universität wien

MAGISTERARBEIT

Titel der Magisterarbeit

„Entscheidungskriterien zur Auswahl von Software - Entwicklungsmodellen im Sourcing Kontext“

Verfasser

Gerald Köhl Bakk. rer. soc. oec.

angestrebter akademischer Grad

Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 066 915
Studienrichtung lt. Studienblatt:	Magisterstudium Betriebswirtschaft
Betreuer / Betreuerin:	ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Danksagung

Mein außerordentlicher Dank gilt Frau ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig für die Betreuung meiner Magisterarbeit. Sie hat es mir nach langjähriger Absenz an der Universität Wien ermöglicht, mein Studium mit dieser Arbeit abzuschließen. Ihre unendliche Geduld und ihr offenes Ohr für Fragestellungen aller Art haben meinen Erfolg maßgeblich beeinflusst.

Meiner Liebsten Marion möchte ich auf diesem Wege ganz besonders für ihr nicht enden wollendes Verständnis und ihre Unterstützung danken. Neben meiner beruflichen Tätigkeit und dem Verfassen dieser Abhandlung blieb meist wenig Zeit für die wirklich wichtigen Dinge des Lebens.

Großer Dank gilt meinem bereits verstorbenen Vater, der dies leider nicht mehr miterleben kann und meinen Eltern generell, die mir die schulische und akademische Ausbildung erst ermöglicht hatten.

Bedanken möchte ich mich auch ganz herzlich bei meinen Freunden Stefan und Martin, die mir mit ihren „aufmunternden“ Worten immer wieder den Spiegel vorgehalten haben.

Danken möchte ich ebenfalls Fari und Reinhard für die Unterstützung während meiner Studienendphase, meiner Schwester Doris und meiner Nichte Bella für ein abschließendes Korrekturlesen.

Eidesstattliche Erklärung

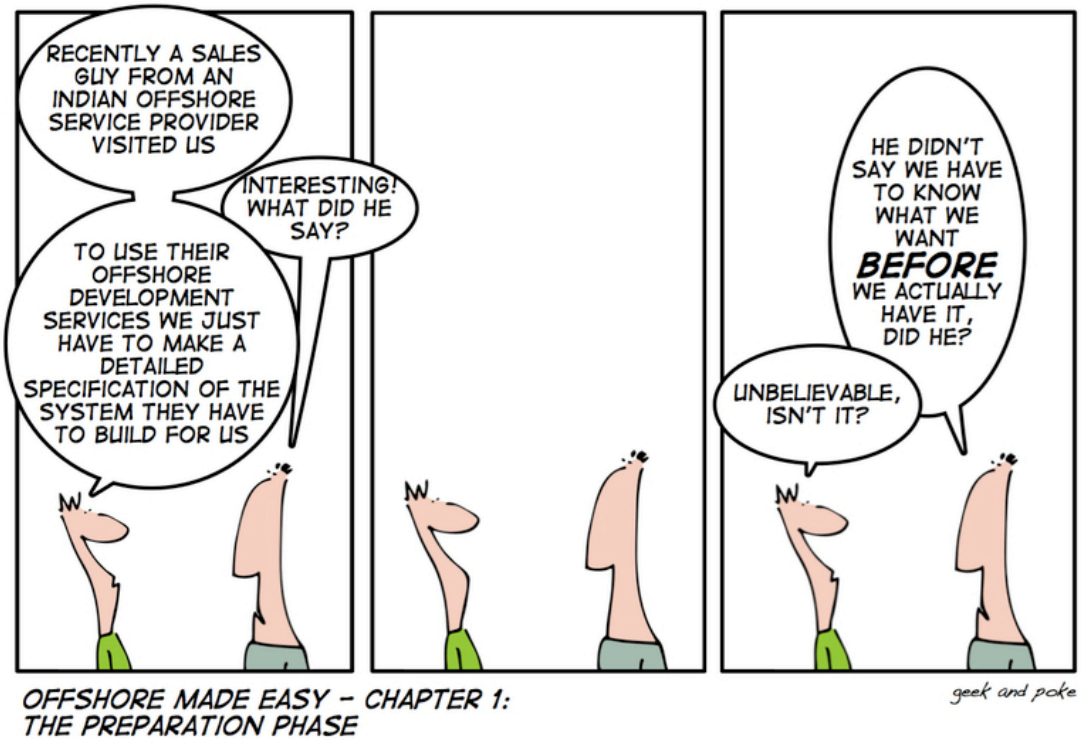
Ich erkläre hiermit, dass ich die vorliegende Magisterarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen benutzt habe und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Diese Arbeit wurde bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt.

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Gerald Köhl

Wien, im September 2012



Quelle: http://geekandpoke.typepad.com/geekandpoke/

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Bereitstellung von IT-Lösungen ist längst keine Angelegenheit für die EDV-Abteilung eines Unternehmens allein. Am Beginn des Computerzeitalters programmierten Spezialisten einfache Algorithmen zur Berechnung von relativ unkomplizierten Fragestellungen nach dem Try und Error Prinzip. Im Laufe der Zeit stieg das Bedürfnis nach Software-Unterstützung rasant an. Jeder kann dies nachvollziehen, indem man kurz inne hält und darüber nachdenkt, in welchen Anwendungsfeldern heutzutage Elektronik und somit Steuerungen in Form von Software enthalten sind. Wird der Blick in Richtung Wirtschaft und Unternehmen gelenkt fällt auf, dass immer mehr Geschäftsprozesse und deren Supportprozesse bereits in hohem Maße IT-unterstützt sind. Der Trend zur Vereinfachung von Tätigkeiten hält ungebremst an, einerseits um Arbeitsschritte zu rationalisieren, andererseits um daraus Kostenvorteile zu generieren.

IT-Software-Entwicklung ist längst erwachsen geworden und steht vor der Herausforderung einer Industrialisierung der IT. Die Idee ist nicht neu, ist es doch seit Jahrzehnten in multinationalen Unternehmen Gang und Gäbe, dass einzelne Geschäftseinheiten anderen im Konzernverbund Dienstleistungen zur Verfügung stellen. Darunter fallen auch IT-Dienstleistungen, die immer stärker arbeitsteilig und vielfach verteilt gestaltet werden. Im Terminus Software Engineering steckt bereits die Erkenntnis, dass ein ingenieursmäßiges Vorgehen aufgrund stärker werdender Komplexität der IT-Lösungen nützlich, wenn nicht sogar unumgänglich ist.

Trotz aller Versuche durch Methoden und Tools die Qualität von Software-Entwicklungen zu erhöhen, krankt es nach wie vor an der erfolgreichen Umsetzung von Vorhaben. Der viel zitierte Chaos-Report der Standish Group attestiert Jahr für Jahr, dass nur rund ein Drittel der Projekte in Time, in Scope und in Budget umgesetzt werden. Etwa ein Fünftel der Projekte werden nachdem sie bereits mehr oder weniger Ressourcen verschlungen haben eingestellt. Der verbleibende Anteil an Projekten wurde zwar abgeschlossen, allerdings unter Überschreitung der Projektlaufzeit, mittels Budgeterhöhung oder durch die Auslieferung weniger Funktionalitäten als ursprünglich geplant (vgl. [Eveleens, J.L. et al. (2008), 2] und [Boczan, O. (2011), 3]).

