

BEWEGLICH BLEIBEN: MÖGLICHKEITEN UND GRENZEN ITERATIVEN VORGEHENS

Das iterative Vorgehen hat eine lange Geschichte, die in den 70er Jahren mit einigen Missverständnissen beginnt. Lange Zeit gab es Akzeptanzschwierigkeiten, bis iteratives Vorgehen schließlich populär wurde und nun manchmal als agiles Wundermittel wieder missverstanden wird. Welche Möglichkeiten bietet iteratives Vorgehen in der Praxis wirklich? Welche Begrenzungen existieren? Welche Erwartungen sind realistisch? Wie unterscheidet sich iteratives Vorgehen in kleinen, mittleren und in sehr großen Projekten? Was sind die jeweiligen Erfolgsfaktoren? Das sind die Themen dieses Beitrages.

In [Lar03] ist das historische Dilemma des iterativen Vorgehens gut zusammengefasst: Der „Erfinder“ des Wasserfallmodells Winston W. Royce verstand sein Modell ursprünglich nur als sehr einfaches, fast zu einfaches Modell, das ausdrücklich nur für einfachste Projekte gedacht war. Seine Idee bestand darin, das Wasserfallmodell innerhalb eines Projekts mindestens zweimal anzuwenden, also einen iterativ-inkrementellen Prozess durchzuführen.

Historisches Missverständnis

Ein anderer Autor, David Maibor, verantwortlich für den Standard DoD-STD-2167 (Standard des US-Kriegsministeriums), griff dies auf, war aber mit iterativer Vorgehensweise überhaupt nicht vertraut und konnte mit *Timeboxing*, Iterationen und ähnlichem nicht viel anfangen. Der DoD-Standard wurde schließlich von anderen Standards (V-Modell, CMM/SEI) übernommen und so begann die Ära des Wasserfallmodells.

Ein wasserfallartiges Vorgehen – also eine einzelne Sequenz von vollständiger Anforderungsanalyse, vollständigem Design, vollständiger Realisierung und schließlich vollständigem Test – funktionierte allen Standards zum Trotz in der Praxis von Softwareprojekten nie richtig. Es ist ganz natürlich, dass die beteiligten Menschen auch in späteren Phasen noch wichtige Erkenntnisse gewinnen, die die Ergebnisse der vorigen Phase(n) partiell in Frage stellen. Wenn aber beispielsweise eine vermeintlich komplette Anforderungserhebung Vertragsgrundlage für ein Soft-

wareprojekt wird, dann passen spätere Erkenntnisse ebenso wie veränderte Rahmenbedingungen nicht zum Vertrag. Entweder müssen diese Erkenntnisse unterdrückt werden, obwohl jeder um ihren Wert weiß, oder eine der beiden Vertragsparteien hat das Nachsehen, weil sie beispielsweise wichtige Features, die leider nur zu spät erkannt wurden, nicht realisiert oder nicht bezahlt bekommt.

Mitte der 90er Jahre verbreitete sich die Einsicht in die Mängel des Wasserfallmodells und das iterative Vorgehen wurde populärer. Das iterative Vorgehen ist bereits länger bekannt und wurde bereits in den 60er Jahren fundiert (vgl. z. B. [Zur68], [Bas75]).

Dem iterativen Vorgehen wurde – und wird teilweise immer noch – ein Misstrauen entgegen gebracht, das ebenfalls auf einem falschen, oftmals sehr naiven Verständnis von iterativem Vorgehen beruht. Hier zwei Extrembeispiele für ein falsches oder naives Verständnis von iterativem Vorgehen:

- *Das so genannte 4-Iterationen-Vorgehen:* Hierbei haben die einzelnen Iterationen jeweils einen ganz konkreten Zweck, der auch durch ihren jeweiligen Namen ausgedrückt wird: die Analyse-Iteration, die Design-Iteration, die Realisierungs-Iteration und die Test-Iteration. (Das Beispiel ist nicht erfunden!)
- *Das extrem planlose iterative Vorgehen:* Hier ist die Haltung: Wir gehen ja iterativ vor und beim iterativen Vorgehen müssen die Anforderungen

► der autor



Bernd Oestereich
(E-Mail: bernd.oestereich@oose.de) ist Geschäftsführer der oose.de GmbH und spezialisiert auf Unterstützung von Projekten im Bereich Projektmanagement, Entwicklungsmethodik und Softwarearchitektur.

nicht vollständig bekannt sein und können sich auch jederzeit ändern. Alles kein Problem. Deswegen müssen wir auch immer nur zwei Wochen im Voraus planen. Mit entsprechenden Refaktorisierungen und vielen kleinen Iterationen nähern wir uns systematisch der Lösung.

