Erfolgreiches Management von Software-Projekten

**Karl-Rudolf Moll · Manfred Broy
Markus Pizka · Tilman Seifert
Klaus Bergner · Andreas Rausch**

Für Unternehmen der unterschiedlichsten Branchen ist hohe Produktivität und Qualität bei der Entwicklung betrieblicher Informationssysteme von strategischer Bedeutung.

Die Erstellung maßgeschneiderter Lösungen ist teuer, zeitaufwändig und für die mittel- bis langfristige Wettbewerbsfähigkeit entscheidend. Dennoch verlaufen gerade große Software-Projekte in der Praxis selten zufriedenstellend. Die Ursachen liegen in der sträflichen Vernachlässigung der besonders kritischen, nichttechnischen Erfolgsfaktoren. In dem vorliegenden Artikel werden diese Faktoren in Erinnerung gerufen und mit Erfahrungen aus der Praxis belegt. Daraus leiten sich Leitlinien für das Projektmanagement ab, die für sich genommen nicht neu sind, aber sowohl in der Praxis als auch in gängigen Vorgehensmodellen immer noch zu wenig Beachtung finden. Es wird ein pragmatisches Vorgehen empfohlen, das die identifizierten Erfolgsfaktoren in Entwicklungs- und Managementprozessen berücksichtigt und durch breite Erfahrung und wissenschaftliche Systematik abgesicherte Methoden in der Praxis zur Wirkung bringt.

Bedeutung und Realität von Software-Projekten

Die Bedeutung von Software wächst weiter dramatisch. Beispielsweise sind heute in der Automobilindustrie nicht nur Fahrzeugtechnik und Fahrzeugproduktion von Software abhängig, auch wesentliche betrieblichen Abläufe – von der Produktions- und Personalplanung bis hin zum Vertrieb – können ohne zugeschnittene Software nicht aufrechterhalten werden.

Allein Schätzungen der Standish Group zufolge werden in den Vereinigten Staaten von Amerika jährlich etwa 200.000 Software-Projekte mit einem Kostenaufwand von etwa 274 Milliarden Dollar durchgeführt [25]. In Deutschland wurden laut Bitkom im Jahr 1999 etwa 28 Milliarden EUR im Bereich der Softwareentwicklung umgesetzt [3, 4].

Einen wesentlichen Anteil an diesem Gesamtvolumen haben betriebliche Informationssysteme[1]. Hierzu zählen angepasste Standard-Software, wie SAP-Produkte, aber auch unternehmensspezifische, „proprietäre“ Informationssysteme. Software dieser Art erreicht inzwischen einen Durchdringungsgrad in der Geschäftsprozessunterstützung von 60–90% [4, 26].

Ungeachtet der enormen wirtschaftlichen Bedeutung ist die Erstellung dieser Software-Systeme nach wie vor mit großen Schwierigkeiten und Missverständnissen verbunden. Nicht weniger als 46% aller Software-Projekte leiden an signifikanten Termin- und Kostenüberschreitungen oder reduzierter Funktionalität des Endprodukts gegenüber den Erwartungen der Nutzer. Noch gravierender ist, dass etwa 28% aller Software-Projekte schon vor Fertigstellung abgebrochen werden. Lediglich etwa

[1] Abkürzend bezeichnen wir im Folgenden betriebliche Informationssysteme auch als Software bzw. Software-Systeme.

DOI 10.1007/s00287-004-0412-y

K.-R. Moll · M. Broy · M. Pizka · T. Seifert
Technische Universität München,
Institut für Informatik,
Garching

K. Bergner · A. Rausch
4Soft GmbH,
München

K.-R. Moll
Technische Universität München,
Institut für Informatik,
85748 Garching
E-Mail: broy@in.tum.de

26% aller Software-Projekte werden termin-, kosten- und funktionsgerecht abgewickelt [25].

Gelingt es einem Unternehmen dennoch ein Software-Projekt bis zur Einführung durchzuführen – die Chancen stehen immerhin bei etwa 72% – so steht es vor der nächsten Herausforderung: Zwischen 60 und 80% der Gesamtkosten eines Software-Systems fallen in den folgenden 15 Jahren für die Pflege und Weiterentwicklung an [10].

Hält man sich vor Augen, dass Spitzenentwickler mit gut durchdachten Vorgehensweisen um den Faktor 10 bis 20 produktiver sind als durchschnittliche [6], so wird das große Potential für Kostenreduzierungen sichtbar.

Zusammenfassend gilt: Für die meisten Unternehmen sind Effektivität und Einsatzfähigkeit ihrer betrieblichen Informationssysteme entscheidend. Trotzdem sind heute nur wenige Unternehmen in der Lage, Entwicklung und Wartung (inklusive der langfristigen Weiterentwicklung) ihrer Software erfolgreich und effektiv durchzuführen. Die Wettbewerbsfähigkeit der Unternehmen, die in diesem Bereich Defizite besitzen, leidet empfindlich.

Die geschilderte Situation ist nicht akzeptabel. Die Erfolgswahrscheinlichkeit, die Produktivität und Kosteneffizienz von Entwicklung, Wartung und Betrieb betrieblicher Informationssysteme müssen signifikant gesteigert werden.

In diesem Artikel zeigen wir, dass dieses Ziel mit überschaubarem Aufwand erreicht werden kann. Hierzu sind weder ein revolutionäres, neues Vorgehensmodell noch einzelne innovative Techniken notwendig. Der Schlüssel liegt vielmehr in der konsequenten Umsetzung gesicherter Methoden, so dass diese in der Praxis nachhaltig zur Wirkung kommen.

Um dies zu erreichen, werden wir im Folgenden zunächst die kritischen und anerkannten Erfolgsfaktoren für Software-Projekte anhand eigener praktischer Erfahrungen in Erinnerung bringen. Hieraus ergeben sich Anforderungen an das Projektmanagement und den Entwicklungsprozess, die in zwei Stufen umgesetzt werden. Im ersten Schritt wird ein allgemeiner Leitfaden für erfolgsorientiertes Vorgehen in der Softwareentwicklung (EViSE) skizziert, der dafür sorgt, dass die vorgestellten Erfolgsfaktoren stärker berücksichtigt werden. Im zweiten zeigen wir exemplarisch, wie diese allgemeinen Vorgaben für Unternehmen konkretisiert und in bedarfsgerechten Stufen eingeführt werden können.

Wie bereits erwähnt, beschränken wir uns bei unserer Darstellung auf betriebliche Informationssysteme, wenngleich viele Aspekte zweifellos auch auf andere Systems übertragbar sind. Der Grund für diese Beschränkung ist, dass wir bei betrieblichen Informationssystemen auf langjährige, persönliche praktische Erfahrungen zurückgreifen können. Vier der sechs Autoren dieses Artikels waren mindestens für zwei Jahren selbst in betrieblichen Software-Projekten tätig. Einer der Autoren beschäftigt sich seit 34 Jahren in verschiedenen Verantwortlichkeiten, von der Programmierung bis zur Leitung der IT in einem Großunternehmen, mit Software-Entwicklung und -Wartung. Bei allen Beispielen in diesem Artikel handelt es sich um eigene Erfahrungen unserer in Summe etwa 50-jährigen praktischen Tätigkeit.

Erfolgsfaktoren

Die Standish Group hat bereits 1994 in einer Umfrage das Management von 365 Unternehmen nach den wesentlichen Erfolgsfaktoren für Software-Projekte gefragt [24]. Im Jahr 2000 wurde diese Studie erneut durchgeführt. Die ermittelten Erfolgsfaktoren haben sich im Vergleich zu 1994 wenig verändert. Sind die sieben wichtigsten Erfolgsfaktoren erfüllt, so liegt die Erfolgsquote der Software-Projekte nach [23] bei über 84%!