Eine Komplexitätsreduktion der fortschreitenden Industrialisierung der IT wird durch die Tendenz der Auslagerung von IT-Dienstleistungen nicht begünstigt.

In den letzten zwei Jahrzehnten waren drei Ereignisse maßgeblich beteiligt, damit IT-Entscheider Services am internen oder externen Markt nachfragten. Um die Jahrtausendwende musste eine Vielzahl an Systemen dahingehend überprüft und angepasst werden, damit diese noch lauffähig blieben. Dadurch entstand ein Fachkräftemangel, der die Auslagerung der Entwicklung in andere Länder begünstigt hat. Durch die „dot.com"-Krise und die globale Wirtschaftskrise 2009 sanken die Ausgaben für IT erheblich, die Nachfrage nach Software-Unterstützung war im Gegensatz dazu innerhalb der Unternehmen ungebrochen hoch [Da Rold, C. (2009), 1]. Unternehmen stehen also verstärkt unter Druck die Kosten zu senken. Deshalb wird „IT-Sourcing" mittlerweile ein hoher Stellenwert beigemessen. Dabei geht es grob gesagt darum interne Ressourcen und externe Bereitstellung von Dienstleistungen optimal auszubalancieren. IT-Entscheider erhoffen sich dadurch einerseits auf günstigere Arbeitskräfte aus Billiglohnländern zuzugreifen, andererseits bei Bedarfsspitzen angemessen skalieren zu können.

1.1 Motivation

Durch meine berufliche Laufbahn bin ich seit geraumer Zeit mit Prozessen und Methoden rund um Software-Entwicklung konfrontiert. Die Systemlandschaft in meinem Unternehmen gilt als äußerst komplex und heterogen. Durch die Vielzahl an unterschiedlichen Systemen ist mein Unternehmen Kunde einer nahezu unüberschaubaren Anzahl von IT-Dienstleistern.

Im Jahr 2009 wurde eine Initiative gestartet, die zum Ziel hatte, die Kosten in der IT zu reduzieren und gleichzeitig die Qualität der Ergebnisse zu erhöhen. Dies sollte durch eine Konsolidierung der Lieferanten auf einige wenige strategische Partner erzielt werden. Das übergeordnete Ziel war allerdings die Aussteuerung der gesamten IT zu verbessern. Im Rahmen dieser Initiative durfte ich ein Methodenprojekt leiten, in dem es darum ging herauszufinden, mit welchen Software-Entwicklungsmodellen, Methoden, Best Practices und organisatorischem Setup idealerweise ein Projekt mit Sourcing-Anteil abgewickelt werden sollte. In einer Reihe von Workshops mit Vertretern von möglichen strategischen Partnern am IT-Sourcing Markt wurde ein „Best of Breed" Framework erarbeitet, das es gestatten sollte, eine einheitliche Vorgehensweise unter allen Beteiligten für jegliche Art betrieblicher Softwareentwicklung zu gewährleisten.

Wir haben ein Paket geschnürt und mittels einiger Pilotprojekte erprobt. Bereits während der Workshop-Serie und später in den Pilotprojekten wurde mir klar, dass es keine einheitliche Lösung für alle Probleme gibt. Das war Treiber und Motivation zugleich diese Arbeit zu verfassen.

1.2 Ziele – Erkenntnisinteresse

Meiner Wahrnehmung nach existiert zum Thema IT-Sourcing oder IT-Offshoring in der Fachliteratur Material zur Standortbestimmung und zur strategischen Ausrichtung eines Unternehmens. Im ersten Fall werden Begrifflichkeiten definiert, allerdings werden diese von den Autoren unterschiedlich und teils für mich widersprüchlich beschrieben. Weiters wird statistisches Material zur Aufteilung des IT-Markts bereitgestellt und analysiert bzw. länderspezifische Charakteristika auf hohem Abstraktionsniveau beschrieben. Im zweiten Fall wird die strategische Herangehensweise aus Sicht des Unternehmens betrachtet. Hierbei geht es um eine Strategiedefinition für IT-Sourcing, eine Make-or-Buy Entscheidung, eine Standortauswahl bis hin zu Modellen zur Auswahl eines Sourcing-Partners. Einige dieser Modelle beinhalten den Schritt “Durchführung“, dieser ist aber nur unzureichend spezifiziert. Für den Fall, dass aus strategischem Blickwinkel bereits eine grundsätzliche Entscheidung zur Durchführung eines Offshoring-Vorhabens gefällt wurde, fehlt aus meiner Sicht die methodische Unterstützung für die operative Umsetzung eines Software-Entwicklungsprojekts (vgl. [Gadatsch, A. (2006] oder [Amberg, M. et al. (2006)].

Daraus lassen sich bereits zwei Ziele dieser Arbeit ableiten. Erstes Ziel ist eine strukturierte Definition von Begrifflichkeiten und deren Einordnung im Kontext von IT-Sourcing. Zweites Ziel ist die detaillierte Auseinandersetzung sinnvoll möglicher Konstellationen von Delivery-Modellen und Kollaborationsmodellen auf operativer Ebene. Dabei werden Fragen beantwortet, wie *„An welchen und an wie vielen Orten erbringt der Lieferant seine Leistung?“, „Welche Prozesse und Methoden sollen angewendet werden?“, „Welche Skills sind im Unternehmen verfügbar?“, „Welche Aktivitäten sollen ausgelagert werden?“*. Bei der Diskussion wird besonders darauf Wert gelegt, die Möglichkeiten nach signifikanten Erfolgskriterien zu beurteilen.

Drittes Ziel ist die Beschreibung ausgewählter Software-Entwicklungsmodelle unter Berücksichtigung des IT-Sourcing Kontexts. Die Modelle werden klassifiziert und hin-

sichtlich signifikanter Erfolgsfaktoren beleuchtet. Fragen hierbei lauten etwa *„Sind die Modelle im Offshoring einsetzbar?“, „Wenn ja, unter welcher Voraussetzung?“, „Bietet das Modell hinsichtlich kritischer Erfolgsfaktoren Unterstützung?“, „Für welche Projekte kann das Modell eingesetzt werden?“*.

Als viertes Ziel wird nach dem erfolgten Brückenschlag zwischen IT-Sourcing und Software-Entwicklung ein Leitfaden beziehungsweise eine Entscheidungshilfe ausgearbeitet. Ausgehend von einer strategischen „Buy“-Entscheidung und den fixen Parametern „Unternehmen“ und „Projekt“ wird versucht auf mögliche Sourcing-Konstellationen und Software-Prozesse zu schließen.