Um darzustellen, warum auch der zweite Ansatz regelmäßig scheitert, müssen wir nun etwas weiter ausholen.

Das Iterationsmodell

Was ist das Wesen iterativen Vorgehens? Wie funktioniert es? Ein beispielhaftes Modell zeigt **Abbildung 1**.

Zu Beginn des Projekts liegt das benötigte Ergebnis in einer Wolke. Es ist unscharf und alle Beteiligten, Fachabteilung wie Entwickler, wissen nur grob, wie das Ergebnis aussehen soll. Es werden verschiedene Annahmen getroffen, die wahrscheinlich nur in Teilen zutreffend sein werden. Auf Basis dieser Annahmen wird das Projekt geplant, es wird ein Ziel angepeilt und dieses dann in der ersten Iteration verfolgt.

Am Ende der Iteration hält das Projekt dann inne und schaut zurück:

- Was haben wir tatsächlich erreicht?
- Was wollten wir ursprünglich erreichen?
- Was lernen wir daraus?

Dann wird der Blick wieder nach vorne gerichtet. Durch die zwischenzeitlich ge-



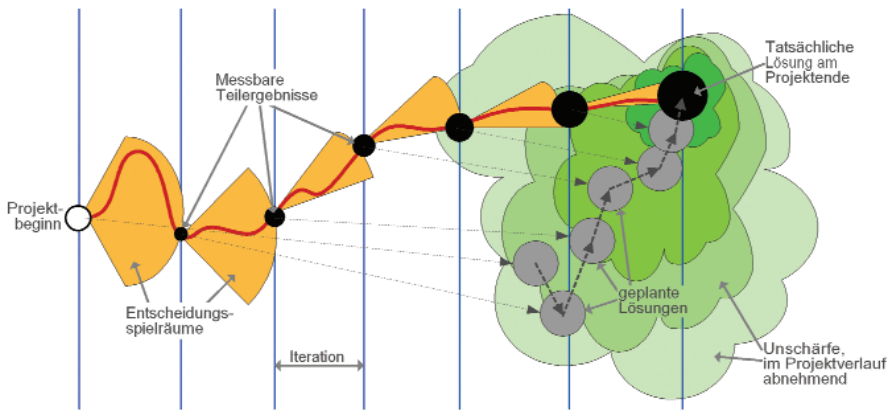


Abb. 1: Das Iterationsmodell

wonnenen Erkenntnisse ist die Wolke etwas kleiner geworden. Das Ergebnis ist immer noch unscharf, aber etwas weniger. Ausgehend von der in der ersten Iteration tatsächlich erreichten Position wird nun ein neuer Plan aufgestellt und wieder ein – nun etwas konkreteres – Ziel angepeilt. Der Prozess beginnt von vorne.

Wichtig hierbei ist, dass in jeder Iteration prinzipiell alle elementaren Entwicklungsaktivitäten (Anforderungen definieren, Lösung konzipieren, Erfolgskriterien festlegen, Lösung entwickeln, Erfolg messen und schließlich Planung aktualisieren) durchlaufen werden (siehe Abb. 2). Spätestens am Ende einer jeden Iteration steht also ein objektiv messbares Teilergebnis, ein Inkrement, eine teilfertige, vorübergehende aber ausführbare Version der angestrebten Lösung.

Verlust der Agilität

Dieses Vorgehen erlaubt Änderungen und Richtungskorrekturen auch während des Projekts und dadurch mehr Beweglichkeit (Agilität). Dennoch wäre es naiv zu glauben, damit seien beliebige Kurskorrekturen zu vollziehen und ständige Änderungen der Anforderungen zu bewältigen. Mit jedem Inkrement wird das Projekt unbeweglicher. Die Agilität ist zwar um Größenordnungen höher als beim einfachen Wasserfallmodell, aber dennoch nimmt die Agilität auch beim iterativen Vorgehen im Laufe der Zeit ab. Die Kosten für Änderungen steigen mit jeder Iteration an und können durchaus ein nicht mehr akzeptables Maß annehmen. Je mehr bereits realisiert wurde, desto schwieriger und teurer ist es, Änderungen einzubringen.

Es gibt eine ganze Reihe von Möglichkeiten, die Beweglichkeit des iterativen

Vorgehens auch zu späteren Zeitpunkten zu steigern bzw. die kontinuierliche Beweglichkeitsabnahme deutlich zu verlangsamen:

- Entwicklung einer geschickten Architektur, beispielsweise mit einem dedizierten Abhängigkeitsmanagement und guten Entwurfsmustern,
- Entwicklung einfacher und testgetriebener Lösungen,
- Testautomatisierung oder
- risiko- und wertschöpfungsorientierte Priorisierung der Anforderungen und Realisierungsschritte.

