnellstmöglich zu eliminieren.
- Übernimm eine aktive Rolle, Veränderungen einzuführen, wenn du siehst, dass es notwendig ist.
- Kreiere Software, die den Kunden zufrieden stellt, aber wisse, wann du aufhören musst.
- Vergrössere konstant dein Wissen.
- Verbessere deine Fähigkeiten im Bereich Kommunikation.

Es wird erwähnt, wie man Software entwirft und implementiert, dass sie Änderungen standhält. Lösungen, wie man Software während Änderungen konsistent hält, werden ebenfalls genannt. Refactoring ist eine Lösung.
Beim Testen soll der Testcode parallel zum eigentlichen Code erzeugt werden. Die Tests sollen automatisch erfolgen. So werden alle gefundenen Fehler eliminiert und Regressionstests werden regelmässig durchgeführt; falls dies nicht geschieht, braucht es viel Zeit und Aufwand, immer wieder die gleichen Fehler zu finden.
Der Automatisierung wird ein grosses Gewicht beigemessen. Ein Kernsatz lautet: „Don’t use manual procedures“. Als Beispiele werden Dokumentationsentwürfe aus dem Quellcode, Generierung von Code aus Datenbankdefinitionen und automatische nächtliche Builds genannt. Es werden Beispiele für Software gegeben, die die Automatisierung unterstützen.

3.11 Lean Programming / Development

Die Begründer von Lean Programming sind Tom und Mary Poppendieck
In den letzten 25 Jahren hat das schlanke Denken einen grossen Einfluss auf die globale Wirtschaft gehabt (z. B. Lean Production) und viel Firmen verändert, wie sie arbeiten und wie sie über ihre Arbeit und ihre Arbeiter denken. Das Schlanke funktioniert, weil es Mehrwert produziert, indem es Abfall eliminiert.
Lean Programming wendet diese Ansätze auf die Software-Entwicklung an.
Schlank betrifft nicht das Team; es betrifft die Entscheidungen, was ein Team tut und wann es dies tut. Die Anwendung von schlanken Denkansätzen ist in allen Bereichen unterschiedlich und ist auch bei verschiedenen Software-Entwicklungszusammenhängen anders. Praktiken, die beim Lean Manufacturing oder bei der Lean Construction

angemessen sind, müssen nicht notwendigerweise für die Software-Entwicklung angemessen sein.
Die Grundgedanken (auf die Software-Entwicklung angewandt) sind:

- Eliminiere Abfall
- Verstärke das Lernen
- Entscheide so spät als möglich
- Liefere so schnell wie möglich
- Bekräftige das Team
- Baue Integrität ein
- Sieh das Ganze

Lean Development ist keine Entwicklungsmethode, sondern eine Art zu denken, welchen Ansatz ein Team wählen soll. Es gibt Unterstützungswerkzeuge, die Teams unterstützen, welche Methode angemessen ist für das betreffende Problem. Die agilen Methoden (XP, Scrum, DSDM, ASD inbegriffen) sind konsistent mit der Art, wie Lean Deveopment das schlanke Denken auf die Software-Entwicklung anwendet.

3.12 Test Driven Development (TDD)

Der Begründer von TDD ist Kent Beck.
TDD ist die Kunst, automatisch Tests für den Produktionscode zu genieren und diesen Prozess zu benutzen, um Design und Programmierung zu steuern.
Für jeden noch so kleinen Teil an Funktionalität wird zuerst ein Testfall geschrieben, der spezifiziert und validiert, was der Code machen soll. Dann wird gerade so viel Code geschrieben, dass der Testfall erfolgreich durchlaufen werden kann. Dann vereinfacht man den Test und den Code.
Die Schritte sind:

- Füge einen Test hinzu, generiere gerade genug Code, dass der Test fehlschlägt.
- Lass den Test laufen (im besten Fall die ganze Testsuite, im Minimum den neuen Test).
- Ändere den funktionellen Code, damit er den neuen Test erfüllt.
- Führe alle Tests aus (die ganze Suite). Wenn das fehlschlägt, ändere den funktionellen Code und teste nochmals.
- Halte nach Duplikaten Ausschau und eliminiere sie.

Diese Sequenz wird während des ganzen Programmierprozesses angewandt. Jeder Zyklus hat eine Dauer von einigen Minuten. Wenn man nicht in der Lage ist, einen fehlgeschlagenen Test in wenigen Minuten zur Erfüllung zu bringen, dann soll man den Testfall und den funktionalen Code eliminieren und einen einfacheren Test erfinden.