1.3 Abgrenzung

In dieser Arbeit beschränke ich mich auf die Fremdvergabe von Software Development Aktivitäten. Die Bereitstellung von Infrastruktur, Applikationen und gesamter Geschäftsprozesse wird in diesem Rahmen nicht betrachtet, da es dabei keinen Zusammenhang zu Softwareentwicklungsprozessen gibt.

Eine strategische Sichtweise wird in den meisten Fällen nicht eingenommen, die Konzentration liegt auf der operativen Ebene. Beispielsweise kann die Fragestellung welche Aktivitäten vom Lieferanten durchgeführt werden sollen auch einen strategischen Aspekt aufweisen. Hier wird grundsätzlich die Frage gestellt, ob eine Aktivität aufgrund der vorhandenen Fähigkeiten eher beim Kunden oder Lieferanten abgewickelt wird. Delivery-Modelle bergen unterschiedliche finanzielle Einsparungspotentiale in sich. Zur Verdeutlichung der Kosteneinsparungen werden fiktive Kostensätze für die Entwicklungsleistung angesetzt. Für Kollaborationsmodelle gilt selbiges. Einsparungspotentiale sind in unterschiedlicher Ausprägung durch die Art der Entwicklungsleistung bestimmt. Zur Komplexitätsreduktion wird der Faktor Kosten nicht näher definiert.

Die Ausführungen dieser Arbeit beziehen sich ausnahmslos auf betriebliche Anwendungen. Das sind beispielsweise CRM-Systeme, ERP-Systeme, e-commerce Applikationen, Business Intelligence Systeme, Kundenverrechnungssysteme, webbasierte Lösungen und ähnliches mehr. Zwecks Komplexitätsreduktion nicht weiter betrachtet werden eingebettete Systeme (embedded systems), Entertainment-Software, Simulations-Software, Datensammlungssysteme, Steuerungs-Software und ähnliches. Diese Soft-

ware-Typen erfordern teils eigene Entwicklungsprozesse, die den Rahmen der Arbeit sprengen würde.

Im Rahmen der Diskussion von Software-Entwicklungsmodellen wird jeweils ein sinnvoller Vertreter des sequentiellen, iterativ /inkrementellen und agilen Paradigmas beleuchtet. Auch hier würde eine umfassende Diskussion den entsprechenden Raum benötigen.

1.4 Struktur der Arbeit

Der erste Teil der Arbeit beschäftigt sich mit dem Gebiet von IT-Sourcing bzw. IT-Offshoring (Kapitel 2-4). Nach einer begrifflichen Einordnung folgt eine Kategorisierung der verschiedenen Offshoring Konzepte basierend auf einem eingehenden Literaturstudium aus Fachbüchern und wissenschaftlichen Beiträgen zu Kongressen und Tagungen. In weiterer Folge werden diese Konzepte nach deren Relevanz für die vorliegende Arbeit beurteilt und eingegrenzt. Wie wir am Ende von Kapitel 2 sehen werden, wird die gängige Klassifikation um eine Kategorie erweitert.

In Kapitel 3 werden grundlegende Organisationsformen im Offshoring-Development identifiziert und beleuchtet. In der Literatur findet sich diese Art der Darstellung lediglich in Randbemerkungen wieder. Umso wichtiger war mir eine detaillierte Diskussion der unterschiedlichen Ausprägungen im operativen Kontext. Um eine objektive Darstellung zu gewährleisten, wurde in diversen Wissensbasen nach Studien gesucht, mit deren Hilfe eine Bewertung erst möglich wurde. Diese kritischen Erfolgsfaktoren finden sich am Beginn des Kapitels 3, folgend eine Beschreibung der Delivery-Modelle mit anschließender Diskussion derselben.

Dem gleichen Aufbau folgt Kapitel 4, in dem Konstellationen bezogen auf Aktivitäten bzw. Software Engineering Disziplinen betrachtet werden.

Der zweite Teil der Arbeit (Kapitel 5-7) beschäftigt sich einleitend mit allgemeinen Erläuterungen zum Thema Software Engineering (Kapitel 5). Ziel davon ist eine allgemeine Begriffsbestimmung zu erreichen. Kapitel 6 bietet einen Überblick zu Software-Entwicklungsmodellen mit Fokus auf einer chronologischen Einordnung und grundlegenden Differenzierungsarten. In Kapitel 7 werden ausgewählte Software-Entwicklungsmodelle betrachtet, die in der Praxis aus meiner beruflichen Erfahrung

gerne angewendet werden. Dabei wurde darauf Wert gelegt, dass die meisten in Kapitel 6 angeführten Differenzierungsarten vertreten sind. Jedes dieser Modelle wird wenn möglich basierend auf der ursprünglichen Literatur beschrieben. Am Ende jedes Unterkapitels findet sich eine Diskussion der Modelle bezogen auf die Erkenntnisse aus Kapitel 3 und Kapitel 4, um deren Anwendbarkeit im Offshoring- Kontext beurteilen zu können.

Im dritten Teil der Arbeit (Kapitel 8) wird ein Modell erarbeitet, um es einem Unternehmen zu ermöglichen eine optimale Konstellation aus Delivery-Modell, Kollaborationsmodell und Software-Entwicklungsmodell zu bestimmen. Dabei wird besonderes Augenmerk auf den Projektkontext und Unternehmenskontext gelegt.

2 IT-Sourcing

2.1 Begriffsdefinition

Unter Sourcing wird aus meiner Sicht der allgemeine Beschaffungsprozess von Produkten oder Dienstleistungen verstanden. IT-Sourcing schließt sich der Definition an und umfasst die Bereitstellung von IT-Dienstleistungen. Eine trennscharfe Definition des Begriffs Sourcing sucht man in der Literatur vergeblich. Sourcing als Synonym für Purchasing, als Abteilung, Verfahrensmodell einerseits und strategische Ausrichtung andererseits beschreibt Larry Paguette [Paguette, L. (2004), 5ff.]. Dabei betrachtet er ausschließlich den Zukauf von Gütern am freien Markt. Der Begriff zielt nicht in erster Linie darauf ab, ob die Leistung im Unternehmen selbst hergestellt oder von einem Lieferanten erbracht wird. IT-Sourcing wird oftmals fälschlicherweise als Synonym für IT-Outsourcing verwendet. IT-Outsourcing ist aus meiner Erfahrung ein in Unternehmen negativ belegter Begriff, da dieser zwangsweise durch die Ausgliederung von zuerst im Unternehmen selbst bereitgestellten IT-Services mit Mitarbeiterabbau assoziiert wird. IT-Outsourcing impliziert, dass eine Dienstleistung bereits unternehmensintern bereitgestellt wurde, während IT-Sourcing auch eine Veränderung der Bereitstellung von bereits ausgelagerten Services oder die Beschaffung neuer IT-Services beinhaltet. Unternehmensinternes IT-Sourcing kann für neuartige Services durchaus mittels Rekrutierung neuer oder der Entwicklung von Fertigkeiten bestehender Mitarbeiter erwirkt werden.