Um jedoch Architektur-Abhängigkeitsmanagement oder Anforderungs- und Aufgabepriorisierung praktizieren zu können, müssen wiederum die fachlichen und technischen Anforderungen weitgehend bekannt sein. Sind wir damit wieder beim

Wasserfallmodell angekommen? Müssen wir doch wieder am Anfang alle Anforderungen komplett erheben? Nein, der Zweck ist hier ein anderer. Um die Beweglichkeit beim iterativen Vorgehen zu maximieren, müssen Anforderungen nur so weit erhoben werden, dass sie bezüglich ihrer Risiken, Abhängigkeiten, Nutzen und Aufwand grob klassifiziert werden können.

Wir teilen hierzu das Projekt in zwei Phasen ein:

1. Architektur- und Entwurfsphase
2. Konstruktionsphase

Das Grundprinzip bei diesem zweistufigen Vorgehen lautet: Unterscheide die vermutlich stabilen von den vermutlich instabilen Anforderungen und Aufgaben.

In der ersten Phase wird hierzu eine vollständige, aber sehr oberflächliche Übersicht benötigt. Anschließend werden die Anforderungen und Aufgaben in der Reihenfolge ihrer vermuteten Stabilität abgearbeitet: die stabilen zuerst, die instabilen zuletzt. Damit werden unnötige Kurskorrekturen vermieden. Natürlich wird es nicht gelingen, zu Anfang eine absolut vollständige Übersicht über die Anforderungen und Aufgaben zu erzielen – die 80-Prozent-Lösung ist für unsere Zwecke jedoch gewöhnlich ausreichend. Mit „vollständig“ ist der zu diesem Zeitpunkt bekannte Umfang gemeint – unabhängig davon, dass sich dieser im weiteren Verlauf noch ändern wird.

Sofern mit Anwendungsfällen gearbeitet wird, reicht es beispielsweise aus diese jeweils mit Name, Auslöser und Ergebnis

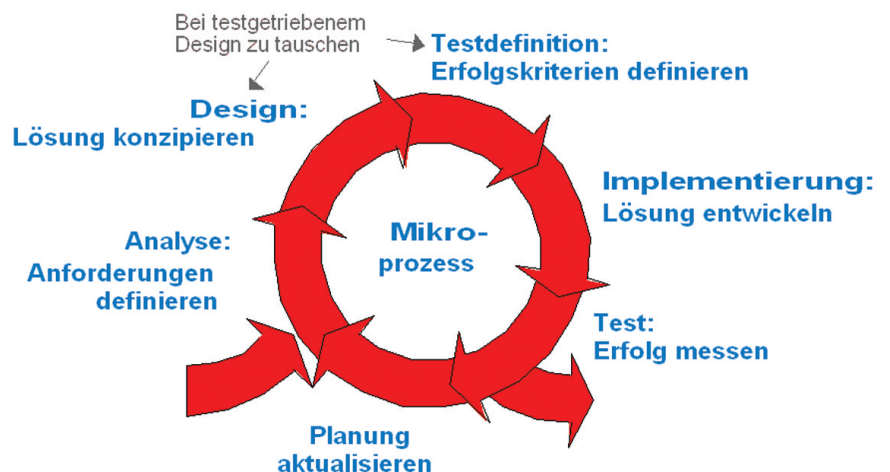


Abb. 2: Elementare Aktivitäten im Mikroprozess



Dauer und Größe von Iterationen

- Iterationsdauer vier bis acht Wochen, je nach Projekteigenschaften. Während des Projekts immer gleich lange Iterationen. Abweichungen von drei Tagen unkritisch.
- Eine Iteration besteht aus einem Arbeits- und einem Orientierungsabschnitt. Der langer Arbeitsabschnitt wird gesteuert durch explizite Arbeitsaufträge und konzentriertes, eigenverantwortliches Arbeiten. Der kurze Orientierungsabschnitt beinhaltet Rückblick, Review der Arbeitsaufträge (Was sollte erreicht werden? Was wurde tatsächlich erreicht? Was blieb offen?), Planungsanpassung und Neuausrichtung des Projekts.
- Gegebenenfalls bedarfsweise so genannte Stabilisierungsiterationen einschieben, bei denen keine neue Funktionalität hinzu kommt, sondern die nur Fehlerkorrektur, Robustheit und Restrukturierung beinhalten. Stabilisierungsiterationen sind deutlich kürzer, beispielsweise ein bis zwei Wochen lang.
- Gesetzmäßigkeiten kleiner und agiler Projekte auf Großprojekte übertragen: kleine möglichst unabhängige Teams von vier bis acht Personen bilden, jedes Team bekommt einen Sprecher als Schnittstelle nach außen.