Die restlichen 16% Erfolgswahrscheinlichkeit verteilen sich auf verschiedene Aspekte, wie stabile Anforderungen und verlässliche Schätzungen. Insgesamt nur 6% Erfolgswahrscheinlichkeit ist auf die Frage der eingesetzten Software-Technik zurückzuführen. Dies ist aus unserer Sicht eine Unterschätzung, die uns allerdings nicht verwundert. Schließlich wurden Manager befragt, die der Software-Technik in der Regel keine besonders große Bedeutung beimessen.

Die von der Standish Group ermittelten Erfolgsfaktoren beziehen sich auf die initiale Software-Entwicklung und sind diesbezüglich mit unseren Erfahrungen [13], aber auch mit denen anderer Autoren [9, 19], weitgehend deckungsgleich. Wie wir eingangs gezeigt haben, sind die Wartung und der Betrieb von Software weitere, wichtige Aspekte, die nicht vernachlässigt werden dürfen. Hierfür und um der Unterschätzung technischer Faktoren entgegenzuwirken haben wir die Erfolgsfaktoren der Standish Group gemäß unseren Erkenntnissen modifiziert bzw. ergänzt (s. insbesondere EF7:

„Anforderungen“, EF8: „Angemessenes Vorgehensmodell“, EF10: „Motiviertes und kompetentes Team“). Die mit den Zitaten angegeben Prozentzahlen der Relevanz wurden unmodifiziert übernommen.

Im Folgenden beschreiben wir kurz die wichtigsten zehn Erfolgsfaktoren und zeigen anhand von Beispielen, welche schwerwiegenden Auswirkungen jeweils eine Missachtung des entsprechenden Erfolgsfaktors haben kann.

EF1: Unterstützung durch die Geschäftsführung

(Standish Group: Executive support, 18%)

Die Geschäftsführung muss einheitlich hinter den Entwicklungszielen und der Einführungsstrategie des betrieblichen Informationssystems stehen. Entscheidend ist darüber hinaus, dass der Projektleiter durch schnelle und konsequente Entscheidungen bei der Ressourcenzuteilung, beim Setzen von Prioritäten sowie bei der Lösung von Konflikten unterstützt wird. Nur so bekommt das Projekt den nötigen Rückhalt im Unternehmen.

Beispiel: Der Vorstand eines Finanzdienstleisters hatte beschlossen, das seit 25 Jahren eingesetzte Informationssystem zur Unterstützung von Vertrieb und Produktverwaltung von Grund auf neu zu entwickeln.

Finanzvorstand und Bereichsleiter IT (engl. Chief Information Officer, CIO), eingesetzt von dem auch für Informationsverarbeitung verantwortlichen Vorstand, bildeten den Projektlenkungsausschuss. Der Vorstandssprecher und Vertriebsvorstand waren nicht direkt in das Projekt eingebunden. Ihre Mitarbeiter legten allerdings viele Anforderungen an das zu entwickelnde System fest.

Eine einheitliche Unterstützung des Projekts durch die Geschäftsführung war damit nicht gegeben. Von der Geschäftsführung gemeinsam akzeptierte und kommunizierte Projektziele existierten nicht.

Jedes Mitglied der Geschäftsführung versuchte über seine Mitarbeiter Einfluss auf das Projekt zu nehmen. Die Projektziele veränderten sich dadurch permanent. Das Projekt wurde nach einer Überziehung des Gesamtbudgets um 600% abgebrochen.

EF2: Einbeziehung der Nutzer

(Standish Group: User involvement, 16%)

Selbst wenn ein Projekt termin- und kostengerecht abgewickelt wird, ist es eine Fehlinvestition, wenn wesentliche Anforderungen der Nutzer nicht erfüllt werden[2]. Deshalb ist es essenziell, dass zukünftige Nutzer eines Systems die Entwicklung durch Vorgaben, Reviews, Erprobungen, Evaluation, Tests etc. begleiten.

Beispiel: In einem Software-Projekt mit einem Budget im zweistelligen Millionen-EUR-Bereich galt es, die Funktionalität einer neuen fachlichen Komponente zu spezifizieren. Das Team überzeugte die Projektleitung, Workshops mit den Anwendern des Systems zu veranstalten.

Im Workshop wurde versucht, die Funktionalität und die Oberfläche der neuen Komponente zu erarbeiten. Früh im Workshop ergab sich die Frage, welche Produkte dem Kunden über das System angeboten werden sollten. Die Antwort des „Anwenders“: „Das kann ich Ihnen aus dem Stegreif nicht sagen, aber vor einigen Monaten habe ich das bereits einem Programmierer im zweiten Stock erzählt. Der müsste es wissen.“

Nach einigem Hin und Her stellte sich heraus, dass die vermeintlichen Anwender nicht wirklich die Nutzer des Systems waren, sondern im Unternehmen nur die Schnittstelle zu den Kunden des Unternehmens, den tatsächlichen Anwendern, repräsentierten. Neben ihren eigentlichen Aufgaben hatte man ihnen noch zusätzlich die Spezifikation der Anforderungen an die Software übertragen. Sie waren also nur ein Puffer zwischen den Nutzern und dem Entwicklerteam. Dabei gingen viele Anforderungen verloren, wurden verfälscht oder frei erfunden.

Nach Einführung des Gesamtsystems mussten über die Hälfte der Masken nochmals angepasst werden. Dies führte zu über 5 Millionen EUR zusätzlichen Entwicklungskosten.

EF3: Erfahrene Projektleiter

(Standish Group:
Experienced project manager, 14%)

Der Projektleiter nimmt eine Schlüsselrolle für den Erfolg eines Software-Projekts ein. Er bildet die

[2] In der ersten Umfrage der Standish Group war dieser Erfolgsfaktor sogar noch an der ersten Stelle [24].

Schnittstelle zwischen Auftraggeber und Entwickler und ist für den reibungslosen Ablauf des gesamten Projekts verantwortlich. Ein erfolgreicher Projektleiter benötigt deshalb neben der durchgängigen Projektverantwortlichkeit über alle Phasen des Projekts hinweg ein grundsätzliches Verständnis sowohl der Fachlichkeit wie der Software-Technik und auch Erfahrung im Bereich der Organisation und der Menschenführung.

Beispiel: In einem Software-Projekt mit einem Volumen von mehreren Millionen EUR wurde an Fachkonzept und Architekturentwurf gearbeitet. Die Voraussetzungen waren günstig: Ein gutes Pflichtenheft lag bereits vor, erste Pläne und Teilergebnisse wurden schnell und in hoher Qualität erarbeitet, die Kommunikation mit dem Kunden gestaltete sich angenehm und kooperativ.

In dieser viel versprechenden Situation schaffte es der Projektleiter, der noch nie an einem Software-Projekt dieser Art beteiligt war, quasi im Alleingang, das Projekt zum Scheitern zu bringen. Durch das Zurückhalten von Informationen zwischen Geschäftsleitung, Anwendern und Entwicklern brachte er zunächst die Kommunikation im Projekt zum Erlahmen. Als sich dadurch erste Komplikationen ergaben, schlug er sich auf die Seite des Auftraggebers und übertrug die unvertrauten und lästigen Aufgaben bei Planung und Controlling sowie die Leitung der Entwicklung vollständig an einen externen Mitarbeiter, den er vor der Geschäftsleitung, Auftraggebern und Anwendern abschottete.

Der Auftraggeber entzog dem Auftragnehmer schließlich auf Grund mangelnden Vertrauens das Projekt und übertrug es einem anderen Unternehmen. Insgesamt ergab sich für das Projekt eine Verzögerung von etwa einem Jahr und ein zusätzlicher Aufwand von etwa 3 Millionen EUR.

EF4: Eindeutige Geschäftsziele und Verantwortung

(Standish Group: Clear business objectives, 12%)

Die Geschäftsziele eines Software-Projekts müssen präzise festgelegt, kommuniziert und überprüft werden. Dabei ist besonders entscheidend, dass es in der Geschäftsführung eine klare Verantwortlichkeit für die Projektkosten sowie die Formulierung, Abstimmung und Umsetzung der Geschäftsziele inkl. des konkreten Nutzens gibt.

