1 INKREMENTORIENTIERTE VORGEHENSMODELLE IM KONTEXT: BEDEUTUNG FÜR DAS RISIKO- UND PROJEKTMANAGEMENT IN DER SOFTWAREENTWICKLUNG 2

1 Inkrementorientierte Vorgehensmodelle im Kontext: Bedeutung für das Risiko- und Projektmanagement in der Softwareentwicklung

1.1 Einleitung

Die inkrementorientierten Vorgehensmodelle, deren Wurzeln in den siebziger Jahren[i] liegen, feiern im Gewand der „agilen Methoden“ seit einigen Jahren ihre Renaissance. Selbstverständlich wiederholt sich die Geschichte nie in gleicher Form und so haben die agilen Methoden neben der Inkrementorientierung interessante und beachtenswerte Ansätze bzw. Prinzipien[ii] zu bieten.

Den sequentiellen Phasenmodellen wie dem Wasserfallmodell wird wieder einmal der Tod vorausgesagt. Vor gut fünfzehn Jahren stellte B. Böhm mit seinem Spiralmodell[iii] ein risikogetriebenes Meta-Vorgehensmodell (es kann in seinen Iterationszyklen unterschiedliche Vorgehensmodelle enthalten) als Alternative zum Wasserfallmodell vor. Dabei benutzte er die folgende Einleitung:

„Stop the life-cycle – I want to get off!“
„Life-cycle Concept Considered Harmful.“
„The waterfall model is dead.“
„No, it isn’t, but it should be.“

Noch deutlicher wird F.P. Brooks jun., der geistige Vater des IBM/360-Betriebssystems, in seiner 1995 erfolgten Aktualisierung zu seinem legendären Buch ‚The Mythical Man-Month‘ aus dem Jahre 1975:
„...The Waterfall Model Is Wrong!”. Die Hauptbegründung liegt für Brooks darin, dass das Wasserfallmodell auf der unrealistischen Annahme aufbaut, dass Fehler ausschließlich in der Realisierung geschehen und im Test problemlos behoben werden können.[iv]

Dieser Artikel beleuchtet abseits der Softwareengineering-Moden die Essenz der inkrementalen Modelle und versucht eine Antwort darauf zu geben, warum das Wasserfallmodell immer noch nicht endgültig seinen Platz in der Geschichte des Software-Engineerings gefunden hat.

1.2 Eine Begriffsbestimmung: Vorgehensmodelle

Ein Vorgehensmodell ist für die Softwareentwicklung ein Leitgerüst mit der Aufgabe, Hilfestellung und Orientierung für die Planung und Durchführung eines Projektes zu geben. Wesentliche Punkte der Hilfestellung sind:

1. Vordefinierte Aktivitätssequenzen und zugehörige Ergebnistypenbeschreibungen
2. Benötigte Rollen, Verantwortlichkeiten und Fähigkeiten, die zur Erledigung der Aufgaben benötigt werden
3. Hinweise zur Meilensteinbildung für das Projektmanagement
4. Standards, Methodenbeschreibungen und ggf. Werkzeugempfehlungen zur fachgerechten Abwicklung der in Frage kommenden Aktivitäten
5. Management- und Entwicklungsprinzipien, Leitwerte.

Die in Punkt 5 adressierten Prinzipien und Leitwerten treffen eher auf die inkrementorientierten Vertreter der agilen Methoden zu. Vorgehensmodelle sind oft nur für bestimmte Projekttypen gültig und müssen, je allgemeingültiger sie sind, mit zunehmenden Aufwand auf das konkrete einzelne Projekt zugeschnitten werden.

