

IT-Projektmanagement in der modernen Softwareentwicklung

Karsten Hoffmann

Die SW-Entwicklung nach dem Wasserfall-Modell basiert auf klaren Festlegungen zum Projektablauf und zu der zu schaffenden Funktionalität. Sie birgt in der Regel jedoch höhere Risiken, ist unbeweglicher und ergibt nicht selten eine unbefriedigende Produktqualität. Mit dem Spiralmodell und dessen erfolgreicher Umsetzung in zahlreichen Projekten steht ein agilerer SW-Entwicklungsprozess zur Verfügung. Dieser kommt der objektorientierten Vorgehensweise sehr entgegen (Generalisierung – Spezialisierung). Richtig angewandt hilft eine solche Vorgehensweise, technische und kaufmännische Risiken zu vermindern. Außerdem ermöglicht sie eine höhere Flexibilität hinsichtlich kurzfristiger Änderungen, ein einfacheres Qualitätsmanagement, ein leichteres Controlling und eine gleichmäßigere Auslastung. Risiken bestehen hier hinsichtlich klarer Festlegungen und eines damit verbundenen eindeutigen Vertragsmodells (Festpreis). Auch wenn sich in den praktisch etablierten Entwicklungsprozessen Elemente beider Basismodelle kombinieren, verlangen heutige Anforderungen die stärkere Berücksichtigung des iterativen Moments und ein entsprechend gestaltetes IT-Projektmanagement.

Typische Probleme bei heutigen IT-Projekten

Entwicklungsprojekte im IT-Bereich gibt es seit Jahrzehnten, seit über 20 Jahren werden sie durch systematische Projektmanagement-Methoden begleitet. Dennoch leiden viele, insbesondere größere IT-Projekte immer wieder an bekannten „Krankheiten“:

- Die Kosten für die Projektdurchführung werden höher als geplant. Insbesondere in den letzten Phasen Realisierung und Test treten plötzlich deutlich höhere Aufwände zutage.
- Der Fertigstellungstermin wird überzogen. Im günstigeren Fall betrifft dies nur firmeninterne Abläufe, die nun erst später mit Hilfe des neuen Systems abgewickelt werden können, im ungünstigen Fall gehen durch die spätere Inbetriebnahme Kundenumsätze verloren oder es drohen sogar Vertragsstrafen.
- Die erstellte Software hat nicht die gewünschte Qualität. Dies äußert sich z. B. in (zu) hohen Fehlerraten, der Instabilität des Systems, wenig benutzerfreundlichem Dialogverhalten oder auch in einer nicht akzeptablen Performance. Ein anderes Problem ist die oft inaktuelle Funktionalität: Die Software enthält Funktionen, die so gar nicht mehr benötigt werden, es fehlen aber Funktionalitäten, deren Anforderung sich erst im letzten halben Jahr ergeben hat.
- Das fertige IT-System bedarf höherer Aufwände für Administration und Pflege als zunächst erwartet. Die Übernahme des Systems in einen normalen „Tagesbetrieb“ gelingt oft nicht ohne markante zusätzliche Betreuungsaufwände in einer Übergangsphase.

Insgesamt sind die Beteiligten oft nicht glücklich mit dem erzielten Resultat. Die Fachabteilung des Auftraggebers ist enttäuscht, da sie nicht das bekommen hat, was sie längst erwartet hatte, die Geldgeber streiten mit

dem Auftragnehmer um die Schuld für Zusatzkosten, der Auftragnehmer selbst hat wieder mal „draufgezahlt“.

In noch schlimmeren Fällen kommt das neue System gar nicht zum Einsatz, nach langem Streit haben nur noch die Juristen das Sagen, der Projektleiter wird entlassen, ein „Sanierer“ versucht vielleicht noch zu retten, was zu retten ist, Millionenbeträge müssen einfach abgeschrieben werden.

Durch solche und ähnliche Erfahrungen verfestigt sich beim Management der Eindruck, IT-Projekte seien teuer und lohnen sich nicht. Zukünftige Projekte werden nur noch genehmigt, wenn ein früher Return on Investment überzeugend nachgewiesen werden kann.

Aber auch die Gründe für die typischen Probleme insbesondere umfangreicher IT-Projekte sind vielen bekannt. So ist es in einem Projekt mit neuer Technologie und neuen Partnern sehr schwer, die Aufwände bereits zu Beginn genau abzuschätzen; technische Risiken werden oft erst (zu) spät erkannt; QS-Maßnahmen werden nur solange sauber eingehalten, solange das Budget noch im Lot ist – beim ersten Zeitverzug oder bei Stundendefiziten wird oft als Erstes bei den QS-Maßnahmen eingespart, oft werden die Anwender auch zu wenig einbezogen; zur Erhöhung der Aktualität der Anforderungen wird zwar oft ein Change-Request-Verfahren etabliert, doch dieses soll nur in Ausnahmen benutzt werden, da es ja zusätzliche Kosten verursacht; die Administrationsaufwände sind in ihrem Umfang erst zu erkennen, wenn das neue System (zumindest in einer ersten Version) vorliegt.

Viele der genannten Schwächen resultieren aus der für die Software-Entwicklung gewählten Vorgehensweise, die das Auftreten o.g. „Krankheiten“ sehr wahrscheinlich werden lässt. Stattdessen bedarf es eines Soft-

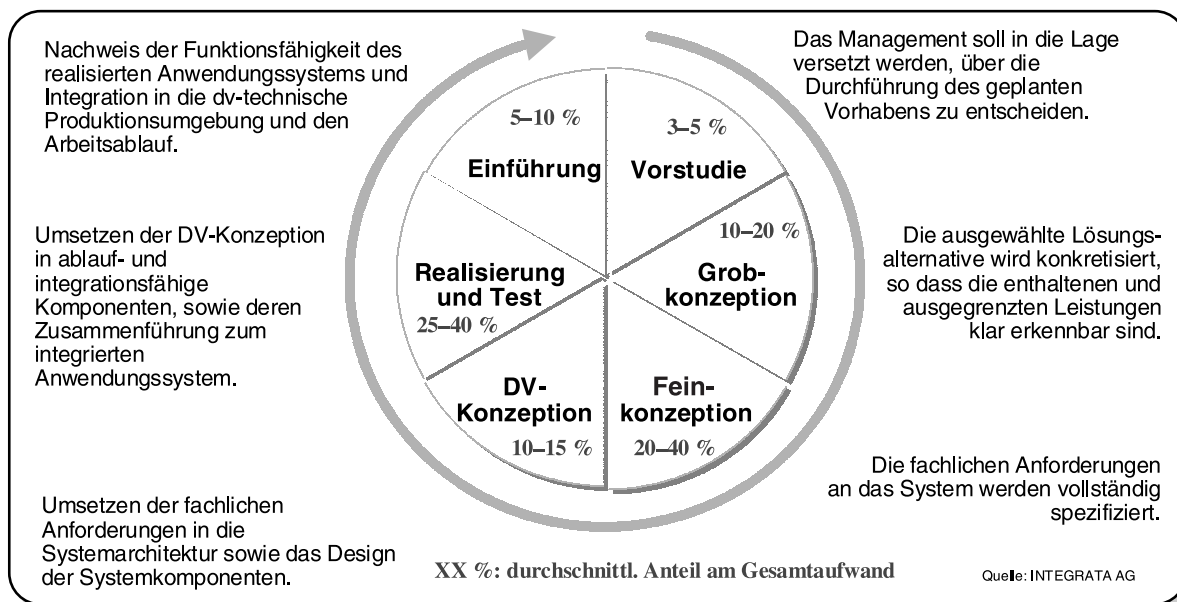


Abb. 1: Firmenbeispiel: Phasen in der strukturierten Systementwicklung

ware-Entwicklungsmodells, das in kurzen Zeiträumen Gestalt, Kosten, Aufwände, Risiken etc. sichtbar und abschätzbar werden lässt – ein iteratives Entwicklungsmodell.