Beispiel: Die Entwicklungs- und Betriebskosten des neuen Gepäckabwicklungssystems eines internationalen Flughafens waren so hoch, dass sich die Investition erst nach 1000 Jahren amortisiert hätte [8, 7]. Das heißt, die alte Gepäckabwicklung hätte noch 1000 Jahre zu denselben Kosten betrieben werden können.

Das primäre Geschäftsziel des Projekts war aber Kostenersparnis durch Rationalisierung. Das Projekt wäre in dieser Art nie gestartet oder zumindest frühzeitig gestoppt worden, wenn dieses Geschäftsziel von den Verantwortlichen kommuniziert und überprüft worden wäre.

EF5: Minimierung der Projektgröße

(Standish Group: Minimized scope, 10%)

Die Größe eines Software-Projekts hat einen bedeutenden Einfluss auf dessen Erfolg. Je größer und komplexer ein Projekt ist, desto schwerer ist es zu beherrschen und desto wahrscheinlicher ist es, dass das Projekt scheitert. Nach [25] ist die Erfolgswahrscheinlichkeit bei einem Projekt mit einem Personalaufwand größer als 3 Millionen Dollar höchstens 15% und bei über 10 Millionen Dollar ist sie nahezu Null. Das Projekt wird, wenn überhaupt, kaum ohne Termin und Kostenüberschreitungen sowie Abstriche bei der Qualität abgeschlossen werden können!

Beispiel: Eine Kooperation von drei Finanzdienstleistern schrieb ein Festpreisprojekt zur Neuentwicklung eines umfangreichen Abwicklungssystems aus. Die Ausgangsbasis war ein erster Anforderungskatalog von etwa 25 Seiten. Der Auftrag wurde an den billigsten Anbieter, ein weltweit agierendes Software-Haus, vergeben.

Unter der Regie des Auftragnehmers wurde daraufhin ein umfassendes Fachkonzept entwickelt, das die Wünsche aller drei Auftraggeber repräsentierte. Aufgrund dessen war es extrem komplex. Erst nach etwa drei Jahren Laufzeit mit einem großen Entwicklungsteam waren die Ergebnisse des ersten Implementierungsmeilensteins testbar. Die Testergebnisse waren bzgl. Funktionalität und Performance inakzeptabel. Das Projekt wurde abgebrochen, da die Hochrechnung der ursprünglichen Projektkostenschätzungen einen Faktor vier ergab. Der gesamte Schaden betrug mehr als 50 Millionen EUR.

Sicherlich wäre es unrealistisch generell zu fordern, dass nur noch kleine bis mittelgroße Projekte

durchgeführt werden sollten. In vielen Fällen ist es jedoch tatsächlich möglich ein Software-Großprojekt, das auf umfangreichen Anforderungen beruht, in mehrere kleinere, isolierte Einzelprojekte zu zerschlagen. Durch die Separation erhöhen sich nicht nur die Erfolgsaussichten der Einzelprojekte sondern auch die des übergeordneten Managements, da durch die zusätzliche Strukturierung erheblich weniger Abhängigkeiten im Auge zu behalten sind.

EF6: Standardisierte Software-Infrastruktur

(Standish Group: Standard software infrastructure, 8%)

Die funktionellen Anforderungen an ein Software-System ändern sich häufig schnell. Dessen ungeachtet sollte zumindest die Software-Infrastruktur und -Architektur stabil sein, damit sich die Software-Entwickler auf ihre eigentlichen Aufgaben konzentrieren können.

Beispiel: Bei einem Dienstleistungsunternehmen mit einem großen Filialnetz wurden die unterschiedlichen Software-Anwendungen zur Sachbearbeitung in verschiedenen Fachgebieten ohne eine standardisierte Software-Infrastruktur entwickelt.

Nach etwa 6 Jahren stand eine Vielzahl von Anwendungen zur Verfügung. Es existierte aber weder ein einheitlicher Style-Guide für die Benutzungsoberfläche einschließlich der Hilfe-Funktion noch gab es ein einheitliches Rechtekonzept mit einmaliger, zentraler Authentifizierung (Single-Logon). Steuerungsdaten waren redundant auf 54 unabhängige Datenbank-Tabellen verteilt. Die Anwendungen waren folglich umständlich zu bedienen sowie schwer zu warten oder zu erweitern.

Die Vereinheitlichung der Benutzeroberflächen einschließlich Single-Logon-Verfahren kostete etwa 30 Personenjahre. Die Konsolidierung der Steuerungsdaten kostete etwa 5 weitere Personenjahre und eine einheitliche Hilfefunktion nochmals etwa 5 Personenjahre.

EF7: Stabile grundlegende Anforderungen

(Standish Group: Firm basic requirements, 6%)

Als erste Stufe eines Projekts wird ein minimaler Satz von grundlegenden, möglichst stabilen Anforderungen festgelegt, der zentrale Funktionalitäten für das Unternehmen realisiert und als Basis für weitere Entwicklungsstufen dienen kann.

Beispiel: In dem bei EF1 beschriebenen Projekt wurde entschieden, die gewünschte Funktionalität parallel für alle Produkte zu entwickeln. Dies führte dazu, dass bis zum ersten testbaren Software-System drei Jahre vergingen. In dieser Zeit wurde sowohl das Daten- als auch das Objektmodell ständig geändert. Durch das Verletzen des Grundprinzips der Entwicklung in sequentiellen Phasen musste ein signifikanter Teil der Arbeitskraft für das redundante Ändern vorhergehender Arbeit und nicht für die Entwicklung neuer Funktionalitäten aufgebracht werden.

Es wäre vermutlich besser gewesen, zunächst das Objektmodell an Hand eines Produktes stabil zu entwickeln und zu testen um dann in weiteren Teilprojekten die restlichen Produkte zu ergänzen. Zusätzlich wären dadurch erste Ergebnisse früher nutzbar gewesen.

EF8: Angemessenes Vorgehensmodell einschließlich Qualitätssicherung für den gesamten Software-Life-Cycle

(Standish Group: Formal methodology, 6%)

Wenngleich folgende Grundregel aus wissenschaftlicher Sicht banal erscheint, so wird sie in der Praxis – insbesondere nach der „Schneller-kürzer-billiger-und-.com-Euphorie“ der vergangenen Jahre – keineswegs stets beachtet und muss deshalb explizit in Erinnerung gerufen werden.

Es sollte ein standardisierter Entwicklungs- und Wartungsprozess inklusive Qualitätssicherungsmaßnahmen etabliert sein, der auf die spezifischen Bedürfnisse eines Unternehmens angepasst ist. Dabei muss besonders auf die folgenden Aspekte geachtet werden:

- Minderung der Risiken durch ein Vorgehen in Phasen.
- Entlastung der Entwickler durch die Bereitstellung entsprechender Methoden und Werkzeuge, die den Prozess instrumentieren.
- Sicherstellung der qualitativen Anforderungen wie der Einhaltung vereinbarter Standards oder der Konsistenz und Vollständigkeit der Phasenergebnisse durch konkrete und gelebte Prozesse zur Qualitätssicherung.

Beispiel: In einem großen Entwicklungsprojekt war das anfänglich kleine Entwicklerteam von

weniger als fünf Personen auf insgesamt über 50 Entwickler angewachsen. Während der gesamten Entwicklung wurde angestrebt, möglichst schnell umfangreiche neue Funktionalität zu entwickeln. Für die Bereitstellung einer angemessenen Verfahrenstechnik standen nur marginale Mittel und Ressourcen zur Verfügung.

Nach insgesamt fünf Jahren Arbeit ohne ein dokumentiertes Vorgehensmodell mit entsprechenden Standards und Qualitätskontrollen und mit ungenügend integrierten Werkzeugen war neben dem Quellcode eine unüberschaubare Vielzahl an uneinheitlich strukturierten Modellen und Dokumenten entstanden (Dateien im Datei- oder Versionsverwaltungssystem, Daten in Datenbanksystemen, E-Mails, Ausdrucke etc.). Dies führte dazu, dass die Kosten für die Weiterentwicklung und Wartung des Systems untragbar wurden: Entwickler konnten erforderliche Informationen nicht oder nur nach stundenlangem Suchen finden und weigerten sich, bestehenden Code zu ändern oder anzupassen, da sich bei jeder Änderung unvorhergesehene Effekte ergeben konnten.