1.2.1 Phasenorientierte Vorgehensmodelle

Phasenorientierte Vorgehensmodelle zeichnen sich dadurch aus, dass gleiche Aktivitäten in einer Phase zusammengefasst sind und möglichst in voller Breite durchzuführen sind. Die Phasen werden sequentiell abgewickelt und die Ergebnisse werden schriftlich dokumentiert. Man spricht in diesem Zusammenhang auch von einem dokument- bzw. spezifikationsgetriebenen Vorgehensmodell. Es wird angestrebt, jede Phase möglichst nur einmal zu durchlaufen bzw. Rückkopplungen auf die unmittelbar benachbarte Phase zu beschränken. Aber das wesentliche Merkmal, auf das im Laufe dieser Abhandlung mehrfach Rückgriff genommen wird, ist, dass die eigentliche Programmierung und der Test entsprechend der Philosophie der schrittweisen Verfeinerung erst zeitlich am Ende des Konstruktionsprozesses lokalisiert ist. Der prominenteste

[i] C. Larman, V. Basili, Iterative and Incremental Development: A Brief History, IEEE Computer, Vol 36, June 2003
[ii] Jens Coldewey, Management am Rande des Chaos: Die Führung agiler Projekte, Objektspektrum 5/2001
[iii] Barry W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer, Vol 21, May 1988
[iv] Frederick P. Brooks, JR, The Mythical Man-Month, Anniversary Edition, Addison Wesley, 1995, S. 264 ff

Vertreter der phasenorientierten Vorgehensmodelle ist das Wasserfallmodell[v], welches auf das Jahr 1970 zurückgeht (Winston Royce hatte es 1970 selbst nicht so genannt) und sich heute immer noch in adaptierten Formen einer großen Verbreitung erfreut (s. Abbildung 1).

„Hopefully, the iterative interaction between the various phases is confined to successive steps."
Dr. Winston Royce (5)

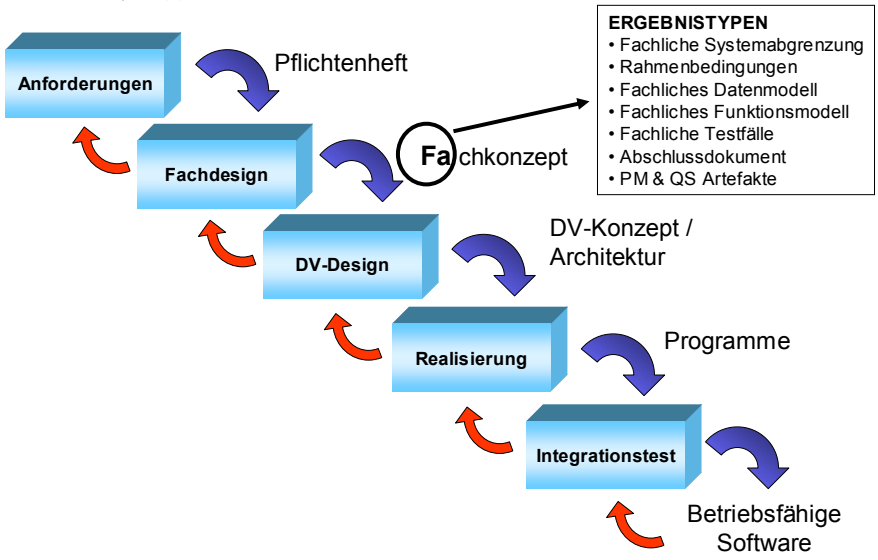

Abbildung 1: Das Wasserfallmodell

Winston Royce war sich der Schwierigkeit wohl bewusst, Änderungen auf die jeweils vorhergehenden Stufen zu begrenzen. Er empfahl sogar, das Wasserfallmodell zweimal zu durchlaufen und im ersten Durchlauf wichtige Hypothesen bzw. Tests (Royce benannte das Ergebnis des ersten Durchlaufs „Pilot") durchzuführen, um nicht am Schluss eines konventionellen Vorgehens entlang der empfohlenen Stufen auf unüberwindliche Schwierigkeiten zu stoßen.