Klassische Vorgehensweise: Das Wasserfallmodell

Firmen, die Software entwickeln (gleich, ob die IT-Abteilung eines großen Anwenders oder ein Systemhaus), haben sich meist dafür ein Regelwerk geschaffen, in dem u.a. das Vorgehensmodell festgelegt und beschrieben ist. Abb.1 zeigt ein solches Phasenmodell, bei dem die Entwicklung des IT-Systems nacheinander die sechs beschriebenen Phasen Vorstudie, Grobkonzeption, Feinkonzeption, DV-Konzeption, Realisierung und Test sowie Einführung durchläuft. In jeder Phase werden bestimmte Ergebnistypen erarbeitet, die zur Grundlage der Begutachtung und Bewertung am Ende der Phase herangezogen werden (dies bildet meistens einen Meilenstein). Bei Mängeln hinsichtlich Vollständigkeit der Ergebnistypen oder auch deren Qualität wird gegebenenfalls eine Nacharbeit verlangt. Erfolgt die Freigabe für die nächste Phase, so bilden die erarbeiteten Ergebnisse den Input für die nächste Phase. Dieses Phasenmodell wird deshalb auch „Wasserfall-Modell“ genannt, da die Ergebnisse jeder Phase den Input für die nächste Phase ergeben. Abhängig von der Projektgröße und dem eingesetzten Personal kann der „Durchlauf“ des gesamten Wasserfalls zwar manchmal auch nur einige Monate dauern, bei Projekten > 1 Mio. EUR liegt er in den meisten Fällen jedoch deutlich über einem Jahr.

Charakteristika des Wasserfallmodells

Wichtigstes Merkmal des Wasserfall-Modells ist es, im Feinkonzept – dies umfasst u.a. das Daten- und Funktionsmodell – alle fachlichen Details des zu realisieren-

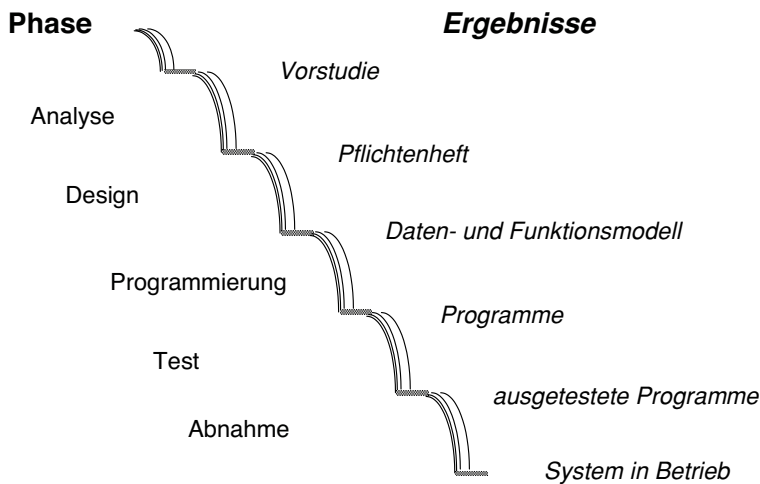


Abb. 2: Wasserfall-Modell

den Systems exakt festzulegen. Diese Festlegung bildet eine eindeutige Bezugsbasis für die Realisierung, für den Test („Test erfolgt gegen die Feinspezifikation“), für die Abnahme und die Gewährleistung. Häufig ist die Feinspezifikation auch Vertragsgrundlage für die zu einem Festpreis zu erbringende Leistung – dabei sind unterschiedlichste Bezeichnungen möglich, vom Pflichtenheft über die Leistungsbeschreibung bis zum Soll- oder Feinkonzept.

Diese eindeutige Festlegung kann in vielen IT-Projekten ein deutlicher Vorteil sein. Der Vorteil der klaren Festlegung für das zu realisierende System kann aber auch zum Nachteil werden: Die Fachabteilung muss sich zum Zeitpunkt der Erstellung der Feinspezifikation eindeutig festlegen. Dies bedeutet auch, dass der vorgesehene Umfang des gesamten Projektes mit der Erarbeitung des Feinkonzepts festgelegt werden muss. Dauert die Umsetzung des Feinkonzepts bis zum realen Einsatz z.B. ein Jahr, so sind die Festlegungen dafür also entsprechend früh vorzunehmen. Es wird meist ein so genanntes Change-Request-Verfahren etabliert, durch das spätere Änderungen noch in den Systemumfang aufgenommen werden können, was jedoch in der Regel Mehraufwände zur Folge hat. Ist die Menge der Änderungen groß, so wird es schwer, entsprechende Nacharbeiten (möglichst) sauber in das Feinkonzept einzubetten, entsprechend sind die DV-Umsetzung, die Realisierungsarbeiten, die Testpläne etc. zu verändern. Bei einigen Projekten führt dies zu deutlichen Verzögerungen, da ein Teil des Teams statt in der Programmierung nun wieder in konzeptioneller Arbeit gebunden ist.

Ein weiteres Merkmal in vielen Entwicklungsrichtlinien nach dem Wasserfallmodell ist, dass die Art der DV-Umsetzung erst mit dem Ende des Feinkonzepts festgelegt wird. Der Vorteil dieser Vorgehensweise liegt darin, dass die zu diesem Zeitpunkt aktuellen Versionen von Werkzeugen, Middleware oder anderer System-Software benutzt werden können. Eine frühere Festlegung hätte stattdessen möglicherweise ein Nachrüsten (Update, neues Produkt etc.) zur Folge. Auf der anderen Seite besteht die Gefahr, dass Mängel in der gewünschten IT-Umgebung erst zu diesem späten Zeitpunkt erkannt werden; dieses stellt bei manchen Pro-

jekten ein erhebliches Risiko dar. Ein weiterer Nachteil dieser Vorgehensweise ist, dass die mit einer Plattform verbundenen technologischen Möglichkeiten keinen Niederschlag in der Spezifikation von Anwendungsdetails finden. Manche Funktionalität ist mit bestimmten Plattformen „leicht“ zu haben, spricht mit wenig Aufwand zu realisieren.

Um die genannten Nachteile zu vermeiden, werden in vielen Projekten Prototypen entwickelt. Durch richtige Prototyp-Definition („vertikaler Prototyp“) gelingt es, die technischen Bedingungen der neuen IT-Plattform auszuprobieren und damit eventuelle Probleme frühzeitig zu entdecken. Durch so genannte Oberflächen-Prototypen kann die Fachabteilung oft leichter erkennen, welche Möglichkeiten das vorgesehene IT-System in seinen Bearbeitungsmasken bietet. Dies hilft, wirtschaftlich effektive Formen der Realisierung für angedachte Funktionalitäten zu erkennen und zu finden.

Bleibt die Frage, ob die entwickelten Prototypen in die zukünftige Realisierung des Systems Eingang finden, und falls nicht, wer die durch Prototyp-Erstellung entstehenden Mehraufwände trägt (der Auftraggeber verlangt dies im Falle technischer Prototypen oft vom Auftragnehmer, da dieser für die gewählte Systemarchitektur in der Regel verantwortlich ist).

Für die Softwareentwicklung nach dem Wasserfallmodell lassen sich einige weitere Nachteile benennen:

- Die erste Hälfte der Entwicklungsarbeiten konzentriert sich fast ausschließlich auf Dokumente. Feinkonzepte von 500 oder gar 1.000 Seiten werden von Fachabteilungen aber nicht immer mit großer Sorgfalt gelesen. Dagegen würde eine frühe erste Version „zum Anfassen“ den Beteiligten der Fachabteilung nicht nur zeigen, wie das neue System etwa aussehen wird, es würde – gerade bei ersten erkannten Mängeln – auch die Sinne für die genauen Formulierungen im Feinkonzept schärfen.
- Mängel auf Grund einer falschen Spezifikation, auch solche grundsätzlicher Art, werden oft erst in der Testphase (am Ende der Realisierung) entdeckt. Eine frühe „erste Version“ macht dies in der Regel sichtbar.
- Der Erarbeitungsprozess in jeder der sechs beschriebenen Phasen wird nur jeweils einmal durchlaufen. Dadurch entsteht wenig Prozesssicherheit und, da erste Erarbeitungen fast immer lückenhaft sind, auch eine geringere Produktqualität. Dies gilt insbesondere für Teams, in denen auch weniger erfahrene Systementwickler sitzen, und für Projekte, die mit neuen Technologien arbeiten.
- Die durch das Wasserfallprinzip erzwungene zeitliche Verteilung der Projektarbeit über die sechs Phasen macht es schwerer, Teams für bestimmte Tätigkeiten (z.B. Programmierer, Testteams, Konzeptersteller) gleichmäßig auszulasten. Auch ist eine „Parallelisierung“ ihrer Tätigkeiten kaum möglich.
- Die Integration des neuen IT-Systems in eine bestehende Umgebung geschieht häufig erst ganz am Schluss. Eine frühere Integration wird auch deshalb gescheit, weil neben der Vermeidung von Mehraufwänden die Fehlerbeseitigung innerhalb des eigenen Systems im Vordergrund steht. Dadurch kommt es am Ende häufig zu einer so genannten „Big-Bang-Integration“, die nicht selten

mit großen Rückschlägen zu kämpfen hat. Dies, da sich zum Beispiel zeigt, dass die vereinbarten Schnittstellen von Nachbarsystemen nicht der Zusage entsprechen, die Kommunikation zwischen den Systemen gestört ist und vieles andere mehr.