Die Projektleitung entschloss sich, ein angemessenes Vorgehensmodell zu definieren und in diesem Rahmen auch die Dokumentation neu zu strukturieren. Allein die Sammlung, Kategorisierung und Neuablage der relevanten Dateien und Dokumente (insgesamt über 20.000, bei einer anfänglichen Schätzung von 1.500) kostete mehr als 3 Personenjahre. Die zusätzlich erforderliche Zeit für eine inhaltliche Aufarbeitung und Konsolidierung wurde von den Bearbeitern auf 30 bis 100 Personenjahre geschätzt. Hinzu kam der bereits während der Entwicklung aufgelaufene Produktivitätsverlust.

EF9: Verlässliche Schätzungen

(Standish Group: Reliable estimates, 5%)

Verlässliche Schätzungen von Aufwendungen und Nutzen eines Software-Projekts sind zum einen eine wesentliche Grundlage für die Entscheidung des Managements, ob ein Projekt durchgeführt werden soll und zum anderen eine Voraussetzung für eine realistische Planung des Projektablaufs.

Beispiel: In dem bei EF1 beschriebenen Projekt wurde statt einer, von fachlicher und technischer Seite bestätigten Wirtschaftlichkeitsberechnung, bestehend aus erwarteten Kosten und dem zu erzielenden Nutzen des Projekts, nur eine grobe Schätzung der Entwicklungskosten des CIO als Basis für den Start des Projekts akzeptiert.

Im Verlauf des Projekts stiegen die Kosten von geschätzten 7,5 Mio EUR auf 40 Mio EUR für 80% der Funktionalität. Das Projekt wurde eingestellt, insbesondere weil schon die Betriebskosten der neuen Anwendung höher als der Nutzen geschätzt wurden.

EF10: Kompetente und motivierte Mitarbeiter

(Standish Group: Other, 5%)

In den Bereich „Other“ fallen laut der Standish Group u.a. auch technische Probleme. Unbedingt zu den zentralen Erfolgsfaktoren hinzuzurechnen ist jedoch das Personalmanagement. Motivation verbunden mit Kompetenz sind die wichtigsten Produktivitätsmotoren [6]. Es ist eine zentrale Aufgabe des Managements, diese durch sorgfältige Auswahl des Personals, durch Weiterbildungsmaßnahmen, einer entsprechenden Arbeitsumgebung und angemessener Führung sicher zu stellen.

Beispiel: Ein Finanzdienstleister entschied sich dafür, ein neues Internet-basiertes System zu entwickeln. Er beauftragte dafür im Rahmen eines Dienstleistungsvertrags einen international tätigen Software-Dienstleister. Der Auftragnehmer startete mit einem kleinen Team von Experten, vergrößerte jedoch das Team rasch und zog für die Implementierung überwiegend neu eingestellte Anfänger heran. Der Auftraggeber stellte die Arbeitsumgebung bereit. Aufgrund der Raumknappheit saßen bald über 40 Entwickler über Monate hinweg in zwei mittelgroßen Räumen – die meisten mussten sich dabei einen Schreibtisch mit einem oder sogar zwei weiteren Entwicklern teilen.

Die Qualität des entstandenen Systems war insgesamt mangelhaft; viele Implementierungsfehler führten dazu, dass die Kernfunktionalität erst drei bis sechs Monate nach dem geplanten Meilenstein ausgeliefert werden konnte.

Existierende Vorgehensmodelle

Zur Verbesserung der Entwicklung betrieblicher Informationssysteme wurden in den vergangenen drei Jahrzehnten zahlreiche Vorgehensmodelle entwickelt. Der Grundstein für systematisches Software Projektmanagement für große Software-Systeme wurde bereits 1970 mit dem Wasserfallmodell

[20] gelegt. Durch die strikte Sequentialisierung aufeinander aufbauender Projektphasen werden in diesem Modell redundante Tätigkeiten im Projektverlauf vermieden (s. Beispiel zu EF 7). Da hiermit theoretisch maximale Effizienz erzielt wird, bildet das Wasserfallmodell nach wie vor die Grundlage der Vorgehensweise in vielen Unternehmen. Die Kehrseite des Wasserfallmodells ist, dass die Linearisierung der Phasen in der Realität oft nicht aufrecht zu erhalten ist. So entwickeln sich insbesondere bei lang laufenden, d.h. großen Projekten die Wünsche der Nutzer und damit auch die Anforderungen an das System selbstverständlich kontinuierlich weiter. U.a. kann daher die Anforderungsanalyse nie wirklich fertig sein bevor die Implementierung beginnt. Darüber hinaus kann im Wasserfallmodell auch auf falsch liegende Kostenschätzungen kaum reagiert werden. Diese Erfahrungen haben in der Folge inkrementelle und iterative Modelle, wie das Spiralmodell [5] und im Weiteren evolutionäre Methoden, s. Extreme Programming (XP; [2]), hervorgebracht. Abgesehen von zahlreichen Details unterscheiden sich diese Vorgehensmodelle ganz wesentlich im Grad der Linearisierung und Sequenzialisierung der Aktivitäten und Phasen.

Für die weiteren Überlegungen in diesem Papier ist diese Frage allerdings von sekundärem Interesse. Die genannten Erfolgsfaktoren sind weitgehend unabhängig von dem gewählten Prozess. Sie finden sich vereinzelt auch in bekannten Vorgehensmodellen in unterschiedlich starken Ausprägungen wieder. So dient der „on site customer“ in XP klar dem Erfolgsfaktor 2 – der Einbeziehung der Nutzer. Was jedoch fehlt ist eine integrierte Berücksichtigung aller Faktoren EF1 bis EF10 in einem Modell!

Zusätzlich zur Reihenfolge der Phasen besteht natürlich großer Spielraum bei der Ausgestaltung des Entwicklungsprozesses mit Rollen und Aktivitäten. Beispiele für die unterschiedlichen Möglichkeiten sind das V-Modell 97 [27], der Rational Unified Process (RUP; [12]) und Cleanroom [18]. Zweifellos leisten diese ausführlichen Vorgehensmodelle einen wesentlichen Beitrag zur Verbesserung der Software-Entwicklung. Immerhin setzen 50% aller Unternehmen ein definiertes Vorgehensmodell ein und sind ISO-9001-zertifiziert [4].

Zusätzlich entstanden (Quasi)standards für die Bewertung und Verbesserung der Software-Entwicklung in Unternehmen, wie zum Beispiel CMM [16], SPICE [22] und ISO 9001 [11].

Wie eingangs dargestellt, werden trotzdem etwa 74% aller Software-Projekte nicht uneingeschränkt erfolgreich abgewickelt [23]. Folglich scheint die Umsetzung dieser Standards in der industriellen Praxis entweder nicht den gewünschten Erfolg zu gewährleisten oder sie findet nicht wirklich statt.

Die Analyse der tatsächlichen Ursachen dieses Phänomens ist schwierig. Ein Problem, das sich in der Praxis jedoch immer wieder zeigt, ist der große Umfang und die hohe Komplexität einiger Standards. Dies führt dazu, dass der Nutzer den gesamten Maßnahmenkatalog verstehen muss, bevor er den Standard anwenden kann. Da die hierfür benötigte Zeit fehlt, kommen die Vorgaben nicht zur Wirkung.

Im Gegensatz dazu zeigen OpenSource-Projekte [1] wie mit verhältnismäßig einfachen und wenigen Regeln, die dafür aber konsequent gelebt werden, gute Ergebnisse erzielt werden können.