Nichtsdestotrotz sehen sich IT-Entscheider aus vielfältigen Gründen veranlasst, Leistungen nicht zur Gänze selbst zu produzieren, IT-Services gänzlich oder zu einem gewissen Grad vom Markt zu beziehen.

IT-Sourcing umfasst immer eine Kunde-Lieferant-Beziehung, die im Wesentlichen auf einer vertraglichen Grundlage beruht. Der Kunde fragt eine Dienstleistung am Markt nach, der Lieferant übernimmt einen Auftrag und erbringt ein vorab definiertes Ergebnis, in unserem Fall eine Dienstleistung in Form einer Software-Entwicklungs-Leistung. In diesem Sinne werden in weiterer Folge Begriffe synonym verwendet, die in einem anderen Kontext unterschiedliche Bedeutungen haben können. Demnach ist der Begriff „Kunde“ einem Unternehmen gleichzusetzen. Analog dazu wird der Begriff Auftraggeber synonym verwendet, obwohl im Fachgebiet des Projektmanagements ein Auftrag-

geber eine wohl definierte, personifizierte Rolle darstellt. Zur besseren Lesbarkeit werden des Öfteren Begrifflichkeiten wie Offshoring-Partner, Sourcing-Partner, Vendor, Dienstleister, Anbieter, Outsourcing-Partner und externer Dienstleister benutzt. Gemeint ist immer der Lieferant, es sei denn, es wird explizit darauf hingewiesen.

2.2 Kategorisierung von IT-Sourcing

Holger von Jouanne-Diedrich stellte die IT-Sourcing Map 2004 vor, die mittlerweile in der dritten Version vorliegt [von Jouanne-Diedrich, Holger (2009)]. Die IT-Sourcing Map ist die mit Abstand detaillierteste in der Literatur zu findende Klassifizierung, wenngleich mir einige Aspekte für die Diskussion dieses Themas fehlen, beziehungsweise nicht in der Tiefe erklärt werden. Begrifflichkeiten wie Hosting Szenarien, Shared Service Center und neuerdings Cloud Computing und Software as a Service geistern durch die Fachliteratur.

[Amberg, M. et al. (2006), 7ff] kategorisieren IT-Offshoring im Gegensatz zu von Jouanne-Diedrich weniger komplex in einem dreidimensionalen Modell nach Organisationsform (Tochterunternehmen, Joint Venture, Fremdunternehmen), Gestaltungsformen (totales und partielles Offshoring) und Leistungsformen (Infrastructure Offshoring, Software Development Offshoring, Business Process Offshoring).

Nichtsdestotrotz halte ich die IT-Sourcing Map als den allgemeinen Standard zur Klassifizierung von IT-Sourcing. Es wird versucht, die Begrifflichkeiten in die Map einzuordnen.

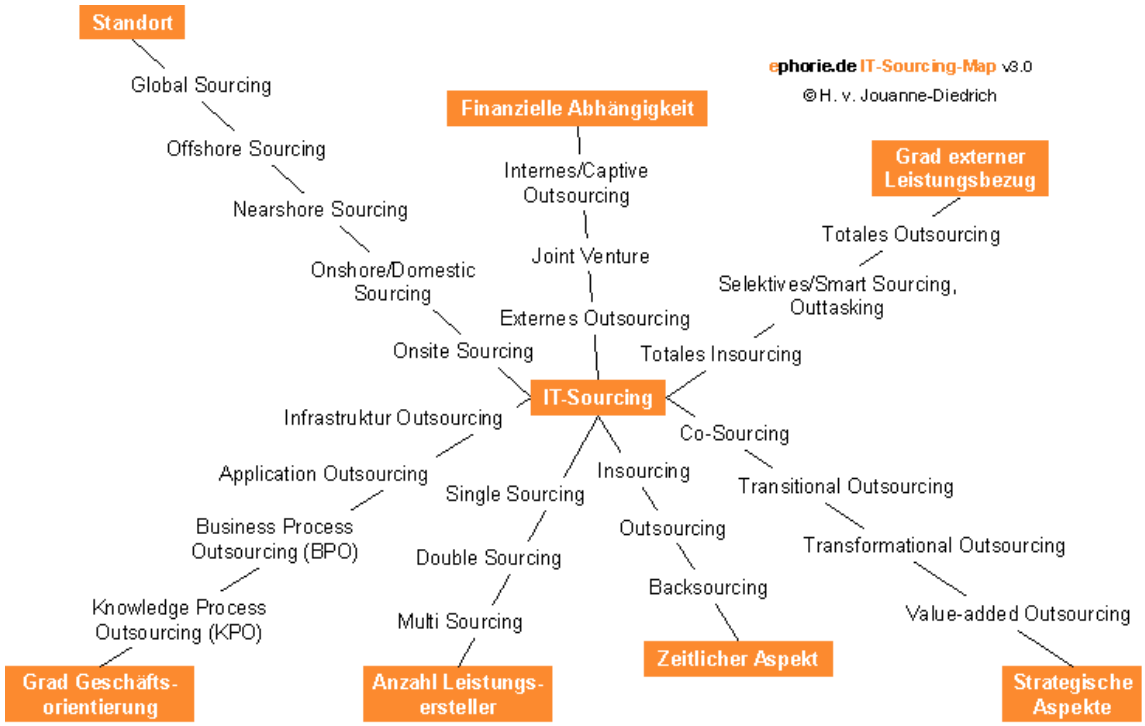


Abbildung 1: IT-Sourcing Map [von Jouanne-Diedrich, H. (2009)]

2.2.1 Unterscheidung nach finanzieller Abhängigkeit

Externes Outsourcing zielt darauf ab, Leistungen außerhalb eines Unternehmens oder Konzerns zu beziehen, während internes Outsourcing im Konzernverbund zugekaufte Leistungen verkörpern. Ausschlaggebend beim internen Outsourcing ist, dass der Lieferant unter Marktbedingungen Leistungen liefert. Davon zu unterscheiden sind Hosting Szenarien oder Shared Service Center, die ausschließlich den eigenen Konzern bedienen.

Beim Shared Service Center werden Services von einem Unternehmen(-steil) für mehrere andere zur Verfügung gestellt. Zum Beispiel übernimmt ein Unternehmen des Konzerns die Entwicklung des Blueprints vom ERP-System und stellt diese jedem anderen Unternehmen im Verbund zur Verfügung. Hosting kann entweder externen oder internen Bezug aufweisen und kann auch zum Kreis des Infrastructure Outsourcings gezählt werden. Wird im Konzernverbund Hosting betrieben, werden physische IT-Ressourcen für die Unternehmen bereitgestellt. Das heißt, die Blueprints des ERP-Systems werden auf verschiedenen Instanzen von einem Unternehmen betrieben. (vgl. [Weber, M. 2008, 15 und 33]).

2.2.2 Unterscheidung nach dem Grad des externen Leistungsbezugs

Wie der Ausdruck vermuten lässt, geht es dabei um den Umfang oder Grad des Leistungsbezugs vom Offshoring-Partner.