- Da jede Phase nur einmal durchlaufen wurde, wird nicht gelernt, mit Änderungen umzugehen. So werden Änderungsanforderungen oft nur lückenhaft in alle bestehenden Ergebnistypen eingepflegt. Das Arbeiten mit Erweiterungen und Veränderungen wurde nicht oder nur wenig „geübt“.

Insgesamt ist das „Re-Engineering“ kaum gelernt worden, so dass im Ergebnis eine Software entstanden ist, die schwer zu warten und zu pflegen sein wird.

Durch Prototyping, frühe Testintegrationen, erweiterte QS-Maßnahmen, gut ausgebildete Projektteams und einige andere Maßnahmen können o.g. Schwächen teilweise kompensiert werden. Es sollte jedoch deutlich geworden sein, dass diese Defizite viel mit dem Prinzip des Wasserfallmodells zu tun haben, also durch das Modell immanent gegeben sind.

Zur objektorientierten Entwicklung passt das Spiralmodell

Seit Ende der 80er Jahre haben sich objektorientierte Methoden zur Softwareproduktion entwickelt und verbreitet. Mitte der 90er wurde aus der Vielfalt unterschiedlicher OO-Methoden durch die Unified Modelling Language (UML) eine einheitliche Plattform geschaffen, die sich inzwischen als Quasi-Standard in der objektorientierten Softwareentwicklung etabliert hat. Das Prozessmodell in der Objektorientierung ist in der Regel ein anderes als in der so genannten strukturierten Programmierung: Kern des Prozesses von objektorientierter

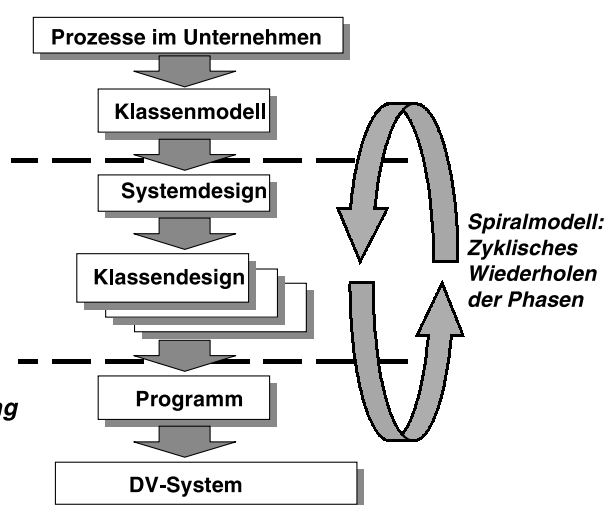


Abb. 3: Objektorientierte SW-Entwicklung

entierter Analyse und Design sind das Auffinden und Definieren von Objektklassen, in denen die gewünschten Daten und Funktionalitäten enthalten sind. Dieser Prozess der Klassenmodellierung wird immer wieder durchlaufen, wobei sich der Fokus der Untersuchung allmählich vom Groben („Generalisierung“) zum Feinen („Spezialisierung“) verlagert. Auf diese Weise wird eine immer weiter gehende Klassenbibliothek aufgebaut, die als Kern des zu schaffenden DV-Systems das wichtigste Ergebnis des Entwicklungsprozesses darstellt.

Aus diesem Grund passt das Wasserfallmodell nicht sonderlich gut zur objektorientierten Software-Entwicklung. Stattdessen wurde das so genannte Spiralmodell als neues Prozessmodell zur SW-Entwicklung „erfunden“ [1].

Beim Vorgehen nach dem Spiralmodell entstehen in regelmäßigen Abständen nacheinander neue Versionen des zu erstellenden DV-Systems. Dazu werden jeweils insgesamt vier Phasen (Analyse, Design, Konstruktion und Test) durchlaufen, so dass am Ende der Testphase jeweils eine neue Version des zu erstellenden Systems zur Verfügung steht.

Charakteristika des Spiralmodells

Bei Entwicklung nach dem Spiralmodell wird mit den wichtigsten Geschäftsobjekten und Geschäftsvorfällen begonnen. Das Zentrum der Spirale soll im fachlichen Kern des neuen DV-Systems angesetzt sein. Nach Analyse der wichtigsten Geschäftsvorfälle (in Form so genannter „Use Cases“) werden durch das OO-Design die zugehörigen Fachobjekte definiert. Die Konstruk-

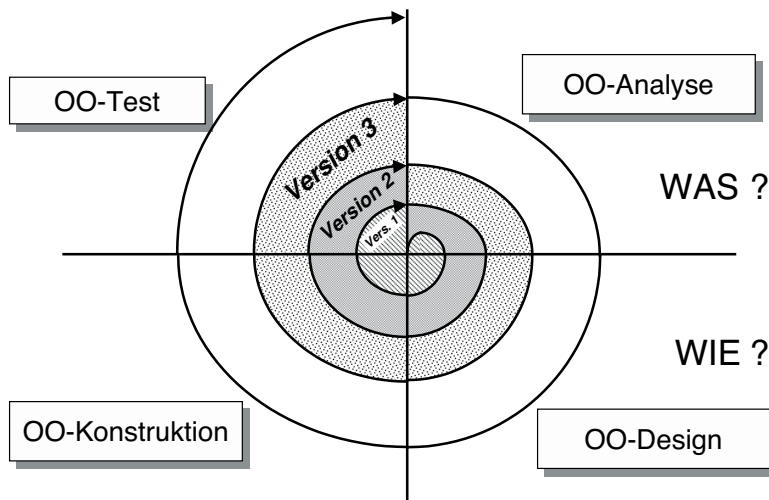


Abb. 4: Spiralmodell

tionsphase umfasst das Programmieren der entsprechenden Klassen, die schließlich in der Testphase einem ausführlichen Anwendungstest unterzogen werden. Auf diese Weise entsteht mit der ersten Version eine „kleine Ausgabe“ des neuen DV-Systems, und auch die Fachabteilung und die zukünftigen Anwender können durch diese Version einen frühen Eindruck dessen gewinnen, was das fertige System in etwa zu bieten hat.

So lässt sich eine bessere Kommunikation mit der Fachabteilung bzw. den Endkunden erreichen und alle Beteiligten erhalten dadurch einen konkreten Bezug zu fachlichen oder auch technischen Fragestellungen. Dies hilft, Missverständnisse hinsichtlich Begriffen, technischen Gegebenheiten, Vorstellungen etc. drastisch zu reduzieren.

Wird mit den wichtigsten Prozessen und Objekten begonnen, so kommt eine solche Vorgehensweise auch der 80:20-Regel entgegen, die viele Systementwickler kennen: 80 % der mit dem DV-System zu erledigenden Aufgaben werden durch Funktionen abgedeckt, die mit etwa 20 % der Aufwände für die Systemerstellung verwirklicht werden. Für die restlichen 20 % der Funktionen (zumeist Spezialfälle, die selten zum Tragen kommen) werden dagegen 80 % der Aufwände benötigt.