Wir sind weder der Auffassung, dass OpenSource-Vorgehensweisen unmittelbar in betriebliche Umgebungen transferiert werden können, noch, dass viele der Regularien in komplexeren Standards sinnlos sind. Dennoch zeigen diese Beobachtungen, dass der Nutzen eines Vorgehensmodell maßgeblich davon abhängt, dass

- die aufgestellten Regeln kurz und prägnant die wichtigsten Erfolgsfaktoren adressieren und
- in den Projekten auch tatsächlich gelebt werden.

Um diesen Forderungen nachzukommen, wird im folgenden Abschnitt ein Leitfaden für erfolgsorientiertes Vorgehen in der Softwareentwicklung (EViSE)vorgeschlagen, der folgende Merkmale besitzt:

- Konzentration auf die oben genannten, wesentlichen Erfolgsfaktoren des Software-Projektmanagement;
- Fokussierung auf Management-Aufgaben:
  Die Sicherstellung der Erfolgsfaktoren ist vorrangig Aufgabe des Managements. Demzufolge betont *EViSE* Managementaspekte stärker als die Tätigkeiten der Entwickler.
- Konzentration auf Ergebnisse:
  Das Vorgehensmodell legt primär zu erbringenden Ergebnisse fest. Die Aktivitäten, die dazu notwendig sind, sind nachrangig. Dies gibt den Entwicklungsteams die Freiheit, selbst ihre Arbeit so zu gestalten, dass sie

die Ergebnisse effizient und mit hoher Qualität erarbeiten können.

Leitfaden EViSE

Genauso wenig wie die oben aufgezählten Erfolgsfaktoren neu sind, ist ein grundlegend neues Vorgehensmodell notwendig. EViSE versteht sich deshalb als ein pragmatischer Leitfaden, der zeigt, wie vorhandene Entwicklungs- und Managementprozesse so verändert werden können, dass die Erfolgsfaktoren berücksichtigt und gesicherte Methoden in der Praxis zur Wirkung kommen.

Die Anforderungen von EViSE an Entwicklungs- und Managementprozesse gliedern sich in drei Gebiete:

- Organisation,
- Software-Life-Cycle,
- Entwicklungsteam und Arbeitsumgebung.

Pro Aufgabengebiet werden im Folgenden jeweils die zu erreichenden Ziele sowie die Grundzüge deren Realisierung beschrieben.

Die Beschreibung erfolgt technologieunabhängig und dient so als Grundlage für die Detaillierung unternehmensspezifischer Vorgehensmodelle (s. unten).

Organisation

Das Aufgabengebiet Organisation umfasst die Linien- und Projektorganisation generell sowie die projektspezifische Auftragsvergabe und -kontrolle. Damit werden organisatorische Rahmenbedingungen für effektive und effiziente Entwicklung und Wartung von Software geschaffen.

Linien- und Projektorganisation

Als Beitrag zum EF 10 „Motiviertes und kompetentes Team“ befasst sich das Teilaufgabengebiet mit der Gestaltung des organisatorischen Umfelds für Entwicklungs- und Wartungsprojekte. Es werden die folgenden Ziele angestrebt:

- Klare Abgrenzung von Linien- und Projektaufgaben;
- Minimierung des Kommunikations- und Abstimmungsbedarfs zwischen Projektteam und Linieneinheiten;
- Motivationsfördernde Arbeitsgebiete.

Die Linien- und Projektorganisation muss dazu die folgenden Anforderungen erfüllen:

- Klare Festlegung einer, von der Linienorganisation unabhängigen Projektorganisation, einschließlich der Berichts- und Eskalationswege. In jedem Fall soll der Projektleiter an genau eine Instanz, i.d.R. Projektlenkungsausschuss, berichten.
- Definition von Linieneinheiten mit in sich geschlossenen (und, soweit möglich, interessanten) Aufgabengebieten.
- Klare Definition und Zuteilung aller wesentlichen Rollen und Verantwortlichkeiten, insbesondere bzgl.:
  - Formulierung und Abstimmung fachlicher Anforderungen,
  - Entscheidung über die Realisierung von Änderungswünschen,
  - Test und Freigabe für den Praxisbetrieb,
  - kontinuierliche Pflege des Vorgehensmodells.
- Unmissverständliche Festlegung des Prozesses für die Ressourcenplanung und -zuteilung.

Auftragsvergabe/-kontrolle

Im Teilaufgabengebiet Auftragsvergabe und -kontrolle wird durch projektspezifische Festlegungen und eine jederzeit transparente Projektkonstellation den Erfolgsfaktoren EF1 bis EF4 Rechnung getragen. Hierzu gehören:

- Eindeutige, bevorzugt projektübergreifende Festlegung von Entwicklungszielen und Einführungsstrategien;
- transparente Zuteilung von Aufgaben und Ressourcen;
- rasche Entscheidung von Grundsatzfragen;
- regelmäßige Kontrolle des Arbeitsfortschritts einschließlich des Ressourcenverbrauchs.

Das Teilaufgabengebiet wird von dem zu etablierenden Gremium „*Projektlenkungsausschuss*“ realisiert, dem folgende Aufgaben zukommen:

- Definierter Start von Projekten einschließlich Zielfestschreibung;
- Festlegung projektspezifischen Details der Projektorganisation wie Leitung, Berichtsintervalle, Änderungs-Verfahren usw.;
- Ressourcenzuteilung;
- Festlegung der Kontrollprozesse (bezogen auf Meilensteine);
- Konfliktlösung;
- Festlegung der Verantwortlichkeiten bzgl. fachlicher Inhalte, Abnahmen etc.;
- Festlegung der Einführungsstrategie.

Die Mitglieder dieses Gremiums sollten entscheidungskompetente Vertreter derjenigen Bereiche

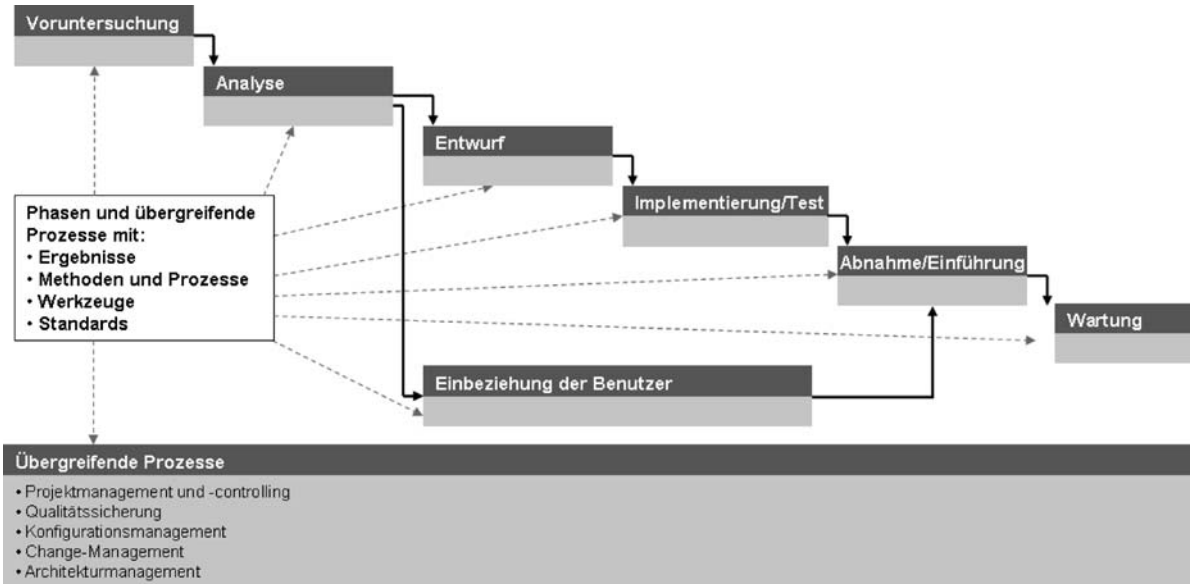

Abb. 1 EViSE SLC

sein, die von den durch das Gremium gesteuerten Projekten operativ betroffen sind. In jedem Fall muss pro Projekt ein „*Auftraggeber*" mit Verantwortlichkeit für *Kosten und Nutzen* festgelegt und im Gremium vertreten sein. Ebenso ist die Zusammensetzung des Gremiums so zu wählen, dass die getroffenen Entscheidungen möglichst breite Akzeptanz finden.