Dabei ist 100% ausgelagerte Leistungserstellung mit totalem Outsourcing gleichzusetzen und bei geringerem Volumen mit partiellem Outsourcing. [Gadatsch, A. (2006), 17]. Widersprüchlich dazu Autoren wie [Amberg, M. et al. (2006), 17] und [Schwarze, L. et al. (2005), 13], die mindestens 80% Auslagerung von IT-Funktionen für totales Outsourcing ansetzen.

Synonym werden hierbei Begriffe wie Right Sourcing, Smart Sourcing, Outtasking und partielles Sourcing verwendet (vgl. [Amberg, M. et al. (2006), 18]).

Beim partiellen Outsourcing werden abgrenzbare Teilleistungen an den Offshoring-Partner abgegeben. Der Vorteil bei dieser Variante ist in der geringeren Abhängigkeit zum Lieferanten zu sehen, da weniger Wissen transferiert werden muss. Darüber hinaus spielt die Organisationsform eine Rolle oder wie von von Jouanne-Diedrich geprägt, die finanzielle Abhängigkeit. Um abseits von der vertraglichen Ausgestaltung von Vereinbarungen nicht gänzlich die Kontrolle über den Sourcing-Partner zu verlieren, können Formen wie Joint Ventures oder bei 100%iger Auslagerung ein Tochterunternehmen die richtige Wahl sein (siehe internes Outsourcing).

Totales Outsourcing im Kontext von „Cloud Computing“ ist eine Form der bedarfsgerechten und flexiblen Nutzung von IT-Leistungen. Diese werden in Echtzeit als Service über das Internet bereitgestellt und nach Nutzung abgerechnet [Weber, M. (2009), 14].

2.2.3 Unterscheidung nach der Anzahl der Leistungsersteller

Multi-Sourcing bezieht sich auf eine Variante, bei der sich mehrere Lieferanten die Lieferung teilen. Im Gegensatz dazu ist Single-Sourcing eine 1:1 Kunde-Lieferanten Beziehung. Oftmals tritt ein Unternehmen als Generalunternehmer auf und beschäftigt hinter sich mehrere weitere Lieferanten. Dies hat einen vereinfachenden Effekt für das beauftragende Unternehmen, birgt jedoch mitunter Risiken, z.B. durch mehrere (Human-) Schnittstellen, die der Generalunternehmer aussteuern muss.

Eine weitere Variante stellt das Double-Sourcing dar, bei der genau zwei Lieferanten an der Leistungserbringung beteiligt sind.

Beispielsweise setzt ein Vendor zur Implementierung und Einführung einer Softwarelösung einen „Integrator“ ein. Vertragspartner ist zwar das Softwarehaus selbst, die (Beratungs-, Implementierungs- und Konfigurations-) Leistung erbringt jedoch ein anderes Unternehmen.

2.2.4 Unterscheidung nach dem zeitlichen Aspekt

Unter Insourcing versteht man die Erbringung von Leistungen aus dem Konzernverbund, die zuvor von einem externen Dienstleister bereitgestellt wurden. Ein vorangestelltes Outsourcing ist nicht zwingend notwendig.

Wird eine Leistung, die zuerst intern erstellt und anschließend außerhalb eines Unternehmensverbundes bezogen wurde in weiterer Folge wieder intern bereitgestellt, spricht man vom Backsourcing.

2.2.5 Unterscheidung nach dem Standort

Offshoring im weiteren Sinn meint die Leistungserstellung im Ausland. Offshoring im engeren Sinn meint die Leistungserbringung in Regionen fern dem eigenen Land, in dem es starke kulturelle Unterschiede zum Heimatland gibt und der Anbieter in einer Zeitzone angesiedelt ist, die sich erheblich vom Ursprungsland unterscheidet. Onshoring (domestic shoring) bezeichnet die Auslagerung im näheren Umfeld, im gleichen Land, in dem sich das Unternehmen befindet. Onsite produziert ein Lieferant dann, wenn dieser direkt am Firmensitz seine Leistungen erbringt. Nearshoring beschreibt eine Leistungserstellung im näheren Ausland, durch geringe kulturelle Unterschiede gekennzeichnet. Der Lieferant liegt zudem in der gleichen Zeitzone oder zumindest mit geringem zeitlichem Unterschied. Daher ist Nearshoring immer auch zwangsweise Offshoring im weiteren Sinn. Werden in einem Unternehmen sämtliche Typen und Kombinationen davon genutzt, spricht man obendrein von Global Sourcing.

2.2.6 Unterscheidung nach dem strategischen Aspekt

Unter Value-added Outsourcing versteht man zum Beispiel Joint-Venture Organisationsformen, bei denen beide Vertragsparteien bestimmte Kompetenzen einbringen, um Leistungen am Markt anzubieten. Dabei handelt es sich um ein revenue-sharing Modell

mit geteilten Chancen und Risiken. Ähnlich verhält es sich beim Co-Sourcing, hier wird der Anbieter transaktions- oder umsatzbasiert abgerechnet.

Beim Transitional Outsourcing werden intern Ressourcen freigespielt, um veraltete Legacy-Systeme durch neue Technologien zu ersetzen. Der Outsourcing-Partner fungiert hierbei als Systemerhalter der bestehenden Technologie.

Transformational Outsourcing verbindet die Auslagerung von IT-Systemen mit der Auslagerung und Neugestaltung von Geschäftsprozessen. Entscheidend hierbei ist, dass der Lieferant gleichzeitig als Unternehmensberater auftritt.

Laut von Jouanne-Diedrich ist die Sinnhaftigkeit dieser Vorgehensweise in der gesamten Industrie umstritten.

2.2.7 Unterscheidung nach dem Grad der Geschäftsorientierung

Infrastruktur-Outsourcing

Diese Form ist die älteste bekannte Art des Outsourcings und beinhaltet zum Beispiel den externen Betrieb von Rechenzentren, Netzwerkdienstleistungen oder die Bereitstellung von PCs im Unternehmen. Beim Infrastructure as a Service fallen Investitionen weg und die vereinbarten Services werden nutzungsbasiert verrechnet (Bestandteil von Cloud Computing).

Application Outsourcing

Das Application Outsourcing (Applikations-Outsourcing, Business Application Outsourcing) bezeichnet den Betrieb von kompletten Applikationen, meist Standardsoftware, wie ERP oder CRM Systeme, durch einen externen Anbieter. Application Outsourcing basiert auf einem Lizenzmodell. Synonyme hierfür sind Net-Sourcing oder E-Sourcing.

Einen Spezialfall stellt Software as a Service (SaaS) dar, bei dem eine nutzungsbasierte Abrechnung erfolgt, eine Beschaffung von Lizenzen entfällt. (vgl. [Weber, M. 2008, 34]. SaaS ist als eine Ausprägung von Cloud Computing anzusehen.