Ziel des Spiralmodells ist es also, das neue DV-System zu entwickeln durch Schaffung mehrerer Versionen, die aufeinander aufbauen. Auf diese Weise wird jede einzelne Phase des SW-Entwicklungsprozesses schon sehr früh zum ersten Mal durchlaufen, und alle Beteiligten gewinnen eine größere Sicherheit sowohl hinsichtlich möglicher technischer Probleme als auch hinsichtlich der tatsächlichen Form des späteren DV-Systems. Für

die Strukturierung dieses kontinuierlichen Aufbauprozesses spielen die einzelnen Versionen als „große“ Meilensteine eine wesentliche Rolle. Die Planung und Vorgehensweise **innerhalb** einer einzelnen Version entsprechen im Wesentlichen dem Wasserfallmodell, oft eben angepasst an die Randbedingungen der objektorientierten Softwareentwicklung (z.B. vier statt sechs Phasen, wie oben beschrieben).

Die Entwicklung in Versionsschritten bedeutet auch, dass für eine Version Ergebnisse der Vorversion wiederverwendet, erweitert und gegebenenfalls auch verändert werden. Die Erweiterung eines Zwischenprodukts erhöht die Prozesssicherheit, denn die Beteiligten wissen bei einer zweiten oder dritten Version besser, was wie beschrieben werden muss und auf welche Ergebnisse zum Beispiel besonderer Wert gelegt wird. Dies führt insgesamt zu einer besseren Ergebnisqualität. Durch dieses Vorgehen finden auch Veränderungen an bereits erarbeiteten Zwischenprodukten leichter Eingang in den Entwicklungszyklus. Jedoch ist zu beachten, dass auch bei einer zyklischen, iterativen Entwicklung Änderungen Mehraufwände erzeugen, wenn auch in geringerer Höhe als beim Wasserfallmodell. Die Weiterarbeit an Zwischenprodukten führt insgesamt zu einer höheren Kompetenz des Re-Engineerings, die für die Wartung und Pflege eines „lebendigen“ DV-Systems unbedingt vorhanden sein muss.

Die Software, die auf diese Weise entwickelt wird, wächst also mit jeder neuen Version in ihrem Umfang. Bei der Definition von Versionen bzw. Versionsumfängen werden neben technischen Aspekten insbesondere Einsatzaspekte berücksichtigt. So kann eine frühe Version zu Demonstrations- und Schulungszwecken eingesetzt werden. Versionen, in denen alle dafür wichtigen Teilfunktionalitäten umgesetzt sind, können für einen Pilotbetrieb oder für einen eingeschränkten Kundenkreis bereits eingesetzt werden usw. Aufwändige Funktionalitäten für Spezialfälle werden meistens erst mit höheren Versionen verwirklicht. Ob deren Verwirklichung sich wirklich lohnt, kann im Lichte des Nutzens von frühen Versionen viel besser entschieden werden. Hier hat das Management also auch die Chance, die Entwicklung bei einer bestimmten Version X erstmal zu stoppen – ein Verfahren, das beim reinen Wasserfallmodell nicht denkbar ist. Auch ein „Design to Cost“ ist beim Spiralmodell viel eher möglich, da die Risiken und zu erwartenden Aufwände nach spätestens zwei oder drei Versionen viel genauer abgeschätzt werden können.

Abschließend die Zusammenstellung wichtiger Vorteile des Spiralmodells:

- Jede Phase des Entwicklungsprozesses wird früh zum ersten Mal ausprobiert. Dies hilft, technische Mängel von verwendeten Produkten, mangelndes Know-how u. Ä. früh zu erkennen, und bietet so frühe Korrekturmöglichkeiten. Die wiederholten Entwicklungsschritte verbessern die Prozess- und damit die Produktqualität.
- Das frühe Vorliegen von Zwischenprodukten des Entwicklungsprozesses erlaubt frühzeitige und zeitnahe QS-Maßnahmen. Ein kontinuierliches QS-Management lässt sich bei diesem Verfahren auf einfache Weise implementieren (die hohen Aufwände für das oft monatelange Testen des Systems nach einer

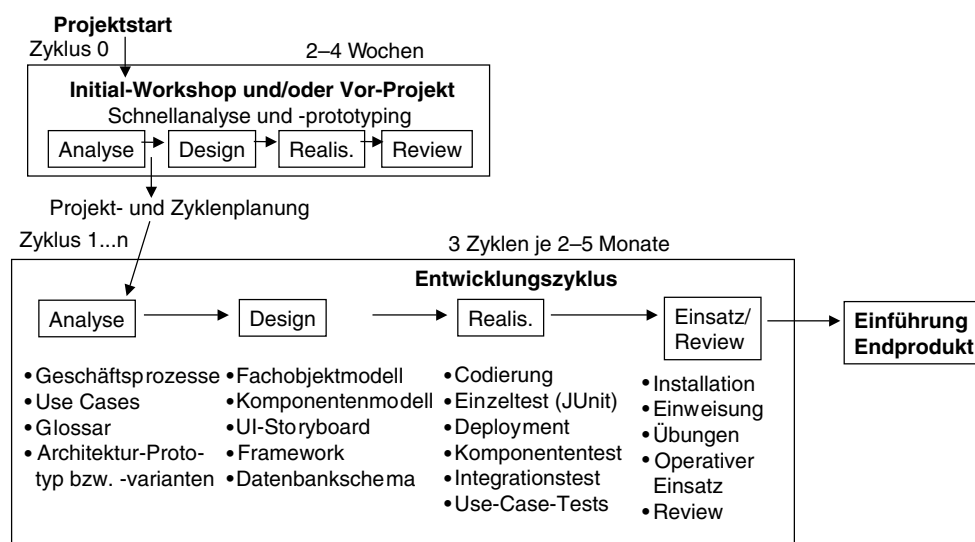


Abb. 5: Organisation des SW-Entwicklungsprozesses in Zyklen

Gesamterstellung können damit drastisch reduziert werden).

- Eine erste Version steht früh zur Verfügung. Das frühe Zeigen von ersten Ergebnissen hilft, die Kommunikation mit dem Endkunden zu verbessern, Missverständnisse auszuräumen und technische Risiken frühzeitig abzuschätzen.
- Tiefere Funktionalitäten für Sonderfälle lassen sich später klären. Dies gibt den Fachabteilungen mehr Zeit zur Klärung, den Entwicklern mehr Sicherheit und dem Management die Chance, über das JA oder NEIN zu speziellen Funktionalitäten erst später zu entscheiden. Die Möglichkeit zu späteren Erweiterungen erleichtert im Übrigen auch der Fachabteilung die Freigabe bzw. Abnahme von frühen Versionen!
- Ergänzungs- und Änderungswünsche lassen sich auf effizientere Art einbringen. Dies vergrößert die Chancen, zeitnah das zu implementieren, was die Anwender wirklich brauchen.
- Der gesamte Entwicklungsprozess lässt sich besser steuern. Verschiedene Entwicklergruppen (Konzeptionierer, Programmierer, Testteams) können gleichmäßiger ausgelastet werden. Über die Versionsplanung werden frühe Teileinsätze, die Beschleunigung der Entwicklung durch Übereinanderschichten von Versionen oder auch die Streckung von Aufwänden eher ermöglicht.

Durch die Elemente des Spiralmodells lässt sich insgesamt ein agilerer Entwicklungsprozess definieren, der den häufigen Änderungsanforderungen in einer schnelllebigeren Welt eher genügen kann.

Prozessorganisation mit dem Spiralmodell

Es gibt viele Möglichkeiten, das Spiralmodell in die praktische Arbeit umzusetzen. Eine der Möglichkeiten ist in Abb. 5 gezeigt. Der vorangestellte Zyklus 0 dient dazu, eine frühe Vorstellung des neuen IT-Systems zu erhalten. Diese Basis kann auch durch ein bereits bestehendes System oder ein Versuchsprojekt (Vorprojekt) als Vorbild gebildet werden. Der Zyklus 0 ist optional;

ist er nicht vorgesehen, muss sehr früh ein Architektur-Prototyp o.Ä. definiert und erarbeitet werden. Denn die im ersten Zyklus zu erarbeitenden Use Cases orientieren sich in der Regel an der Basis-Architektur des zu schaffenden IT-Systems (d.h., die Art der zu definierenden Use Cases steht in einer gewissen Abhängigkeit von der Basis-Architektur). Es ist wichtig, in dieser frühen Phase ggf. die Unterstützung von erfahrenen IT-Architekten zu suchen. Diese wird insbesondere zur Schaffung eines ersten zufriedenstellenden Beispiels benötigt.