Software-Lebenszyklus (SLC)

Der von uns vorgeschlagene Software-Lebenszyklus (SLC) orientiert sich an der Norm ISO/IEC 12207 „Information Technology Software Life Cycle Processes" und legt alle Ergebnisse fest, die angefangen von der Entwicklung bis hin zur Außerbetriebnahme eines Systems zu erbringen sind.

Wie in Abbildung 1 dargestellt umfasst der SLC sowohl einzelne Phasen der Entwicklung und des Betriebs eines Software-Systems als auch parallel laufende, übergreifende Prozesse.

Der hier skizzierte SLC mag dem Leser wie ein schlichter und evtl. sogar überholter Wasserfall erscheinen. Hierbei sind jedoch einige Unterschiede zu beachten.

Wie in Abbildung 1 links, „Phasen und übergreifende Prozesse mit:", angedeutet, fordert der EViSE SLC, dass nicht nur Phasen und deren temporaler Zusammenhang, sondern für alle Aktivitäten und übergreifende Prozesse konkrete Ergebnisse, anzuwendende Methoden, einzuhaltende Standards und zu nutzende Werkzeuge eindeutig festgelegt sind. Die Erfüllung dieser Anforderung ist Voraussetzung für den Erfolgsfaktors 8 (Vorgehensmodell). In der Praxis zeigt sich immer wieder, dass ein Vorgehensmodell nur dann gelebt wird, wenn Aktivitäten nicht nur eingefordert, sondern mit Methoden und Werkzeugen unterstützt werden, selbst wenn es sich hierbei nur um so triviale Dinge wie Dokumentvorlagen handelt.

Für die Gesamtdarstellung eines solchen SLC eignet sich eine Matrix mit Aktivitäten als Zeilen und den Spalten Ergebnisse, Methodik, Werkzeugunterstützung und Standards. Einen Ausschnitt einer EViSE SLC-Matrix zeigen wir in Tabelle 1.

Phasenkonzept

Durch das Phasenkonzept wird der Lebenszyklus der Software in sieben Phasen gegliedert, die im Regelfall sequentiell abgearbeitet werden. Auch wenn diese Sequentialisierung altmodisch scheint, so gibt es nach wie vor keine belastbaren Nachweise, dass stärker inkrementelle oder evolutionäre Prozesse in der Praxis zu höherer Effizienz führen. Aktuelle evolutionäre Vorgehensmodelle, wie das „Extreme Programming" [2, 14], erfordern Zusatzaufwand für das Modifizieren oder gar Zurücknehmen bereits erarbeiteter Ergebnisse. Die hierfür erforderlichen Aufwendungen sind erstens schwer kalkulierbar, da es sich um nichtantizipierte Änderungen handelt, und zweitens existiert die oftmals angenommene und propagierte Werkzeugunterstützung für Refactoring in der Praxis nicht [17].

Demgegenüber besteht gerade bei betrieblichen Informationssystemen die Chance, mit einem

Tabelle 1

Auszug allgemeine EViSE SLC-Matrix

Ergebnistypen	Methodik und Prozesse (Typen)	Werkzeugtypen	Standardtypen
Spezifikation • Anwendungsarchitektur • Technische Architektur – Hardware – Systemsoftware inkl. Middleware – Datenfernübertragung – Programme, Module, Komponenten – Daten(bank)-Design • Prozessabläufe, Workflows (fortgeschrieben) • Schnittstellen (fortgeschrieben) • GUI-Spezifikation (fortgeschrieben)	Entwurfsmethodik	Entwurfstool	Dokumentvorlage (DV): Spezifikation Blueprint Architektur

Die vollständigen Matrizen stehen unter http://www.4soft.de/EViSE/ zur Verfügung.

stark sequentiellen Vorgehen zum Ziel zu gelangen, da es in diesem Bereich wenige technische Unwägbarkeiten und ein stark exploratives Vorgehen selten begründet ist. Ausnahmen von dem strikten „Wasserfall-Prinzip" sind natürlich möglich, sollten jedoch fundiert begründet sein.

Die Phase „Einbeziehung der Benutzer" läuft parallel zu den Phasen Entwurf und Implementierung. Sie schafft alle Voraussetzungen für eine erfolgreiche Einführung des Software-Systems bei Nutzern: Benutzerhandbücher, Schulungen usw.

Jede Phase sollte durch einen formalen Review abgeschlossen werden. Die Review-Ergebnisse werden dem Projektlenkungsausschuss als Basis für Entscheidungen zum weiteren Vorgehen vorgelegt (s. Abschn. Qualitätssicherung). Arbeitsergebnisse werden nur mit Zustimmung des Projektlenkungsausschusses im Rahmen des Änderungsverfahrens verändert.

Die Ergebnisse der allgemeinen SLC-Matrix müssen in der Regel projektspezifisch angepasst werden. Wird beispielsweise zur Entscheidung der Machbarkeit eines Systems ein Prototyp gebraucht, so muss der Prototyp als Ergebnis der Phase Voruntersuchung vorgesehen sein.

Die Festlegung der Ergebnisse sollte auf jeden Fall folgenden, grundsätzlichen Anforderungen Rechnung tragen:

- Beschränkung der Projektgröße durch Zerlegung in unabhängige Teilprojekte. Unsere Erfahrung hat gezeigt, dass kein Teilprojekt mehr als 10 Personenjahre haben sollte (s. EF5).
- Frühest mögliches Erfassen aller fachlichen Anforderungen pro Teilprojekt inklusive Daten, Funktionen, Masken, etc. unter Einbindung von Benutzern (s. EF2).
- Sicherstellung der Phasenergebnisse in endgültiger Form durch exakte Vorgaben und Qualitätssicherungsprozesse (EF8: „Angemessenes Vorgehensmodell").

Entwicklungsmethodik

Die Festlegung, Bereitstellung und der Einsatz von Methoden und Prozessen nicht nur für den Gesamtprozess sondern auch für jedes zu erbringendes Ergebnis (s. Tabelle 1) ist eine unmittelbare Voraussetzung für die Steigerung der Effizienz und der Produktivität (EF8: „Angemessenes Vorgehensmodell").

Bei der Auswahl konkreter Methoden und Prozesse ist zu berücksichtigen, dass

- die Methoden und Prozesse von den Betroffenen erlernt werden können und durch Werkzeuge unterstützt sind sowie
- das Format der Ergebnisse – wo immer sinnvoll – standardisiert bzw. formalisiert ist.

Werkzeugunterstützung

Werkzeuge sollen die Entwickler von Routineaufgaben entlasten und darüber hinaus auch für die Einhaltung von Qualitätsstandards sorgen. Ein typisches Beispiel hierfür ist der Einsatz von Textverarbeitungssystemen in Verbindung mit Dokumentvorlagen.

Die von uns vorgeschlagenen Arten von Werkzeugen werden in der allgemeinen SLC-Matrix dargestellt (s. Tabelle 1).

Standards

Die Vorgabe von Standards für Ergebnisse ist eine wesentliche Voraussetzung für die Minimierung der Kosten, die für die Erstellung und Begutachtung der Ergebnisse aufzuwenden sind.

Neben offensichtlichen Standards, wie Programmier- und Dokumentationsrichtlinien (Style Guides) sollten auch für andere Ergebnisse, wie der Wirtschaftlichkeitsberechnung, Unternehmensstandards existieren.

Für die Einhaltung der Standards ist folgendes sicher zu stellen:

- transparente Dokumentation der Standards,
- kontinuierliche Aktualisierung
- regelmäßige Kontrolle der Einhaltung (Qualitätssicherung); so weit wie möglich automatisiert.

Übergreifende Prozesse

In diesem Abschnitt befassen wir uns mit phasen- und/oder projektübergreifenden Ergebnissen, mit dem Ziel erhöhte Produktivität und Qualität aus der „Wiederverwendung" von Prozessen und (Zwischen)produkten zu ziehen.