Kasten 1

zu beschreiben. Diese werden dann priorisiert und nur die wichtigsten werden dann detailliert (vgl. [Vig03]). Die wichtigsten Anforderungen, also solche mit hohem Nutzen oder hohem Risiko, werden möglichst schnell in den ersten ein bis zwei Iterationen realisiert.

Es gibt verschiedene Schulen des iterativen Vorgehens mit jeweils eigener Begrifflichkeit, wobei die Konzepte sich inhaltlich sehr ähneln. Beispielsweise wird in „Scrum“ (vgl. [Sch04]) eine *Timebox* auch *Sprint* genannt. Auch bezüglich der Art und Weise, wie Anforderungen definiert und die Entwicklungsarbeit geplant und gesteuert wird, gibt es unterschiedliche Ansätze, die jedoch alle mit einem iterativen Vorgehen kombinierbar sind. Für den einen sind es *Anwendungsfälle*, *Features* und *Arbeitsaufträge*, für den anderen *User-Stories*,

Task-Cards und *Tests* und für einen dritten sind es *Product-* und *Sprint-Backlog*. Hinter diesen verschiedenen Begriffen verbergen sich zwar ähnliche, aber dennoch spezielle Konzepte, die jeweils spezifische Vor- und Nachteile besitzen. Grundsätzlich passen sie aber alle zu einem iterativen Vorgehen.

Einschränkende Faktoren

Neben der expliziten Berücksichtigung der abnehmenden Beweglichkeit beim iterativen Vorgehen existieren noch weitere einschränkende Faktoren:

- die Iterationsdauer
- der notwendige Planungs- und Steuerungsaufwand
- Intensität und Zyklus des Feedbacks
- Konkretheit und Objektivierbarkeit der Teilaufgaben und -ergebnisse
- Projektgröße (Anzahl der Personen, Dauer)
- Formalisierungsgrad

Die Strategie agiler Softwareentwicklung lautet, den Zeitraum vom Erkennen einer Anforderung bis zum Nachweis ihrer korrekten Umsetzung möglichst kurz zu halten. Dies gelingt nur, wenn die Gesamtaufgabe in entsprechend handliche Teilaufgaben zerlegt wurde. Je konkreter die Teilaufgaben werden, desto eher eignen sie sich für eine möglichst objektive Fortschritts- und Erfolgskontrolle. Je kleiner die Teilaufgaben sind, desto eher sind Rückmeldungen zum Erfolg dieses Teilschrittes möglich. Iterationen sind Lern- und Erfahrungsschleifen. Es ist daher ein wichtiges Ziel die Iterationen möglichst kurz zu halten.

Andererseits besteht beim iterativen Vorgehen ein erheblich größerer Planung- und Steuerungsaufwand. Je kleiner die einzelnen Teilaufgaben sind, desto mehr Planungseinheiten sind zu verwalten. Je öfter Rückmeldungen hierzu erfolgen, desto höher der Aufwand für die Anpassung der Planung. Um diesen Aufwand zu begrenzen, sollten die Iterationen möglichst lang sein.

Mittelweg

Wir haben es also mit zwei widersprüchlichen Zielen zu tun. Der Segen liegt in der goldenen Mitte.

Allerdings spielt die Projektgröße noch eine wichtige Rolle. In kleinen Projekten

„Best Practices“ für iteratives Vorgehen

- Beschreibung der Anforderungen aus Anwender-/Auftraggebersicht (Anwendungsfälle, Features usw.). Entwickler und Anwender priorisieren die Anforderungen gemeinsam.
- Entwicklung von Arbeitsaufträgen auf Basis der priorisierten Anforderungen (jeweils: 1 bis n Mitarbeiter, davon einer als Verantwortlicher, 1 Gutachter). Formulierung objektiv messbarer Ergebnisse, einmalige Aufwandsschätzung durch Beteiligte.
- Wöchentliche Wahrscheinlichkeitsabfragen zur Zielerreichung der Arbeitsaufträge (keine Restaufwandschätzungen) mit Hilfe eines „Läufers“: jede Woche wechselnder Projektmitarbeiter, der am Montag Vormittag alle Arbeitsauftragsverantwortlichen kurz besucht und Wahrscheinlichkeitswerte zu den Arbeitsaufträgen abfragt – und dabei auch viele andere wichtige Informationen erfährt.
- Arbeitsauftragspezifische Fakturierung der Arbeitszeit (für Controlling und Metriken).
- Kontinuierliche Integration, tägliches *Build*, hoher Grad von Testautomatisierung.
- Risiko-Workshops vor jeder Iteration (Identifikation von Risiken, Planung der Risiko-Handhabung).
- Für größere Projekte: Unterteilung in Architektur- und Entwicklungsphase. Meilenstein der Architekturphase: oberflächliche, aber in der Breite vollständige Festlegung aller Anforderungen, Beherrschung aller elementaren Projektrisiken.