Die Länge der Zyklen ist u.a. von der Projektgröße abhängig; je kleiner der vorgesehene Gesamtumfang ist, umso kürzer sollten auch die Zyklen sein. Als Anhaltspunkt seien zwei Beispielprojekte (jeweils knapp 1 Mio. EUR) genannt, bei denen mit Zykluslängen von etwa 3 Monaten gute Erfahrungen gemacht wurden. Jedoch kann die zu wählende Zykluslänge von zahlreichen weiteren Faktoren abhängen. Nach dem Prinzip der rollierenden Planung werden innerhalb eines Zyklus – also einer Version – die einzelnen Aufgaben (bzw. die damit verbundenen Arbeitspakete) genau durchgeplant. Jedes Unternehmen (bzw. jede Entwicklungsabteilung innerhalb eines Unternehmens), das Softwareentwicklung nach dem Spiralmodell betreibt, entwickelt erfahrungsgemäß eine eigene Ausgestaltung des dabei benutzten Entwicklungsprozesses.

1995 wurde UML als standardisierte Methode von objektorientierter Analyse und Design

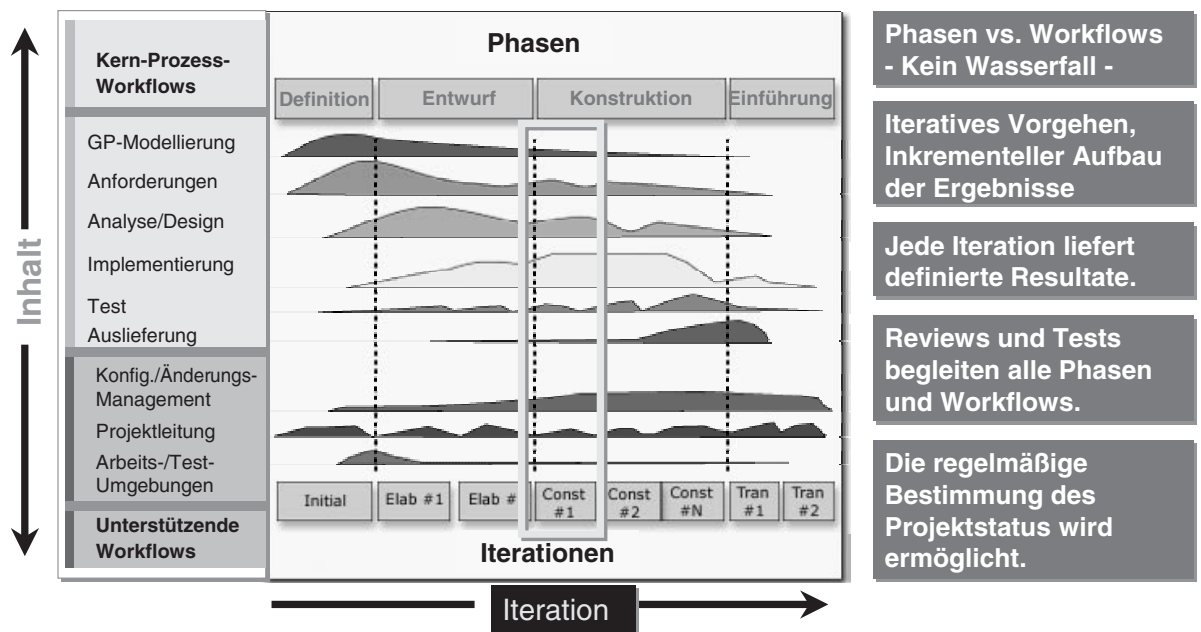


Abb. 6: Rational Unified Process (RUP) als Beispiel für ein spiralförmiges Prozessmodell

entwickelt und – verbunden mit Gründung der Fa. Rational – mit dem Werkzeug Rational Rose auf den Markt gebracht. Erst 1999 wurde endlich auch ein entsprechendes Prozessmodell publiziert, der Rational Unified Process (siehe Abb. 6) [2], [4]. In diesem Prozessmodell werden sechs so genannte Kern-Workflows definiert, mit deren Hilfe das neu zu schaffende IT-System entwickelt werden soll (GP-Modellierung, Anforderungsanalyse, Design, Implementierung, Test und Auslieferung). Zusätzlich werden drei unterstützende Workflows definiert, u.a. Konfigurationsmanagement und Projektleitung. Die über die horizontale Achse aufgetragenen Iterationen (Versionen) werden in vier Phasen aufgeteilt (Definition, Entwurf, Konstruktion, Einführung), bei denen die Phasenbezeichnung das Hauptgewicht der Arbeit der jeweiligen Version ausdrücken soll. Die Kurven zu jedem Workflow stellen die jeweiligen Aufwände dar.

Bei genauer Betrachtung ist zu erkennen, dass der Buckel der größten Aktivität im Vergleich der sechs Workflows allmählich von links nach rechts wandert. In dieser Verlagerung der jeweiligen Hauptaktivität von der GP-Modellierung in der ersten Phase bis zur Auslieferung in der letzten Phase ist die „Spur“ des Wasserfallmodells zu erkennen. Der Unified Process zeigt – im Unterschied zum reinen Wasserfallmodell – klar an, dass schon mit der ersten Version auch mit Realisierungsarbeiten begonnen wird. Bleibt kritisch anzumerken, dass der Autor die Aktivitätskurve im unterstützenden Workflow „Projektleitung“ anders sieht als im Beispiel angedeutet (nämlich eher gleichmäßig verteilt als gezackt).

In der Praxis hat sich in vielen Unternehmen ein Software-Entwicklungsprozess (mehr oder minder klar) herauskristallisiert, der meistens Elemente des Spiralmodells mit Elementen des Wasserfallmodells kombiniert [3]. So ist das V-Modell eine Weiterentwicklung des Wasserfallmodells unter Hinzunahme iterativer Elemente. Ist eine Auswahl oder Modifizierung mög-

lich, so sollte abhängig von der Projektaufgabe das eine oder andere Moment (Wasserfall oder Spirale) stärker betont werden.

Beispiel 1: Auftragsprojekt (Festpreis) für eine einmalige Entwicklung mit klassischer Programmiersprache, gleiche Basisarchitektur wie ein ähnliches Projekt, das als Beispiel vorliegt, ausführliches Pflichtenheft liegt vor, keine Änderungen und keine Weiterentwicklung vorgesehen → Betonung Wasserfall.

Beispiel 2: Entwicklungsprojekt (gemeinsam) einer neuen Anwendung mit objektorientierter Entwicklung wie z.B. Java, Zielarchitektur noch offen, genaue fachliche Beschreibung fehlt noch, Weiterentwicklung ist auf jeden Fall vorgesehen → Betonung Spiralmodell.

Managementaufgaben bei der SW-Entwicklung

Im Folgenden werden einzelne Managementaspekte bei der Softwareentwicklung beleuchtet, die insbesondere unter Verwendung des Spiralmodells Geltung haben.

Projektplanung und Ressourcen-Management

Die wichtigsten Meilensteine im zeitlichen Ablauf sind die Versionen. Es ist darauf zu achten, trotz solider Entwicklungsarbeit möglichst früh eine erste Version zu erreichen („keine Angst vor Lücken oder Fehlern“). Die folgenden Versionen sollten etwa gleiche Umfänge haben, da dies der gleichmäßigen Auslastung des Teams, der Prozess- und Terminalsicherheit jedes Einzelnen und dem einfacheren Controlling entgegenkommt.

Ein wichtiger Aspekt der Strukturierung insbesondere größerer Projekte ist die „intelligente Bildung“ von Komponenten. Unter Komponenten werden in diesem Zusammenhang eigenständige Teilsysteme verstanden, die alleine testbar und lauffähig sind. Für jede Komponente sind sowohl die von ihr **benötigten** als auch die von ihr **gebotenen** Dienstleistungs-Schnittstellen exakt festzulegen – diese Festlegungen haben quasi Vertrags-

charakter. Die „richtige“ Komponentenbildung ist eine Aufgabe für erfahrene IT-Architekten – von einer weiteren Erörterung dieses wichtigen Problems wird hier jedoch abgesehen.