3.13 Open Source Software Development

Open Source Software-Entwicklung hat Gemeinsamkeiten mit den agilen Entwicklungsmethoden. Der Open Software-Entwicklungsprozess startet mit frühen und häufigen Releases; es fehlt ebenfalls an vielen traditionellen Mechanismen, die die Entwicklung mit Plänen, Systemdesigns und definierten Prozessen koordinieren.
Ein Open Source-Projekt hat folgende Phasen:

- Problem erkennen
- Freiwillige finden
- Lösung identifizieren
- Code entwickeln und testen
- Codeänderungen reviewen
- Code gutheissen und dokumentieren
- Releases freigeben

4 Organisationen und Konferenzen der agilen Methoden

In diesem Kapitel werden die Organisationen und Konferenzen, die sich mit den agilen Methoden befassen, eingeführt und erklärt.

4.1 Agile Alliance [11]

Die Agile Alliance ist eine lose Organisation, die sich mit den agilen Methoden befasst. Sie kam im Februar 2001 zusammen und enthält die Vertreter und Gründer verschiedener agiler Methoden (XP, ASD, Scrum, DSDM, ASD, Crystal, FDD, PP und andere). Aus dieser Zusammenkunft resultierte das Agile Manifesto (siehe 4.1.2).

4.1.1 Mitglieder

Die Mitglieder der Agile Alliance sind:

Name	Org / agile Methode	Name	Org / agile Methode
Kent Beck	XP	Andrew Hunt	PP
Mike Beedle	Scrum, XBreed	Ron Jeffries	XP
Arie van Bennekum	DSDM	Jon Kern	
Alistair Cockburn	Crystal	Brian Marick	Agile Testing
Ward Cunningham	XP	Robert C. Martin	
Martin Fowler		Ken Schwaber	Scrum
James Grenning		Jeff Sutherland	Scrum
Jim Highsmith	ASD	Dave Thomas	PP

Tabelle 5: Mitglieder der Agile Alliance

4.1.2 Manifesto

Das Manifesto der Agile Alliance lautet wie folgt:

We are uncovering better ways of developing software by doing it and helping others to do it.
Through this work we have come to value:

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

Was bedeutet dieses Manifest?

- Individuals and interactions over processes and tools
  Die agile Bewegung unterstreicht die Beziehung und Gemeinschaft von Software-Entwicklern und den Menschen im Zentrum der Entwicklung. Das wirkt sich in engen Teambeziehungen, angenehmen und teamgerechten Arbeitsbedingungen und allem, was den Teamgeist fördert, aus.

- Working software over comprehensive documentation
  Das Ziel ist, gestestete und funktionsfähige Software auszuliefern. Neue Releases werden in kurzen Abständen gefertigt (zwischen stündlich und monatlich). Die Entwickler werden angehalten, den Code einfach, zielgerichtet, nach den neuesten technologischen Erkenntnissen zu erstellen und so die Dokumentation auf ein erträgliches Minimum zu reduzieren.

- Customer collaboration over contract negotiation
  Der Beziehung und Zusammenarbeit zwischen Entwicklern und Kunden wird Vorrang gegeben gegenüber strikten Verträgen. Dennoch steigt die Wichtigkeit von gut ausformulierten Verträgen mit der Projektgrösse. Der Verhandlungsprozess sollte als Mittel gesehen werden, eine lebensfähige Beziehung zu erreichen und zu erhalten.

- Responding to change over following a plan
  Die Entwicklungsgruppe (Entwickler und Kunden) sollten gut informiert über den Projektstand sein, autorisiert und kompetent sein, mögliche Änderungen während des Entwicklungsprozesses zu erwägen. Das bedeutet, dass die Teilnehmer auf Änderungen vorbereitet sind und die Verträge solche Änderungen erlauben.

4.2 XP Agile Universe [12]

Diese Konferenz fand dieses Jahr im August (10. bis 13.) in New Orleans, Louisiana, statt. Sie bietet neben Vorträgen zu vielen agilen Methoden auch Workshops und Tutorials an. Sie wird mittels der Open-Space-Track-Technologie (OST) organisiert.

4.2.1 OST

OST ist eine agile Methode, um Meetings zu organisieren. Sie wird angewandt, um Treffen von bis zu 1000 Leuten zu organisieren. Themen werden ausgewählt, Zeit und Ort festgelegt.

Diese Methode hat vier Prinzipien und ein Gesetz:
- Die Prinzipien:
  - Egal wer kommt, es sind die richtigen Leute.
  - Was auch immer passiert, es war das einzige, was passieren konnte.
  - Wann auch immer es beginnt, es ist die richtige Zeit.
  - Wenn es vorbei ist, ist es vorbei.
- Das Gesetz:
  - Wenn jemand während des Treffens findet, dass er weder lernen noch etwas beitragen kann, muss diese Person an einen produktiveren Platz gehen.