Business Process Outsourcing

Business Process Outsourcing bezeichnet das Auslagern kompletter nicht business-kritischer Geschäftsprozesse samt Infrastruktur und Applikationen, wie zum Beispiel Human Resources, Personalverrechnung, oder ähnliches.

Knowledge Process Outsourcing

Knowledge Process Outsourcing ist eine Mutation von Business Process Outsourcing und meint die Auslagerung wissensintensiver Geschäftsprozesse, wie Customer Loyalty Management oder Business Intelligence im Allgemeinen. Kritisch anzumerken ist hierbei die Tatsache, dass unternehmenskritische Daten und Informationen an Dritte weitergegeben werden.

Software Development Sourcing

Die für diese Arbeit enorm wichtige Ausprägung des Software Development Offshorings fehlt hingegen in der taxativen Aufzählung der IT-Sourcing Map.

Möglicherweise ist die Software-Entwicklung unter dem Punkt „Grad des externen Leistungsbezugs“ oder im „Grad der Geschäftsorientierung“ zu finden. Ich möchte allerdings explizit darauf verweisen.

Im ersten Fall würde es darum gehen, ob ein komplettes Software-Entwicklungsprojekt abgewickelt werden soll oder nur Teilbereiche davon. Im zweiten Fall kann das Application Outsourcing als Klammer gedient haben, dies geht allerdings nicht hervor. Vielmehr ist die Rede von mehr oder weniger anpassbaren Standardsoftware-Produkten, welche zur Gänze von einem Lieferanten (ein Application Service Provider), vorzugsweise über Webzugriff bezogen werden (vgl. [von Jouanne-Diedrich, H. (2008), 27]).

[Amberg, M. et al. (2006), 10ff] subsummieren ein Application Service Providing durch die für viele Kunden zugängliche Standardlösungsvariante unter Infrastructure Offshoring. Sie weisen als eigene Leistungsform „Software Development Offshoring“ aus. Hierbei geht es um die totale oder partielle Fremdvergabe von Software-Entwicklungsprojekten. Kurz skizziert handelt es sich hierbei um Softwareneu- und weiterentwicklungen von Individualsoftware (z.B. in Form von Release-Projekten), die Implementierung und Konfiguration von Standardsoftwareprodukten, eine Migration von bestehenden Systemen auf Plattformen neueren Standards oder auch Business Intelligence Projekte.

2.2.8 Eingrenzung nach Relevanz

Die oben angeführte Auflistung und Skizzierung von Sourcing-Kategorien erhebt keinen Anspruch auf Vollständigkeit. Ich möchte auch darauf hinweisen, dass die Begrifflichkeiten in Literatur und wissenschaftlichen Schriften nicht einheitlich, manchmal widersprüchlich verwendet werden.

Für diese Arbeit sind nicht alle Kategorien wesentlich, folglich müssen sie teils kategorisch ausgeschlossen, teils hinsichtlich Relevanz ausgesucht werden. Manche Aspekte können durchaus für Spezialfälle Bedeutung gewinnen. Diese werden jedoch im Sinne einer Komplexitätsreduktion bewusst vernachlässigt.

In dieser Arbeit versuche ich Entscheidungskriterien für die Auswahl von geeigneten Software-Entwicklungsmodellen und Kollaborationsmechanismen zu finden. In der Kategorie nach dem Grad der Geschäftsorientierung verbleibt ausschließlich das Klassifizierungsmerkmal Software Development Offshoring. Diese Arbeit betrachtet in weiterer Folge nicht die Vergabe von Infrastruktur, Application Outsourcing sowie Business Process- und Knowledge Process Outsourcing.

Die Unterscheidung nach der Anzahl der Leistungsersteller wird untergeordnet mit betrachtet. Mehrere Leistungserbringer, die direkt vom Auftraggeber gesteuert werden müssen, erhöhen zwar die Komplexität, dies hat aus derzeitiger Sicht allein keine Auswirkung auf die Auswahl des Software- Entwicklungsmodells. Es ist als unwahrscheinlich anzusehen, dass in einem Projekt mit mehreren Lieferanten unterschiedliche Modelle zum Einsatz kommen. Ein zielgerichtetes Projektmanagement wäre unter diesen Umständen kaum zu bewerkstelligen.

Der Auftraggeber wird darüber hinaus darauf bedacht sein, ein dem Unternehmen wohlbekanntes und erprobtes Modell sicherlich mit einigen Anpassungen in Projekten auch mit mehreren Lieferanten zu Grunde zu legen.

Dieser Beitrag betrachtet nicht die Vergangenheit, sondern den Status Quo. Hier und jetzt versucht der Entscheider Aussagen über die nähere Zukunft seines Offshoring-Engagements zu treffen. Der zeitliche Aspekt ist bei diesem teils strategischen, teils taktischen Auswahlverfahren nachrangig und kann vernachlässigt werden. Es geht vielmehr darum, ob das Unternehmen künftig die Leistung selbst erstellt oder von einem Lieferanten bezieht und wenn ja, mit welchem Kooperationsmodell.

Die Auswahl des passenden Standorts wird aus diesem Blickwinkel eine ernsthafte Rolle spielen. Tendenziell werden für weiter entfernt gelegene Standorte andere Kontrollmechanismen gelten müssen als für Beziehungen vor Ort. Kommunikationsbarrieren sprachlicher Natur und durch die räumliche Distanz erschweren die Beziehung zu einem Offshore-Anbieter aus meiner Sicht erheblich. Stehen vor Ort oder besser Onsite, Face-to-Face Gespräche an der Tagesordnung, ist es im Offshore Modus unumgänglich, geeignete Instrumente zu finden, die den regelmäßigen Informationsaustausch reibungslos ermöglichen.

Die Form der finanziellen Abhängigkeit wählt der Auftraggeber nach seiner strategischen Ausrichtung. Je mehr Einfluss auf den Auftragnehmer erwünscht ist, umso mehr kommen die Übernahme von Anbietern oder der Aufbau von Tochterunternehmen in Betracht. Im Gegensatz dazu steht die Fremdvergabe an einen externen Dienstleister, der rein vertragsrechtlich an den Kunden gebunden ist. Im Fokus dieser Arbeit steht hauptsächlich die Fremdvergabe von Software-Entwicklungsaktivitäten, daher spielen die weiteren Ausprägungen eine eher untergeordnete Rolle. Ich möchte jedoch nicht ausschließen, dass die interne Softwareentwicklung für gewisse Projektarten, wie etwa bei Prestigeprojekten, die nötigen Ressourcen vorausgesetzt, ein gangbarer Weg ist. Wenn dies der Fall sein sollte, wird das speziell in weiterer Folge erwähnt.