Einem Entwicklungsauftrag liegt entweder bereits ein vorgesehene, festes Budget zugrunde oder es ist früh festzulegen, welches Budget benötigt werden wird und welche Leistung dafür zu erwarten ist. Oft wird ein Gesamtbudget – gegebenenfalls für einen ersten Entwicklungsabschnitt – festgelegt, mit dem ein bestimmter Funktionsumfang in einer bestimmten Zahl von Versionen realisiert werden soll.

Für die Feinplanung der Versionen ist hinsichtlich Termintreue meist der Grundsatz „Termin vor Funktionsumfang“ etabliert. Dies bedeutet, dass der Fertigstellungstermin einer Version auf jeden Fall eingehalten werden sollte, notfalls an der Funktionalität kleine Abstriche gemacht werden. Die Verpflichtung zu unbedingt notwendigen Funktionen kann durch Festlegung von MUSS-Funktionalitäten erfolgen, an SOLL- und KANN-Funktionalitäten einer Version wird gegebenenfalls gekürzt.

Die Aufwandsschätzungen (Soll) und -erfassungen (Ist) sollen in regelmäßigen Abständen (Ist täglich, Rest-Soll wöchentlich) erfolgen. Geschieht dies bereits während der Erstellung der ersten Version, ist mit wachsender Versionsnummer eine umso größere Sicherheit und Genauigkeit im Controlling zu erwarten. Diese oft deutlich bessere Beherrschung der Aufwandsabschätzung auch heiklerer Aufgaben wie Gesamttest oder Systemintegration ist eine der offensichtlichen Vorteile der iterativen Entwicklung.

Die Entwicklungsschritte innerhalb einer Version erfolgen zeitlich hintereinander; die Analysephase einer zweiten Version kann jedoch schon beginnen, kurz nachdem die Analysephase der ersten Version beendet (und qualitätsgesichert!) ist. Ähnlich können die Designer bereits an der neuen Version arbeiten, während die vorige noch im Test ist. Die zeitliche Verzahnung einzelner Entwicklungsschritte auch zwischen zwei Versionen ist also möglich – eine „Übertreibung“ an dieser Stelle geht allerdings zu Lasten der zeitlich noch möglichen Rückkopplungen aus der Vorversion und wirkt damit qualitätsmindernd. Insgesamt hilft die o.g. Verzahnung, die verschiedenen Gruppen des Entwicklungsteams gleichmäßiger auszulasten, außerdem kann sie helfen, sehr enge zeitliche Vorgaben für das Gesamtsystem durch Überlagerung der Versionen eher erfüllen zu können.

Risiko-Management

Das Anstreben einer raschen ersten Version zwingt die Entwickler, alle für eine erste Version des Systems benötigten Teile früh zusammenzumontieren. Dies hilft, technische Probleme zeitig zu erkennen, so dass auch mangelhaftes Know-how, falsche oder fehlende Hersteller-Dokumentation, fehlende System-SW (Treiber, ...) und vieles mehr früh erkannt werden. Entsprechende Korrekturmaßnahmen können früh erfolgen und vielerlei Formen haben wie z.B. Nachschulungen, Hinzuziehen weiterer Experten, Veränderung der Basisarchitektur usw. Auch ein früher Projektstopp kann eine der Alternativen sein, die gegebenenfalls helfen

können, viel Geld zu sparen! Insgesamt schafft eine frühe Version in der Regel mehr Sicherheit und ein geringeres späteres Risiko für alle Beteiligten.

Der Projektmanager muss sein Team anhalten, zwischen zwei Polen der SW-Entwicklung den richtigen Mittelweg zu finden: Sorgfältige Entwickler, insbesondere in modernen Technologien, versuchen oft sehr grundsätzliche Lösungen zu finden (z.B. Design Patterns), deren Umsetzung allerdings möglicherweise eine Menge Zeit benötigt. „Schlampige“ Entwickler versuchen dagegen gelegentlich, bestimmte technische Probleme durch Hintertürchen oder Tricks zu umgehen, ohne das eigentliche Problem zu fokussieren. Auch in frühen Versionen müssen Teile der eigentlichen technischen Probleme bereits gelöst werden, andererseits sind manche „Generallösungen“ oder Frameworks in der Nachbetrachtung oft um einiges umfangreicher als unbedingt nötig. Aufgrund des richtigen Mittelwegs kann eine effektive wirtschaftliche Entwicklung erreicht werden, die durch kontinuierliches Qualitätsmanagement und gelegentliche kritische Selbstreflexion begleitet werden sollte.

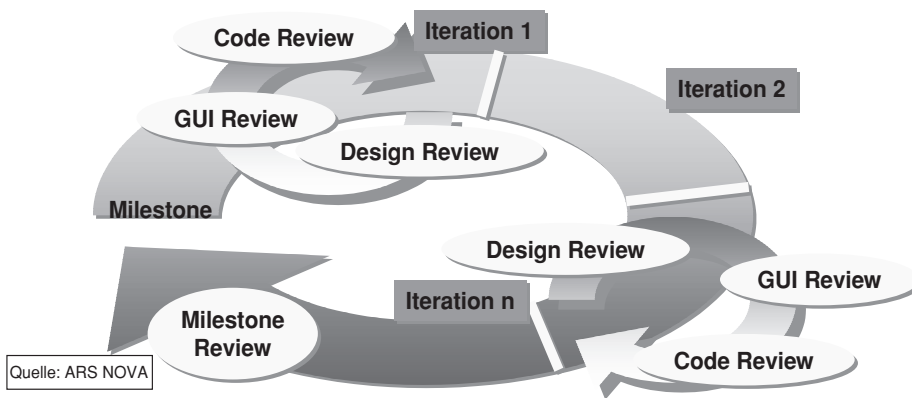
Bei richtiger Prüfung kann der Auftraggeber eine frühe erste (oder zweite) Version umfangreich nutzen:

- Entspricht die erkennbare Oberfläche den Erwartungen der Fachabteilung bzw. Anwender? Ist hinsichtlich Dialogverhalten, Performance oder Zuverlässigkeit die richtige Tendenz zu erkennen?
- Liefern die Dienstleistungs-Schnittstellen das Gewünschte? Ist ein Zusammenspiel oder eine Integration mit anderen Systemen schon möglich?
- Wird durch die ersten Versionen deutlich, dass organisatorische Veränderungen oder erweiterte Schulungsmaßnahmen für die zukünftigen Anwender notwendig sind?
- Welche administrativen Arbeiten müssen an der frühen Version bereits erfolgen, welche weiteren sind erkennbar? Wie funktioniert das System-Management für das neue System?
- Lässt sich eine frühe Version für Messen oder andere vertriebliche Einsätze nutzen? Könnte eine Vorversion für einen ausgewählten Anwenderkreis neue Kunden binden (oder auch abschrecken!)?

Die Beachtung dieser und weiterer Fragen hilft dem Auftraggeber, unvermeidbare Risiken früh zu erkennen und unnötige Risiken zu vermeiden.

Qualitätsmanagement

Für die Qualität der zu entwickelnden Software ist ein ständiges prozessbegleitendes Qualitätsmanagement unverzichtbar. Gerade



- **Review zeitnah bei jeder Version und zu jedem Artefakt**
- **Steigende Qualität insbesondere durch steigende Prozess Erfahrung**
- **Frühes Kunden-Feedback zu UI-Storyboard und neuen Versionen**
- **Reduzierung der Zahl der Artefakte/Dokumente, Konzentration auf diejenigen Artefakte, mit denen wirklich gearbeitet wird**

Abb. 7: Qualitätsmanagement im Spiralmodell – der Review-Zyklus

eine iterative Vorgehensweise ermöglicht kontinuierliche QS-Maßnahmen, erfordert diese aber auch. Gut bewährt haben sich viele kleine Reviews, die innerhalb jedes Versionszyklus am Ende der Erstellung jedes Dokuments bzw. Artefakts (Diagramm, Sourcecode, Programm usw.) durchgeführt werden. Solche Reviews benötigen oft nur 2 oder 4 aufmerksame Stunden eines oder zweier Reviewer, deren Ergebnis protokolliert und zunächst mit dem Projektleiter besprochen wird. Dieser entscheidet über durchzuführende Nachbesserungsarbeiten, ggf. die Veranlassung eines erneuten Reviews (Re-Review) und klärt diese Punkte mit dem verantwortlichen Entwickler. Durch zeitnahes Nachbessern wird vermieden, dass entdeckte Qualitätsmängel in die nächsten Bearbeitungsschritte durchschlagen.