4.3 Agile Development Conference [13]

Diese viertägige Konferenz beschäftigt sich Techniken und Technologien, Einstellungen und Politik, Forschung und Erfahrung, Führung und Entwicklung der agilen Software-Entwicklung. Sie beschränkt sich nicht auf eine einzelne Technologie, sie ist ein Forum für den Informationsaustausch von allen agilen Entwicklungstechnologien.
Die Konferenz bietet den Teilnehmern Zugang zu den neuesten Erkenntnissen der agilen Methoden und dient als Brücke zwischen einzelnen Communities, um Ideen und Gedanken auszutauschen. Sie bringt Akademiker, Entwickler aus Forschungslabors und der Praxis und Manager zusammen.
Es werden Forschungspapiere und Erfahrungsberichte präsentiert.
Forschungspapiere befassen sich mit:
- Signifikante Beiträge zum Thema agile Software-Entwicklung
- Vorantreibung des „State-Of-The-Art“
- Beeinflussung des Frameworks in der Praxis
- Begründete Kritik an agilen Entwicklungsmethoden

Erfahrungsberichte enthalten Informationen aus erster Hand und Rückblick. Sie enthalten Erkenntnisse, die durch reale Projekte gesammelt wurden: was funktioniert, was nicht, und warum.

Die nächste Agile Development Conference, die vom 23. bis 26. Juni 2004 in Salt Lake City, Utah, stattfindet, wird sich mit folgenden Themen befassen:

- Forschung an neuen oder existierenden agilen Entwicklungsmethoden und –ansätzen wie ASD, Crystal, DSDM, FDD, Scrum, XP
- Fallstudien und empirische Studien, die agile Entwicklung oder eine bestimmte Technik, Werkzeug oder Ansatz haben: Was funktionierte, was nicht?
- Skaliert die agile Entwicklung im Grossen (inkl. Multi-Personen-/Multi-Jahre-/Multi-Komponenten-Projekte) und sicherheits-, lebens- und aufgaben-kritische Systeme?
- Kritische Vergleiche und Evaluationen von alternativen agilen Entwicklungsmethoden
- Geschäftsanalysen: ist agile Entwicklung kosteneffizient und gerechtfertigt?
- Führung von Projekten, die agil entwickelt werden, den Teams und deren Mitglieder
- Beziehung zwischen agiler Entwicklung und User-centered design (UCD)
- Einführung von agilen Methoden in bestehende IT-Organisationen; Reaktionen von Entwicklern und Managern auf die agilen Methoden
- Soziologie der agilen Entwicklung: gibt es personenbezogene Probleme bei der Anwendung von agilen Entwicklungsmethoden und wie können diese gelöst werden? Soziologie von Software-Entwicklung und Konsequenzen für die agile Entwicklung
- Beziehungen zwischen CSCW und agiler Entwicklung
- Werkzeuge für die agile Entwicklung

5 Fazit

5.1 Was sagen die Verfechter der klassischen Methoden zu den agilen Methoden?

Die Verfechter der klassischen Methoden suchen einen Beweis, dass die agilen Methoden funktionieren (siehe 5.2). Dies ist eine verständliche Frage, denn viele Entwickler wollen Sicherheiten, dass diese neuen Methoden wirklich funktionieren.

5.2 Funktionieren agile Methoden?

In diesem Unterkapitel wird erläutert, unter welchen Umständen agile Methoden funktionieren. Nur weil die Methode bei jemand anderem funktioniert, muss sie noch lange nicht allgemeingültig sein.
In News-Gruppen ist die Frage aufgetaucht, ob es einen Beweis gibt, dass agile Methoden funktionieren. [14]

5.2.1 Die agilen Methoden sind neu

Da die agilen Methoden relativ neu sind (Ursprünge Mitte der Neunzigerjahre), ist noch zu wenig Zeit verstrichen um zu beweisen, dass die agilen Methoden in einer Vielzahl von Situationen funktionieren.
Es gibt zwar Belege aus Anekdoten, dass die agilen Methoden funktionieren und immer mehr Anekdoten kommen dazu, aber ein statistischer Beweis, der auf einer detaillierten Studie basiert, existiert zur Zeit noch nicht und das wird sich wahrscheinlich erst in ein paar Jahren ändern.