1.2.2 Inkrementorientierte Vorgehensmodelle

Die inkrementale Entwicklung von Software eignet sich besonders, wenn die zukünftigen Anwender nicht genau spezifizierbare Vorstellungen über die Funktionalität des zu erstellenden Endproduktes (Finalziel) haben. Die folgende Aussage mag das illustrieren: Ich kann Ihnen nicht genau sagen, was ich benötige, aber wenn ich es sehe, dann weiß ich es. Man sollte sich allerdings davor hüten, dies generell dem Unvermögen der Anwender zuzuschreiben, denn gerade wenn kreative Akte gefordert sind, hat selbst der Spezialist selten eine konkrete Anforderungsspezifikation vor Augen bzw. in der Hinterhand.
Die inkrementorientierte Vorgehensweise fokussiert darauf, dem Anwender sehr früh im Entwicklungszyklus lauffähige Versionen für den Test bereitzustellen. Es muss sich hierbei nicht um eine Produktionsübergabe handeln. Der Vorteil ist, dass der Anwender seine Produktvorstellung (typische Inkrementdauern liegen je nach Projekttyp zwischen ein und sechs Monaten) anhand eines ersten lauffähigen Inkrementes überprüfen kann. Aufgrund seiner konkreten Erfahrung, die er beim Test des ersten Inkrementes gewinnt, wird in Absprache mit dem Entwicklungsteam die Funktionalität des nächsten Inkrementes geplant. Zusätzlich besteht nach jeder Inkrementfertigstellung die Möglichkeit, Änderungen des Umfeldes oder der hauseigenen Strategie in die Planung für das nächste Inkrement aufzunehmen. Ein weiterer unbestrittener Vorteil der Methode ist, dass vermutete technische Risiken zusammen mit den Anwenderfunktionen in den ersten Inkrementen adressiert werden können, um der Überraschung aus dem Wege zu gehen, dass man am Ende eines langen Konzeptionsweges vor unüberwindlichen technischen Problemen steht und die ganze Investition abgeschrieben werden muss. Das Problem des rein inkrementalen Vorgehen stellt sich, falls die Software-Architektur mit der Weiterentwicklung nicht Schritt halten kann und eine komplette Überarbeitung der bis zu diesem Zeitpunkt erstellten Inkremente notwendig wird. In diesem Fall haben die bisher erstellten Inkremente den Status eines Wegwerfprototypen, der allerdings die verifizierten und sehr wahrscheinlich auch die validierten, funktionalen Anforderungen auf dem konkretesten Level (aller möglichen) enthält.

[v] Dr. Winston Royce, Proceedings IEEE Wescon, August 1970, pages 1-9, Reprinted Proceedings ISCE9, Computer Society Press, 1987