Durch kleine Reviews schon innerhalb des ersten Entwicklungszyklus wird für die zweite Version bereits eine Menge gelernt. Die Prozessqualität und damit auch die Ergebnisqualität steigt normalerweise in der Folge mit jeder neuen Version. Alle Beteiligten wissen, worauf es besonders ankommt und welche fehlerkritischen Stellen besonderer Aufmerksamkeit bedürfen. Dies lässt sich ergänzen um Hinweise, wo die Mitarbeiter genauere Aussagen erwarten und die Artefakte also zu verbessern sind.

Erfahrungsgemäß sind kontinuierliche kleine QS-Maßnahmen wirtschaftlicher als die Vernachlässigung der QS während des Entwicklungsprozesses, um dann am Ende einen entsprechend langen Zeitraum mit umfangreichen Tests vorzusehen. Der größte Gewinn für die Qualität entwickelt sich durch die Rückkopplung, durch das Lernen aus Fehlern; genau dies ist bei einer Untergewichtung prozessbegleitender QS-Maßnahmen kaum möglich. Im iterativen Prozess gelingt es leichter, konstruktive QS-Maßnahmen („verhütende“ Maßnahmen vor dem jeweiligen Entwicklungsschritt) und analytische QS-Maßnahmen (prüfende Maßnahmen am

entwickelten Produkt) intelligent miteinander zu verschränken. Denn aus gefundenen Fehlern lassen sich in der Regel leicht Maßnahmen entwickeln, die solche Fehler in Zukunft vermeiden helfen.

Gerade für den wiederholten Entwicklungszyklus ist die Einführung z.B. von Metriken zur systematischen Qualitätsmessung mit geringem Erhebungsaufwand möglich.

Ein weiteres Qualitätsmerkmal ist durch die frühe Existenz einer Kundenversion gegeben. Diese erlaubt ein direktes Kunden-Feedback, durch das insbesondere auch Einfluss auf die Ausgestaltung der weiteren Versionen genommen werden kann. Bei richtig angelegten effektiven Rückkopplungsprozessen hilft dies, ein IT-System zu erarbeiten, mit dem sich die Kundenseite vollständig identifiziert.

Schließlich erlaubt es der frühe Rückkopplungsprozess, die Existenzberechtigung der Einzelergebnisse des SW-Entwicklungszyklus

zu überprüfen. Erfahrungsgemäß werden in gut ausgearbeiteten Entwicklungsprozessen eine größere Anzahl von Artefakten (in der Hauptsache Dokumente) erzeugt. Kritisch zu prüfen ist jedoch, mit welchen Artefakten tatsächlich gearbeitet wird. Ist jedes Diagramm notwendig, soll heißen, gibt es einem Designer, Entwickler oder Tester notwendige Anhaltspunkte für seine nächsten Arbeitsschritte? Diese kritische Frage resultiert aus der Erfahrung, dass einige Ergebnisse lediglich aus Gründen der Vollständigkeit erarbeitet werden. Benutzt sie niemand zur Weiterarbeit, so sind sie im Erarbeitungsprozess auch nicht unbedingt erforderlich. Also kann aufgrund der Erfahrungen aus den ersten zwei Versionen entschieden werden, welche Artefakte wirklich notwendig sind. Dies hilft, den Produktionsprozess schlank zu halten, was in der Regel die Qualität der tatsächlich zu erarbeitenden Artefakte erhöht.

Change-Management

Neue technische Möglichkeiten, organisatorische Änderungen durch Fusionen, Abspaltungen oder Outsourcing, der Ruf nach neuen Kundendienstleistungen oder auch neuen Märkten: Dies alles sind Faktoren, die Änderungsanforderungen an IT-Systeme verlangen. Dies wirkt nicht nur auf bestehende, sondern auch auf solche IT-Systeme, die gerade erst im Entstehen sind. Die zügige Berücksichtigung wechselnder Anforderungen erfordert ein effektives Change-Management, das bereits kurz nach Projektstart installiert sein sollte. Durch einen iterativen Entwicklungsprozess wird das Einüben eines effektiven Change-Managements gefördert (Kritiker meinen, diese Möglichkeiten verführen dann auch zu mehr Änderungen; dies ist aber das grundsätzliche Problem von gewachsenen Freiheiten, die natürlich auch missbraucht werden können).

Auch im Spiralmodell erzeugen Änderungen an bereits

erstellten Konzepten, Modellen oder Programmen einen Mehraufwand. Jedoch ist dieser in der Regel deutlich geringer, da die Auswirkungen solcher Änderungen meist weniger „erstellten Bestand“ betreffen und außerdem die Ausprägung von Spezialisierungen oft erst in höheren Versionen erfolgt. Die Versionierung verfolgt u. a. das Prinzip, mit steigender Versionszahl vom Allgemeinen zum Speziellen zu kommen. Aus dieser Sicht ist die Erweiterung eines Programms in Form der Berücksichtigung eines Spezialfalles einer der Standardfälle in der iterativen Entwicklung. Deshalb wird das dazu notwendige Procedere früh ausprobiert und gelernt, so dass das Change-Management ein natürlicher Bestandteil der iterativen Entwicklung wird. Dieser Umstand kommt insbesondere der späteren Wartung und Pflege des fertig erstellten Programms zugute.

Gewachsene Möglichkeiten des Change-Managements erlauben dem Projektmanagement insgesamt eine flexiblere Vorgehensweise. Es gelingt damit eher, relativ schnell zu einer tragfähigen ersten Lösung zu kommen, die je nach den dann bestehenden Randbedingungen im Projektumfeld eingefroren, erweitert oder verändert werden wird. Es kommt außerdem zu einem größeren Vertrauen in den zukünftigen Entwicklungsprozess (eine ganz andere Mentalität hat sich oft in Organisationen mit „Wasserfallmodell-Kultur“ entwickelt: „Wenn wir schon die Projektfreigabe endlich bekommen haben, dann muss in das Pflichtenheft alles rein, was wir seit 2 Jahren und für die nächsten 2 Jahre wünschen.“).

Ständige Verschlingung als Daueraufgabe

Für ein neues Projekt wird in vielen Unternehmen auf eine bereits bestehende Entwicklungsumgebung zurückgegriffen. Es kommen dann so genannte Frameworks zum Einsatz, die z. B. bei objektorientierter SW-Entwicklung aus Klassenbibliotheken bestehen, in denen eine Vielzahl von Systemfunktionen bereits verwirklicht sind. Für den Entwicklungsprozess wird in der Regel eine Menge von Artefakten (oder Ergebnistypen) vorgeschrieben, die im Laufe des Entwicklungsprozesses erzeugt werden sollen. Für beide Aspekte gilt es zu überprüfen, was im neuen Projekt tatsächlich übernommen werden soll; Verschlingung ist hier das zentrale Stichwort.

Gerade zu Beginn ist kritisch zu prüfen, welche Artefakte wirklich gebraucht werden und erstellt werden müssen (siehe entsprechenden Abschnitt unter Qualitätsmanagement). Wiederverwendung ist in der Objektorientierung ein wichtiges Ziel. Aber auch für den Entwurf und die Erstellung von Klassen gilt es genau abzuwägen, welche Vorarbeiten wieder verwendet werden sollen oder müssen. Je größer der „Rucksack“ ist, mit dem eine neue fachliche Entwicklung bereits beginnt, umso größer der Aufwand, die damit eingebrachten Elemente kennen zu lernen (um sie auch einsetzen zu können), zu pflegen und weiterzuentwickeln.

Auch in der Güterproduktion müssen solche Verschlingungsprozesse immer wieder durchlaufen werden, um schlagkräftige Produkte mit hoher Qualität zu erhalten.