Wie bereits dargestellt, steht beim Grad des externen Leistungsbezugs der Anteil der ausgelagerten Aktivitäten im Mittelpunkt. Gerade bei Software-Entwicklungsprojekten gilt es die nötige Balance zu finden. Natürlich besteht die Möglichkeit alle Disziplinen von einem erfahrenen, mit exzellenten Branchenkenntnissen ausgestatteten Lieferanten abdecken zu lassen. Im ersten Augenblick scheint eine Auslagerung von Teilbereichen sinnvoll. Für die Gestaltung von Geschäftsprozessen muss der Lieferant darüber hinaus über die unternehmensinternen Prozesse bestens Bescheid wissen. Eine Komplexitätsstufe, die entweder nach längerer Zusammenarbeit auf ein Minimum reduziert werden kann oder zu Beginn einer Geschäftsbeziehung mit enormem finanziellem Aufwand quittiert wird.

Eine Unterscheidung nach dem strategischen Aspekt ist im Kontext dieser Arbeit eventuell in der Ausprägung Value-added Outsourcing denkbar. Dies ist ein revenue-sharing basierter Ansatz, bei dem von einem Partner Leistung zugekauft wird. Im Rahmen der Neuentwicklung eines Kundenportals wäre die Integration einer auf Kundenwunsch adaptierten Applikation („Branding“) möglich, über die gewisse Transaktionen laufen.

Beispiel: Eine Download-Plattform für Musik-Files wird in ein Kundenportal integriert, den Content stellt der Anbieter zur Verfügung. Die Erlöse werden transaktionsbasiert zwischen den Vertragspartnern abgerechnet.

Die in Frage kommenden Varianten des Sourcing werden graphisch in der nachfolgenden, adaptierten IT-Sourcing Map verdeutlicht.

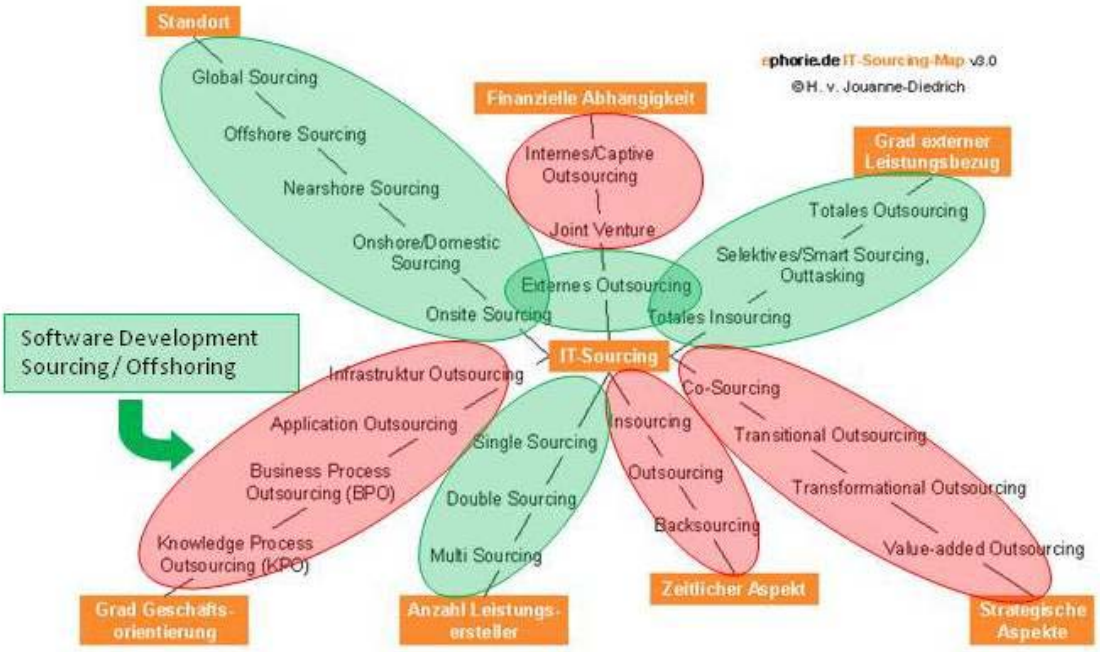


Abbildung 2: modifizierte IT-Sourcing Map

3 Delivery-Modelle

Der folgende Abschnitt hat zum Ziel die grundlegenden Organisationsformen im Outsourcing / Offshoring Umfeld heraus zu arbeiten. Die Gliederung orientiert sich an der oben beschriebenen Klassifizierung und Unterscheidung nach dem Standort.

Die zu Grunde liegende Frage lautet: „An welchen und an wie vielen Orten erbringt der Lieferant seine Leistung?“

Die Basis bilden die reinen Organisationsformen Onsite, Onshore und Offshore im weiteren Sinne (vgl. [Amberg, M. et al. (2006), 27]). Darüber hinaus werden gängige Kombinationen aus der Praxis angeführt und diskutiert.

Um eine objektive Betrachtung zu gewährleisten werden ausgewählte Kriterien, die in der Literatur genannt werden, herangezogen und mit den Erkenntnissen aus Offshoring-Projekten aus dem beruflichen Umfeld verdichtet. Basis dafür sind die in diversen Studien genannten kritischen Erfolgsfaktoren, aber auch Zielsetzungen, Chancen und Risiken, immer mit Bezug auf die Fragestellung dieser Arbeit. Es werden daher nicht zwingend die meistgenannten oder die wichtigsten Faktoren aus der Literatur betrachtet.

3.1 Kritische Erfolgsfaktoren

Für eine Entscheidung die Software-Entwicklung ins Ausland zu verlagern, steht das Kostenmotiv im Vordergrund. Dies unabhängig davon, welches Offshoring-Modell verwendet wird. 91.9% der Befragten nennen in einer Studie von Regina Moczadlo geringere Lohn-und Lohnnebenkosten als wichtig bzw. sehr wichtig [Moczadlo, R. (2005), 2] (vgl. hierzu auch [Deutsche Bank Research (2005), 12]). Demzufolge werden in diesem Rahmen fiktive Stundensätze als Bezugsrahmen herangezogen. Abbildung 3 zeigt mögliche Konstellationen im Offshoring. Die Aufteilung gibt an, zu welchen Teilen die Leistung in den verschiedenen Regionen erstellt wird. Als Basis für die Berechnung der Blended Rates wurde auf [Deutsche Bank Research (2005), 12] und [Boehm, C. (2004), 6] bezogen.