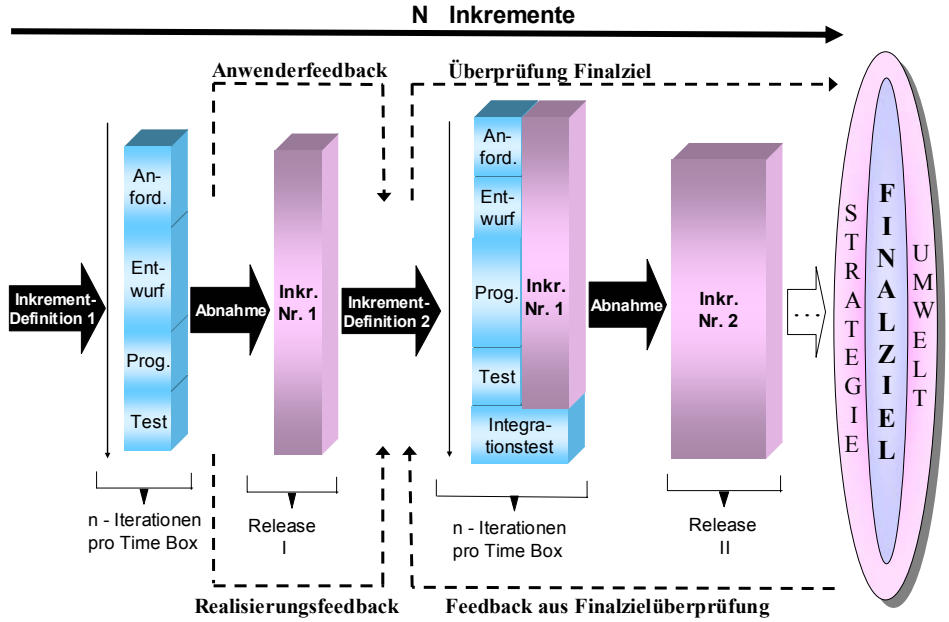

Abbildung 2: Das inkrementorientierte Modell

Das Modell in Abbildung 2 stellt die inkrementorientierte Entwicklung eines Softwareprodukts dar. Es zeigt, dass nach jeder Inkrementfertigstellung die Ziele für das jeweils nächste Inkrement gemeinsam von Anwendern und Entwicklern definiert werden und eine Überprüfung des Finalzieles stattfindet. Bemerkenswert ist, dass die Aktivitäten der Inkrementerstellung selbst (hier vertikal dargestellt) eine große Ähnlichkeit mit dem phasenorientierten Wasserfallmodell aufweisen. Der Unterschied ist, dass das Inkrementergebnis innerhalb kurzer Zeitspannen (als Bestandteil und nicht am Ende des Entwicklungszyklus) zur Verifikation und Validierung bereitgestellt wird und somit ein wesentlich geringerer Preis für Änderungen und Fehler gezahlt werden muss als beim klassischem Phasenmodell. Der geringere Preis speist sich aus zwei Quellen: Es müssen weniger bzw. in geringerem Umfang Entwicklungszwischenprodukte (Artefakte) geändert werden und der Qualitätssicherungsaufwand für Artefakte kann aufgrund der geringeren Fehlerbehebungskosten wesentlich niedriger gehalten werden.[vi]
Die Grafik (Abbildung 2) zeigt aber auch auf, dass mit jeder Iteration auch das schon getestete Inkrement erneut getestet werden muss. Das heißt, einer effizienten werkzeugbasierten Testunterstützung kommt eine erhöhte Bedeutung zu.

1.2.3 Prototypingorientierte Vorgehensmodelle

Prototypen sind eine preiswerte Methode, notwendige Erkenntnisse für eine geplante Entwicklung zu gewinnen. Mit ihrer Hilfe lassen sich Problemstellungen fachlicher und technischer Natur adressieren. Die verschiedenen Arten der Prototypenerstellung (z.B. evolutionäre, explorative, vertikale, horizontale Prototypen) sind mit Ausnahme des evolutionären Prototypen Bestandteile in anderen Vorgehensmodellen.
Werden bspw. technische Risiken in lauffähigen Codeblöcken adressiert, ohne dass zusätzliche Funktionalität ausgeliefert wird, oder einfach Benutzerschnittstellen ohne Hintergrundfunktionalität generiert, spricht man eher von Prototyping als von inkrementalem Vorgehen. Inkremente sind (echte) lauffähige Anwendungskomponenten, die dem Anwender zur Überprüfung der Funktionalität zur Verfügung gestellt werden. Dies kann, muss aber nicht mit einer produktiven Inbetriebnahme verbunden sein.

In Abgrenzung zum verwandten evolutionären Prototyping existiert beim inkrementorientierten Vorgehen zumindest eine mehr oder weniger klare Vorstellung über die Anforderungen (Finalziel) an das zu entwickelnde Endprodukt. Nach Balzert sind zu Beginn einer inkrementalen Entwicklung die Anforderungen an das zu entwickelnde Produkt möglichst vollständig zu erfassen.[vii] Der Verfasser dieses Beitrages schließt sich dieser restriktiven Definition nicht an, da sie das Einsatzgebiet der inkrementorientierten Vorgehensweise unnötig einschränkt.

[vi] Jens Coldewey, Beständig ist nur der Wandel – Agile Entwicklung und Änderbarkeit, Objektspektrum 6/2001
[vii] H. Balzert, Lehrbuch der Softwaretechnik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Spektrum Akademischer Verlag 1998, S. 122

1.3 Inkrementorientierung und agile Methoden

1.3.1 Inkrementorientierung ist das Merkmal der agilen Entwicklung

Im Februar 2001 trafen sich in Utah 17 Personen, unter ihnen Kent Beck (Extreme Programming), Alistair Cockburn (Crystal Methods) und Martin Fowler. Diese Gruppe legte das agile Manifest[viii] nieder, das aus vier Leitwerten und zwölf Prinzipien besteht. Die vier Leitwerte sind:

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

Zwei der Prinzipien (Nr. 1 und Nr. 3)[ix] lauten:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.*