Kasten 2

mit wenigen Mitarbeitern bleibt der Planungs-, Steuerungs- und Verwaltungsaufwand auch bei kurzen Iterationen beherrschbar. *XP-Projekte* (*eXtreme Programming*) (vgl. [Bec00]) mit ca. fünf Mitarbeitern und zweiwöchigen Iterationen funktionieren daher gut. Das gleiche für 25 Mitarbeiter ist kaum noch zu beherrschen.

Meine Empfehlung, die sich auch mit anderen Verfahren wie z. B. Scrum deckt,



lautet, Iterationen auf ca. sechs bis acht Wochen auszulegen. Dies ist auch mit 40 Mitarbeitern zu realisieren.

Mit wenigen Projektmitarbeitern können Projekte noch weitgehend informell gesteuert und ohne großen Verwaltungsaufwand durchgezogen werden. Anforderungen und Arbeitsaufträge können mit Karteikarten auf einer Pinnwand erfasst und priorisiert werden. Auch einfache Excel-Listen können noch als leichtgewichtig angesehen werden.

Im Kontrast dazu gibt es umfassende Softwarelösungen fürs Projektmanagement. Vor allem Großprojekte und Großunternehmen sind häufig versucht, den Erfolg ihrer Projekte mit solchen Mitteln sicherzustellen. In vielen Fällen gelingt dies nicht. Je schwergewichtiger die Werkzeuge, desto defensiver und schwerfälliger wird das Projektmanagement. Die mögliche Beweglichkeit eines iterativen Vorgehens kann unter solchen Umständen gar nicht ausgeschöpft werden.

Für Großprojekte muss das Ziel daher sein, das Projekt so zu strukturieren, dass die Gesetzmäßigkeiten kleiner Projekte möglichst weitgehend Gültigkeit behalten bzw. wieder hergestellt werden. In der Praxis heißt dies, eine Gliederung in Einheiten von ca. acht Personen zu erreichen, die dann möglichst unabhängig voneinander, aber im gleichen Iterationstakt (synchrone Team-Iterationen) arbeiten.

Fazit

Iteratives Vorgehen ist ein mächtiges Projektmanagement-Werkzeug, das der qualifizierten und kompetenten Anwendung bedarf, um seine Potenziale und Möglichkeiten in der Praxis erfolgreich zu entfalten. Die Wirkungsweise, mögliche Hindernisse und unterstützende *Best Practices* aus anderen Disziplinen (z. B. Softwarearchitektur), wie sie hier beschrieben wurden, sollten dem Projektmanagement bekannt sein. Sonst sind iterative Projekte genau so problematisch wie Wasserfallprojekte. ■

Literatur

- [Bas75] V. Basili, A. Turner, Iterative Enhancement: A practical Technique for Software Engineering, in: IEEE TSE, S. 390-396, Dezember 1975
- [Bec00] K. Beck, eXtreme Programming, Addison-Wesley, 2000
- [Hru03] P. Hruschka, Iterationen: von 3 Stunden bis 3 Jahre, in: OBJEKTSpektrum 1/03
- [Lar03] C. Larman, V.R. Basili, Iterative and Incremental Development, a brief history, in: IEEE Computer, S. 47-56, Juni 2003
- [Oes00] B. Oestereich, Die Macht des Rhythmus: das OEP-Timepacing-Verfahren, in: OBJEKTSpektrum 2/00
- [Roy70] W. Royce, Managing the development of large software systems, in: Proc. of IEEE Westcon, 1970
- [Sch04] K. Schwaber, Agile Development with Scrum, Microsoft Press, 2004
- [Vig03] U. Vigneschow, C. Weiss, Das Essenzschritt-Verfahren: Aufwandschätzungen auf der Basis von Use-Cases, in: OBJEKTSpektrum 2/03
- [Zur68] F.W. Zurcher, B. Randell, Iterative multi-level Modeling – A Methodology for Computer System Design, Thomas Watson Research Center, New York, 1968