Eine ähnliche Haltung muss Daueraufgabe des Projektmanagements im SW-Entwicklungsprozess sein, dies gilt gerade für iterative Entwicklungsmodelle. Denn hier ist

einerseits die Gefahr groß, im Projekt unnötigen Ballast aufzuhäufen, andererseits ist aber auch die Chance gegeben, beim Überarbeiten einer Version „auszumisten“. Beim „eXtreme Programming“ (XP), ein besonders extremes iteratives Entwicklungsverfahren, wird dieser Prozess als so genanntes „Refactoring“ immer wieder – oft täglich – durchgeführt.

Das Ringen um Verschlingung ist eine Managementaufgabe, die sich in vielen Aufgabenfeldern – auch außerhalb der IT – stellt (auch dabei gibt es natürlich Übertreibungen). Für die iterative SW-Entwicklung ist dies ebenfalls sehr angeraten, hilft es doch, den Entwicklungsprozess effektiv und damit auch agil zu halten.

Vertragsprobleme und andere Risiken beim Spiralmodell

Vorteil des Wasserfallmodells ist es, nach Abnahme des Feinkonzepts eine klare und saubere Grundlage zu haben, die oft Basis eines Realisierungsvertrages (mit Festpreis) ist. Bei unklaren Formulierungen im Feinkonzept oder dem Auftreten von Change Requests kann es jedoch zu Unstimmigkeiten und einem „Nachverhandeln“ der Aufwände kommen.

Beim Spiralmodell liegt eine klare Beschreibung über den Inhalt einer Version erst dann vor, wenn die entsprechenden Entwurfs- und Designdokumente erarbeitet sind. In der Vorplanung können zwar wesentliche Funktionalitäten grob beschrieben werden, doch dieses taugt nicht als „wasserfeste“ Vertragsgrundlage. Vielmehr müssen sich Auftraggeber und Auftragnehmer darüber im Klaren sein, dass die Vereinbarungen zum Umfang einzelner Versionen jeweils erst innerhalb des Entwicklungszyklus festzulegen sind. Durch Vorgabe von MUSS-Funktionen kann der Auftraggeber gewisse Sicherheiten erreichen, doch das Gelingen des Projekts ist wesentlich von einer vertrauensvollen Zusammenarbeit während des Entwicklungsprozesses abhängig.

Um wirtschaftliche Rahmenbedingungen zu setzen, wird häufig ein Budget festgelegt, in dessen Rahmen eine bestimmte Anzahl von Versionen mit definierten Mindestfunktionalitäten zu realisieren sind. Aufgabe der Projektplanung ist es, eine entsprechende Versionsplanung mit anschließenden Feinplanungen umzusetzen. Die „Vertragsphilosophie“ bekommt mit dem Spiralmodell eine andere Ausrichtung. Im Spiralmodell sichert der Auftragnehmer zu, mit seiner Vorgehensweise effektiv und transparent das neue DV-System zu entwickeln. Der Auftraggeber „kauft“ den **Entwicklungsprozess** ein, weniger ein fertig bestimmtes **Werk**. Für die erfolgreiche Projektdurchführung ist eine enge Mitarbeit des Auftraggebers notwendig, denn nur gemeinsam kann eine Win-win-Situation erzielt werden.

An dieser Stelle darf nicht verschwiegen werden, dass das iterative Vorgehen eine stärkere Mitwirkungs- und Entscheidungspflicht des Auftraggebers erfordert. Kommt dieser – wenn auch nur über kurze Zeiträume – seinen Pflichten nicht nach, kann dies den Entwicklungsprozess erheblich behindern. Im positiven Fall erwirkt diese Mitwirkung ein zielgerichtetes und effektives Arbeiten im Sinne aller Wünsche des Kunden. Deshalb ist auf Kundenseite ein effektives Informations- und Entscheidungsmanagement notwendig, um die Vorteile des iterativen Modells tatsächlich zu nutzen.

Durch Festlegungen während der jeweils zu erarbeitenden Version kann im Spiralmodell jedoch genauso eine vertragliche Grundlage geschaffen werden, auf deren Basis eine Systemabnahme (für die Version) und eine Gewährleistungspflicht vereinbart werden können. Auch diese Vereinbarungen müssen iterativ geschlossen werden. Insgesamt gibt es auf diesem Gebiet noch wenig Erfahrungen. Offensichtlich gilt jedoch, je besser die Vertrauensbasis, umso leichter sind die Hürden bei den vertraglichen Regelungen zu überwinden. Insbesondere der steuerbare iterative Prozess kann helfen, dieses Vertrauen aufzubauen.

Ausblick

Zum Schluss eine vielleicht etwas gewagte weitergehende Betrachtung: Es lassen sich Parallelen finden zwischen der SW-Produktion nach dem Wasserfallmodell und der Taylorisierung in der materiellen Produktion. Für bestimmte Situationen („Massenproduktion“, starre Festlegungen) können beide durchaus jeweils die effektivste Organisationsform sein. Je individueller und auf wechselnde Kundenbedürfnisse zugeschnitten jedoch die SW-Produktion erfolgen soll, umso eher scheint eine iterative Entwicklungsmethode angeraten zu sein. Eine Parallele zur „post-fordistischen“ Produktion ist auch hier zu erkennen. Iterative Entwicklungsmethoden verlangen mehr Feedback, erzwingen daher auch eine bessere Zusammenarbeit verschiedener beteiligter Gruppen auf Seiten des Entwicklerteams und des Kunden¹. Insgesamt schaffen iterative Methoden eine größere Abwechslung in der SW-Entwicklungsarbeit (im Sinne von job enrichment). Sie erfordern eine kontinuierlichere Tätigkeit von Seiten des Projektmanagements und einen intensiveren Abstimmungsprozess mit dem Kunden.

Auch wenn es auf diesem Gebiet bisher noch wenig ausgereifte Erfahrungen gibt, werden zurzeit neue Wege ausprobiert (z. B. XP), ist also noch vieles zu lernen und zu optimieren. Hierfür und für den vermehrten Einsatz von iterativen und rückkoppelnden Prozessen spielt soziale Kompetenz eine wachsende Rolle – ein Trend, der nicht nur für das IT-Projektmanagement gilt. ■

Literatur

- [1] Boehm, B. W.: *A Spiral Model of Software Development and Enhancement*. IEEE Computer, 21(5); S. 61–72, 1988
- [2] Kruchten, P.: *Der Rational Unified Process*. Addison Wesley, München 1999
- [3] Reinhold, M.: *Rational Unified Process 2000 versus V-Modell '97*. Gesellschaft für Informatik, 8. Workshop der Fachgruppe 5.1.1. Glashütten/Taunus (Hessen) 7.–8. März 2001
- [4] Versteegen, G.: *Projektmanagement mit dem Rational Unified Process*. Springer, Berlin u. a. 2000

¹ So berichten Entwickler, die mit eXtreme Programming arbeiten, dass ehemalige Programmier-Einzelkämpfer durch Einsatz von XP zunächst zu neuen Formen der Teamarbeit gezwungen werden müssen, sich jedoch bald das soziale Klima im Team deutlich verbessert.

Schlagwörter

IT-Projektmanagement, iterative SW-Entwicklung, Softwareentwicklung, Spiralmodell, SW-Entwicklungsprozess, Wasserfallmodell



Autor

Dr. Karsten Hoffmann ist als freiberuflicher Coach und Berater auch für größere IT-Projekte tätig. Er hält außerdem Seminare zu modernen DV-Architekturen und zum IT-Projektmanagement. Nach dem Diplom in Mathematik promovierte er im Bereich Fertigungstechnik. Während 10 Jahren Tätigkeit bei einer IT-Unternehmensberatung sammelte er zahlreiche Projekterfahrungen. Er lebt und arbeitet im Raum Stuttgart, ist seit 1999 selbständig und seither auch Mitglied der GPM. Seit 1. 1. 2003 leitet er das Steinbeis-Transfer-Zentrum IT-Projektmanagement in Stuttgart.

Anschrift

*Steinbeis-Transfer-Zentrum IT-Projektmanagement
Gorch-Fock-Str. 1
D-70619 Stuttgart
Tel.: 07 11/47 26 26
Fax: 07 11/4 79 26 28
E-Mail: KHoffmann@hitpm.de*