5.2.2 Unrealistische Erwartungen

Es gibt fünf Profilarten von Technologie-Einführern: [15]
- Innovatoren: aggressive Verfolgung neuer Konzepte
- Früh-Einführer: verfolgen neuen Konzepte am Anfang von deren Lebenszyklus
- Frühe Mehrheit: warten ab, bevor sie ein neues Konzept einführen
- Späte Mehrheit: sind besorgt über ihre Fähigkeit, ein neues Konzept zu beherrschen, wenn sie es einführen sollen
- Nachzügler: wollen ganz einfach nichts mit neuen Ansätzen tun

Innovatoren und Früh-Einführer sind zufrieden, wenn sie eine Webseite oder ein Buch lesen, das die agilen Techniken beschreibt; sie überdenken das Konzept und passen es an ihre Umgebung an. In diesem Stadium sind die meisten agilen Techniken zur Zeit.
Die frühe Mehrheit wird darauf warten, dass genügend Belege aus Anekdoten existieren bevor sie auf den Zug der agilen Methoden aufspringen. Einige haben diesen Schritt schon vollzogen.
Die Frage nach dem Beweis wird von der späten Mehrheit oder sogar den Nachzüglern gestellt.

5.2.3 Wir sind ruiniert (We’re spoiled)

Ältere Arbeiten im Bereich der Software-Metriken haben möglicherweise die Latte für Beweise zu hoch gelegt [14].
Gute und schlechte traditionelle Methoden wurden zur Genüge erforscht, aber neuere Techniken wie Refactoring und Zusammenarbeit (co-location) mit dem Kunden wurden noch nicht erforscht.
Erschwerend kommt dazu, dass die meisten Erfinder der agilen Methoden Praktiker sind, die an Softwareprojekten arbeiten und deswegen sehr wenig Zeit haben, theoretische Studien über ihre Techniken zu machen. Es kann, um es mit anderen Worten zu sagen, noch eine

ganze Weile dauern, bis es Studien über agile Methoden gibt, die auf der gleichen Ebene wie die von Jones.

5.2.4 Es ist nicht klar, was eigentlich bewiesen werden soll angesichts der agilen Software-Entwicklung

Agile Ansätze beruhen auf beweglichen Umgebungen, in denen Plantreue (Pläne Monate im voraus erstellt) ein schlechtes Mass für Erfolg ist [17]. Wenn Agiltiät wichtig ist, muss diese Agilität gemessen werden. Traditionelle Erfolgsmessungen heben die Plantreue hervor. Agilität hebt die Reaktionsfähigkeit auf Änderungen hervor. Es entsteht ein Konflikt, weil Manager sagen, dass sie Flexibilität wollen aber den Erfolg an der Plantreue messen.

5.2.5 Vielleicht ist der Grund der Frage nach einem Beweis nicht angemessen

Sind diejenigen, die nach einem Beweis fragen, wirklich daran interessiert, einen effektiven Prozess zu finden oder suchen sie nur nach einem Grund, einen Ansatz in Verruf zu bringen, mit dem sie sich nicht anfreunden können? Sind sie realistisch genug um zu sehen, dass kein Softwareprozess perfekt ist, dass es kein Patentrezept (silver bullet) gibt?

5.2.6 Es gibt immer mehr Belege von der Forschungsgemeinde

Einige Bücher, die in letzter Zeit erschienen sind, haben gezeigt, dass agile Techniken funktionieren [18], [19].

5.2.7 Der Beleg mittels Anekdoten sieht ziemlich gut aus

Es gibt zur Zeit signifikante Belege von Anekdoten, dass agile Software-Entwicklungs-techniken funktionieren; man muss nur sich die Zeit nehmen, es in Newsgruppen oder Mailing-Listen zu entdecken. Wem das nicht genug ist, der ist (noch) nicht reif fürs Agile.

5.2.8 Es gibt genügend Belege, dass iterative und inkrementelle Entwicklung funktioniert

Craig Larman [20] listet eine ganze Reihe von Abfassungen, die sich auf iterative und inkrementelle Entwicklung beziehen.
Es wird gezeigt, dass viele der Praktiken der agilen Softwareentwicklung wirklich funktionieren, Praktiken wie inkrementelle Lieferungen und iteratives Vorgehen, welches Änderungen einschliesst.
Es zeigt auch, dass serielles Vorgehen, grössere Projekte und grössere Abstände zwischen einzelnen Releases zu einer grösseren Häufigkeit führt, dass Projekte scheitern.

5.3 Was ist Hype, was hat sich durchgesetzt?

Methode	Status (8/02)	Beschreibung
AM	In Entstehung	Methode ist weniger als ein Jahr verfügbar. Keine Forschung vorhanden, keine Erfahrungswerte
FDD	In Entwicklung	Methode ist weit herum anerkannt, erste Erfahrungsberichte, aktive Nutzergemeinde
Crystal		
Scrum		
PP		
ASD		
XP	Aktiv	Methode in verschiedenen Orten angewandt, Erfahrungsberichte, aktive Forschung, aktive Nutzergemeinde
RUP		
OSS		
DSDM		

Tabelle 6: Status der agilen Methoden [21]