Diese Prinzipien werden durch den zweiten Leitwert flankiert und implizieren unmittelbar ein inkrementorientiertes Vorgehen. Auch in der agilen Vorgehensweise des „Extreme Programming“ findet sich unter den fünf Grundprinzipien das Prinzip der „inkrementalen Veränderung“ an dritter Stelle.[x] In A. Cockborns Crystal-Methodenfamilie wird „Frequent Delivery“ als erste der drei Kern-Eigenschaften genannt.[xi] Ebenso findet sich die inkrementale Entwicklung in der agilen Entwicklungsmethode „Dynamic System Development Method“ (DSDM) in zwei (Nr. 3 und Nr. 5) von neun Prinzipien[xii] wieder:

- *The focus is on frequent delivery of products.*
- *Iterative and incremental development is necessary to converge on an accurate business solution.*

Es lässt sich also die Schlussfolgerung ziehen, dass die inkrementale Entwicklung ein Kernelement der agilen Methoden ist.

Die Leitwerte des agilen Manifestes relativieren die Stellung der dokumentierten Formalismen und Prozesse, die in den letzten Jahren mit ihren exponierten Vertretern wie ISO 9000 und dem Capability Maturity Model ihren Zenit vorerst erreicht haben. Die niedere Priorisierung von Prozessen und Dokumentation zugunsten von Eigenverantwortung und Kommunikation erlaubt auf der einen Seite schnelle Anpassungen an die Umwelt, funktioniert andererseits aber nur bis zu einer bestimmten Team- bzw. Projektgröße effizient. Boehm und Turner gehen von einer maximalen Teamgröße von vierzig Personen aus.[xiii] Das heißt, dass die Eignung der agilen Methoden von der Problemstellung abhängig ist und somit die „silver bullet“ des Softwareengineering immer noch nicht gefunden ist.

1.3.2 Architektursensibilität in der agilen Entwicklung

Das inhärente Problem der agilen (inkrementorientierten) Entwicklung ist die Korrumpierung der Architektur, d.h. Änderungen an der Software führen aufgrund redundanter Strukturen zunehmend zu ungewollten Nebenwirkungen. Die Software wird somit änderungsfeindlich und am Ende der Skala steht dann eine fehleranfällige, kaum wartbare Software, die von Entwicklern gemieden wird. Diesem Problem versucht Kent Beck, der prominente Vertreter des Extreme-Programming-Ansatzes, wie folgt zu begegnen: Die Architektur wird anhand der konkret vorliegenden Anforderungen (kein Designaufwand für nicht konkret vorliegende Anforderungen) für das jeweilige Inkrement geplant. Stellt man beim Codieren fest, dass die Architektur änderungsunfreundlich wird, weil z.B. duplizierter Code eingefügt werden muss, wird sie direkt überarbeitet. Dieses Prinzip wird Refactoring genannt. Kent Beck nennt eine Reihe von Voraussetzungen für eine erfolgreiche Anwendung des Refactoring- Prinzips, wie gemeinsame Verantwortlichkeiten im Programmiererteam, akzeptierte Programmierstandards, automatisierte Tests zur schnellen Aufdeckung ungewollter Beschädigungen, einfaches Design und fortlaufende Integration.[xiv]

[viii] http://www.agilemanifesto.org/, Download vom 2. April 2004
[ix] http://www.agilemanifesto.org/principles.html, Download vom 2. April 2004
[x] Kent Beck, Extreme Programming, Das Manifest, Addison Wesley 2000, S. 37
[xi] Crystal Clear, Alistair Cockburn, http://alistair.cockburn.us/crystal/articles/alistairsarticles.htm, Draft of: Crystal “Clear “ book, S. 6f, last save date: May 14, 2004, Download vom 16. Mai 2004
[xii] http://www.dsdm.com/tour/principles.asp, Download vom 2. April 2004
[xiii] Barry Boehm, Richard Turner, Balancing Agility and Discipline, Addison Wesley 2004, S. 4f, S.28, S. 176
[xiv] Kent Beck, Extreme Programming, Das Manifest, Addison Wesley 2000, S. 58, S. 65

WENN SIE WISSEN WOLLEN WIE ES WEITERGEHT SCHREIBEN SIE AN INFO@MERATHOR.DE. GEBEN SIE BITTE IHREN NAMEN UND IHRE FUNKTION/TÄTIGKEIT AN. SIE BEKOMMEN DANN DEN ARTIKEL KOSTENLOS ZU GESENDET