Neben den bekannten Prozessen Projektmanagement/-controlling, Qualitätssicherung und Konfigurationsmanagement gehört hierzu auch das Architekturmanagement einschließlich technischer Basissysteme.

Wenngleich gerade diese Prozesse in der Literatur wohl bekannt sind, müssen auch hier einige Aspekte in der Praxis stärker zur Wirkung gebracht werden. Hierzu gehören die folgenden Punkte, die mit entsprechenden Ergebnissen und Methoden im SLC zu verankern sind.

Projektmanagement und -controlling:

- Sicherstellung adäquater Kompetenz bei der Projektleitung und im Team,
- Festlegung von Kommunikations-, Dokumentations- und Berichtsstrukturen einschließlich Meetings,
- konsequentes Analysieren und Bearbeiten der Risiken,
- konsequentes Änderungsmanagement,
- nachhaltiges Controlling.

Qualitätssicherung:

- Verletzungen von Anforderungen müssen so früh wie möglich erkannt werden um den späteren Korrekturaufwand zu minimieren.
- Reviews sind ebenso eine unverzichtbare Voraussetzung für die Freigabe einer Phase wie durchgehende Testprozesses von der fachlichen Spezifikation der Testfälle, bis hin zu regelmäßigen Regressionstests und der Schwachstellenanalyse einschließlich Verbesserungsvorschläge bei Projektende.

Architekturmanagement (s. EF6):

- Einsatz von Standard-Software anstatt Eigenentwicklung, speziell für technische Funktionen (Help-Funktion, Object-Request-Broker usw.).
- Redundanzfreie Implementierung von fachlichen und technischen Funktionen (beispielsweise Kundendaten nur in der Kundendatenbank, Risikodaten nur in der Risikodatenbank, Berechtigungsdaten nur in der Berechtungsdatenbank, jeweils einschließlich der benötigten Zugriffs- und Verwaltungsfunktionen usw.).
- Die Festlegung und Implementierung einer unternehmensweiten fachlichen und technischen Architektur.
- Auswahl, Implementierung und Pflege von technischen Basissystemen zur Realisierung der Architektur.

Entwicklungsteam und Arbeitsumgebung

Methoden und Werkzeuge können Hilfestellungen bieten und Tätigkeiten spürbar erleichtern. Letztendlich wird die erzielte Produktivität jedoch von der Motivation und Kompetenz der Mitarbeiter bestimmt. Hierfür ist es u.a. essentiell, dass die Mitarbeiter ihre Aufgaben und Ziele kennen und produktivitätsfördernde Arbeitsplätze erhalten (EF10: „Motiviertes und kompetentes Team"). Darüber hinaus stellt die Auswahl kompetenter Mitarbeiter und deren nachhaltige Motivation komplexe Anforderungen an das Management, die hier selbstverständlich nicht erschöpfend diskutiert werden können. EViSE fordert dazu auf, mindestens folgende Punkte zu berücksichtigen:

- Systematische Auswahl und Weiterentwicklung von Mitarbeiter (technische, fachliche, soziale und organisatorische Kompetenz).
- Kompetente Projektleiter und Führungskräfte, die zielorientiert führen und Mitarbeiter fachlich und persönlich fördern.
- Leistungs- und marktorientierte Bezahlung
- Ruhige Arbeitsplätze mit angemessener technischer Ausstattung
- Kommunikationsmöglichkeiten.

Die Produktivität sehr guter Softwareentwickler liegt um ein vielfaches höher als die durchschnittli-

cher. Ihre Ausbildung ist kostspielig und sie sind auf dem Arbeitsmarkt gefragt. Deshalb sind Kompetenzaufbau und Personalpflege von herausragender Bedeutung.

Verbesserung unternehmensspezifischer Vorgehensmodelle mit EViSE

Nach der Diskussion der Erfolgsfaktoren und der Formulierung allgemeiner Anforderungen an erfolgreiches Software-Projektmanagement verbleibt zu klären, wie der allgemeine Leitfaden in der Praxis nachhaltig effektiv werden kann.

Wir schlagen hierzu eine schrittweise Abbildung der Vorgaben aus EViSE auf unternehmensspezifische Bedürfnisse und Prozesse vor. Das Ergebnis dieses Umsetzungsprozesses ist ein Katalog klar definierter Ergebnisse, Standards und Werkzeuge, der sowohl die Anforderungen von EViSE als auch unternehmensspezifische Besonderheiten reflektiert.

Schritt 1: Definition eines unternehmensspezifischen „Best-Practice-Vorgehensmodells" auf Basis von EViSE.

Dazu werden im Unternehmen bereits eingesetzte Verfahren identifiziert und mit Ergebnissen der einzelnen Phasen konkretisiert. Bei der anschließenden Auswahl der unterstützenden Methoden, Prozesse und Werkzeuge wird darauf geachtet, dass es sich um empirisch gesicherte oder zumindest am Markt bewährte Techniken und Verfahren handelt und ihre Hersteller hinreichend zukunftssicher sind.

Zuletzt werden die unternehmensspezifischen Standards festgelegt. Dabei empfehlen wir am Markt vorhandene Standards und ggf. deren Anpassung. Aus unserer Erfahrung sollte der Anteil an Eigenentwicklung bei Methoden, Prozessen, Werkzeugen und Standards so gering wie möglich gehalten werden.

Wir gehen davon aus, dass auf diese Weise Vorgehensmodelle entstehen, die eine gute Abdeckung des anerkannten „Stand der Kunst" aufweisen und bezeichnen sie deshalb als „Best-Practice-Vorgehensmodelle".

Als Beispiel für ein solches „Best-Practice-Vorgehensmodell" zeigen wir in Tabelle 2 einen Auszug einer SLC-Matrix für Java-basierte Softwareentwicklung mit Open-Source-Werkzeugen.

Ein weiteres Beispiel für das Umfeld der „klassischen" Entwicklung von Host-Software auf Basis von MVS, DB2, IMS, PL1 findet man bei [13].

Schritt 2: Vergleich des Vorhandenen mit dem Best-Practice-Vorgehensmodell. Dabei werden die Möglichkeiten zu Verbesserungen identifiziert und entsprechend des zeitlichen Aufwands für die Umsetzung in drei Gruppen gegliedert: kurz-, mittel- und langfristige Verbesserungen.

Kurzfristige Verbesserungen (Quick-Wins) können sofort, spätestens nach drei Monaten umgesetzt werden. Typische Quick-Wins können sein:

- Installation eines Projektlenkungsausschusses;
- Überprüfung wichtiger laufender Projekte in Hinblick auf
  - Ziele,
  - Wirtschaftlichkeit,
  - Einbindung der Nutzer,
  - Qualifikation des Projektleiters;
- Werkzeuge für Basisaufgaben:
  - Versionsverwaltung,
  - Debugger,
  - Testautomatisierung;
- Verbesserung der Arbeitsplätze in technischer Hinsicht.

In der Praxis werden oft gerade durch Quick-Wins Erfolgsfaktoren mit hohem Wirkungsgrad realisiert.

Mittelfristige Verbesserungen können in 4 bis 12 Monaten eingeführt werden. Typische mittelfristige Verbesserungen können sein:

- Vervollständigung des SLC bzgl. wesentlicher Bestandteile, wie:
  - Festlegung von Ergebnissen, beispielsweise Benutzeroberflächen, Testfälle, Mengengerüste, Katastrophenvorsorge,
  - Methoden, z.B. zur Unterstützung von Analyse und Entwurf,
  - Werkzeuge, u.a. zur Unterstützung von Konfigurationsmanagement und Regressionstests,
  - Standards, wie Programmier- und Dokumentationsstandards.
- Einführung von Reviews.

Die mittelfristigen Verbesserungen stellen umfangreichere Veränderungen der Software-Technik dar und sollten in Pilotprojekten erfolgen. Dies fördert die Praxistauglichkeit und die Akzeptanz der Veränderungen.

Die Realisierung der *langfristige Verbesserungen* nimmt mehr als ein Jahr in Anspruch. Typische langfristige Verbesserungen können sein:

Tabelle 2

Auszug Java/Open-Source EViSE SLC

Ergebnis	Methodik und Prozesse	Werkzeuge	Standards
Spezifikation • *Anwendungsarchitekturmodell* – UML Paket- und Komponentendiagrammen und UML Klassendiagrammen für die Modellierung einer modularen Struktur des Anwendungsmodells – UML Sequenz- und Zustandsdiagramme für die Modellierung des Verhaltens der Strukturelemente – UML Aktivitätsdiagramme für die Modellierung der Workflows und übergreifenden Abläufe – UML Klassendiagramme für die Modellierung der Abb. persistenter Klassen auf eine Datenbank • *Technisches Architekturmodell* – Festlegung der Systemarchitektur mit Hardware, Systemsoftware, Middleware und Datenbanken – Spezifikation der technischen Infrastruktur mit Ausarbeitung von Client-/Server-Schnitt Transaktionskonzept Persistenzmechanismus Konfigurationsmanagement inkl. Mandantenfähigkeit Multi-Sprachunterstützung ...	Verfeinerung des UML-Modells aus der Analysephase zu einem fachlichen Anwendungsarchitekturmodell und einem technischen Architekturmodell Die Spezifikation wird dabei aus den Architekturmodellen generiert und zusätzlich mit Text angereichert	*Spezifikation*: OpenOffice *Modellierung*: ArgoUML *Benutzeroberflächen*: v4all	DV: Spezifikation Blueprint Architektur

- Einführung eines Architekturmanagements,
- Verbesserung der Raumsituation für Entwickler,
- Verbesserung der Qualifikation,
- Verbesserung der Führungssituation,
- Aufbau einer Datenbasis für *Projektkenndaten*, insbesondere als Basis für Aufwandsschätzungen.

Diese Aktivitäten langfristig zu planen und umzusetzen ist Aufgabe des IT-Leiters (bzw. Chief Information Officer, CIO).

Die Gesamtaufwendungen für die Verbesserung des Software-Projektmanagements eines Unternehmens mit EViSE sind abhängig vom Ausgangszustand des Vorgehensmodells. Im Durchschnitt sind 1 bis 3 Personenjahre zu veranschlagen, wobei die Aufwendungen für das Pilotprojekt nicht eingerechnet sind. Quick-Wins mit hohem Wirkungsgrad – speziell auf die Projektrisiken – sind selbstverständlich mit weniger als einem Personenjahr zu erzielen.

Zusammenfassung

Die Diskrepanz zwischen der Praxis der Entwicklung und Wartung betrieblicher Informationssysteme in vielen Unternehmen und dem Stand der bekannten „Best Practices“ ist äußerst unbefriedigend. Gesicherte Erfahrungen, erfolgreiche Werkzeuge und Methoden sind nicht hinreichend bekannt, bzw. werden nicht eingesetzt. Nicht selten wird eklatant gegen Grundregeln der Softwareentwicklung verstoßen. Die Folge sind gescheiterte Projekte, verschwendete Projektmittel und verschenkte Chancen für hohe Produktivität und Qualität bei der Entwicklung und Wartung betrieblicher Informationssysteme. All dies geschieht, obwohl die gezielte Unterstützung von Geschäftsprozessen durch Informationstechnologie für fast alle Unternehmen längst von existenzieller Bedeutung ist.

Unser Ansatz EViSE liefert eine pragmatische Anleitung für erfolgreiches Management betrieblicher Informationssysteme. Er beschreibt die Entwicklung von Best-Practice-Vorgehensmodellen,

die den gesamten Life-Cycle von Software einschließlich wesentlicher organisatorischer, technischer und personeller Aspekte abdecken. Er beruht gleichermaßen auf langjähriger praktischer Erfahrung der Autoren und auf aktuellen wissenschaftlichen Erkenntnissen. Ein besonderer Schwerpunkt wird dabei auf die Realisierung bekannter Erfolgsfaktoren für Software-Projekte gelegt.

Mit einer konsequenten, schrittweisen Umsetzung unseres Ansatzes EViSE können kurzfristig wesentliche Risiken in Projekten vermieden und mittelfristig signifikante Verbesserungen von Qualität und Produktivität der Entwicklungs- und Wartungsprozesse erzielt werden.

Danksagung

Wir danken den Herren Gerd Beneken und Professor Dr. Ernst Denert für kritische Anmerkungen und konstruktive Vorschläge sowie dem Bundesministerium für Bildung und Forschung für die Förderung dieser Arbeit im Rahmen des Projekts ViSEK.

Literatur

1. Bauer A., Pizka M.: The contribution of free software to software evolution, IWPSE, Helsinki, 2003
2. Beck K.: Extreme programming explained, embrace change. Addison-Wesley 1999
3. Bitkom-Presseinformation: Informationstechnik und Telekommunikation im Dauerhoch.http://www.bitkom.org/presse/ pr230200.htm (2000)
4. BMBF: Analyse und Evaluation der Softwareentwicklung in Deutschland. http://www.dlr.de/IT/IV/Studien/evasoft_abschlussbericht.pdf (2000)
5. Boehm B.: Software-engineering economics. Prentice Hall 1981
6. DeMarco, T.; Lister, T.: Peopleware, productive projects and teams. 2nd edn. featuring eight all-new chapters. Dorset House 1999
7. Flowers, S.: Software failure: Management failure. John Wiley & Sons 1996
8. Glass, R.: Software runaways. Prentice Hall 1998
9. Hoch, D.J.; Roeding, C.R.; Purkert, G.; Lindner; S.K.: Secrets of software success, management insights from 100 software firms around the word. Harward business School Press 2000
10. Information Week 500 Europa: Der Club der Innovatoren (InformationWeek Studie). InformationWeek Januar 1999
11. Jenner, M.: Software quality management and ISO 9001. Addison-Wesley 1995
12. Kruchten, P.: The rational unified process, an introduction. 2nd edn. Addison-Wesley 2000
13. Moll, K.-R.: Informatik-Management: Aufgabengebiete, Lösungswege, Controlling. Berlin Heidelberg New York Tokio, Spinger 1994
14. Moll, K.-R.: Thesen zu Extreme Programming. Workshop Agile Methodologies, Technical Report ViSEK/014/E. Institut für Informatik, TUM
15. Moll, K.-R.: Langlebige Software-Systeme: Bedeutung und Erfolgsfaktoren. Hot Spots der Software-Entwicklung 2003. Technical Report ViSEK/032/D. Institut für Informatik, TUM
16. Paulk, M.C.; Weber C.V.; Curtis, B.; Chrissis, M.B.: The capability maturity model, guidelines for improving the software process. Addison-Wesley 1995
17. Pizka, M.: Straightening spaghetti-code with refactoring, SERP'04. Las Vegas 2004
18. Prowell, S.: Cleanroom software engineering. Addison-Wesley 1999
19. Reel, J.S.: Critical success factors in software projects. IEEE Software 16(3),1999
20. Royce, W.W.: Managing the development of large software systems, IEEE Wescon, San Francisco 1970
21. Sommerville, I.: Software engineering. 6th edn. Addison-Wesley 2000
22. Software Process Improvement and Capability Extension, SPICE-Konsortium 1998,http://www.sqi.gu.edu.au/spice/
23. Standish Group International, Inc. Collaborating on Project Success. Software Magazine, February/March 2001, Wiesner Publishing.
24. Standish Group International, Inc. CHAOS 1995
25. Standish Group International, Inc. CHAOS: A Recipe for Success 1999
26. Wildemann, H.: Kostenmanagement in der Softwareentwicklung: Leitfaden zur markt- und anforderungsgerechten Produkt- und Prozessgestaltung, München 2000
27. Das V-Modell. Planung und Durchführung von IT-Vorhaben. Entwicklungsstandard für IT-Systeme des Bundes.http://www.v-modell.iabg.de