Delivery Modelle	Onsite	Onshore	Nearshore	Offshore	Aufteilung in %	Blended Rates
Offshore				X	100	20
Nearshore/ Offshore			X	X	40:60	36
Global Delivery	X		X	X	20:20:60	40
Global Delivery	X	X	X	X	10:10:20:60	42
Global Delivery		X	X	X	20:20:60	44
Onsite / Offshore	X			X	40:60	44
Global Delivery	X	X		X	20:20:60	48
Onshore / Offshore		X		X	40:60	52
Nearshore			X		100	60
Onsite / Nearshore	X		X		40:60	68
Global Delivery	X	X	X		20:20:60	72
Onshore / Nearshore		X	X		40:60	76
Onsite	X				100	80
Onsite / Onshore	X	X			20:80	96
Onshore		X			100	100

Abbildung 3: Blended Rates

Der kommerzielle Erfolg oder Misserfolg eines Projekts hängt allerdings neben den Lohnkosten auch maßgeblich von anderen Faktoren ab. Insbesondere zu Projektbeginn, der sogenannten Ramp-Up Phase, sind hohe Anlaufkosten zu erwarten, projektbegleitend erhöhen sich die Koordinationskosten für die Offshore-Teams um bis zu 40% (vgl. [Amberg, M. et al. (2006), 62]). Für diese Arbeit ist die Betrachtung der Lohnkosten ausreichend, da angenommen werden darf, dass diese für Offshore-Projekte typischen Kostenblöcke in jedem Offshoring-Vorhaben zu einem gewissen Grad anfallen. Tendenziell werden aus diesem Blickwinkel Projekte aus finanzieller Sicht umso erfolgreicher sein, je länger eine Geschäftsbeziehung andauert.

Im Umkehrschluss wird es, wenn überhaupt, nicht viel Ersparnis bringen, für ein Kleinprojekt die Maschinerie mit einem Offshoring-Partner hoch zu fahren (vgl. [Böhm, C. (2004), 6]).

In einer exploratorischen Studie zu kritischen Erfolgsfaktoren von Michael Amberg wurden fünfzehn zuvor erstellte Studien durchleuchtet und mit Software-Entwicklungsprojekten im Offshoring-Kontext in Verbindung gesetzt. Die identifizierten Erfolgsfaktoren wurden nach internen und externen Eignungs- sowie Managementfaktoren klassifiziert. Für die Bewertung von Delivery-Modellen ist meiner Ansicht nach die externe Sichtweise grundsätzlich maßgeblich und es wurden aus den sechzehn Erfolgsfaktoren dieser Kategorie sechs isoliert (vgl. [Amberg, M. et al. (2005), 29ff.].

Externe Eignungsfaktoren sind nach Amberg „Erfolgsfaktoren hinsichtlich der Auswahl eines Offshore-Entwicklungspartners“ und Externe Managementfaktoren sind „Erfolgs-

faktoren hinsichtlich der gemeinsamen operativen Durchführung eines Offshore-Entwicklungsprojekts“ (vgl. [Amberg, M. et al. (2005), 31]).

3.2 Die isolierten Erfolgskriterien im Einzelnen:

3.2.1 Sicherstellen eines kontinuierlichen Kommunikationsflusses

Amberg verweist in seiner Studie darauf, Projektaufgaben nicht auf zu viele Standorte zu verteilen, um eine ordentliche Kommunikation zu gewährleisten (vgl. [Amberg, M. et al. (2005), 40]). Demzufolge sind Single-Source Modelle denen des Global Sourcings vorzuziehen, da sich jene negativ auf den Projekterfolg auswirken können. Informationen müssen über eine erhebliche Anzahl an Schnittstellen transportiert werden, das erhöht die Fehlerquote durch Informationsverlust und Missinterpretation. Darüber hinaus sollte die Wahl eines Kommunikationsmediums immer an die Aufgabenstellung angepasst werden. Ist die Komplexität einer Aufgabe gering oder ist kein unmittelbares Feedback nötig, reicht Telefon oder Email aus. Ist die Komplexität hoch, wie z.B. bei der technischen Analyse von Problemstellungen, sind etwa Videokonferenzen oder Face-to-Face Gespräche zweckmäßig (vgl. [Amberg, M. et al. (2006), 173]). Es macht Sinn eigene Projekt-Websites oder zentrale Wikis und Collaboration-Werkzeuge für alle Stakeholder einzurichten. Eine gemeinsame Sicht auf Informationen ist ein Baustein auf dem Weg zum Erfolg.

[Nicklisch, G. et al. (2009), 34] schlägt in die gleiche Kerbe, in dem er Risikofaktoren und Möglichkeiten zur Minimierung dieser Quellen listet. Demnach erhöht sich die Effektivität von verteilten Teams, wenn die Kommunikationsmedien normiert werden und es erhöht die Qualität von Analysergebnissen, wenn verstärkt Face-to-Face Meetings (auch Videokonferenzen) durchgeführt werden.

3.2.2 Persönliche Treffen zwischen den Projektpartnern

Oben bereits erwähnt und ausdrücklich als Erfolgsfaktor in [Amberg, M. et al. (2005), 40] nochmals genannt, erweist sich der direkte Informationsaustausch zwischen Auftraggeber und Lieferant speziell zum Projektstart und weiter im Projektverlauf insbesondere in kritischen Situationen als hilfreich. Besonders bei komplexen Aufgaben mit hohem Interaktionsbedarf, wenn es also verstärkt um das beiderseitige Verständnis von

Sachverhalten geht, kann auf dieses Medium schwer verzichtet werden. Als Nebeneffekt wird das Teamgefüge positiv beeinflusst und das partnerschaftliche Zusammengehörigkeitsgefühl gestärkt.

3.2.3 Gute Deutsch- bzw. Englischkenntnisse auf Anbieterseite

In vielen europäischen Unternehmen ist Englisch als maßgebliche Sprache in der Projektarbeit nicht mehr wegzudenken, obgleich das nicht für die Gesamtheit unterstellt werden kann. Gerade in größeren und auf jeden Fall in global agierenden Unternehmen dürfte Englisch mittlerweile zum Standard gehören. Nichtsdestotrotz wünscht sich die Auftraggeberseite deutschsprachige Ansprechpartner im Nearshoring, wogegen Englisch als Standard im Offshoring betrachtet wird (vgl. [Amberg, M. et al. (2005), 37] und [Nicklisch, G. et al. (2009), 34]). Ebenso gehört das Beherrschen der englischen oder deutschen Sprache zu den Erfolgsfaktoren im Offshoring-Report der Bitkom [Deutsche Bank Research (2005), 18].

Bei einem Informationsaustausch in einer Sprache, die nicht der eigenen Muttersprache entspricht, besteht die Gefahr, dass das Gesagte oder Geschriebene beim Empfänger der Nachricht nicht richtig verstanden und daher nicht korrekt interpretiert wird. Beispielsweise wäre es von Vorteil auf schriftliche Medien wie Instant Messaging umzusteigen, wenn eine Seite Probleme mit der Aussprache des Vertragspartners hat.

3.2.4 Sicherstellen eines beidseitigen Wissenstransfers

In einem Offshoring-Projekt muss besonderer Wert auf die Vermittlung des für die Abwicklung notwendigen Wissens gelegt werden. Die involvierten Unternehmen sollen die Abläufe und Prozesse des anderen Unternehmens verstehen lernen, idealerweise findet eine Annäherung auf Prozessebene statt. Der Lieferant sollte über das Geschäftsfeld Bescheid wissen, dies gilt besonders in komplexeren Projekten. Bei modularen Standardapplikationen mag es ausreichend sein, diese Informationen knapp zu halten, bei Individualentwicklungen darf ruhig mehr ins Detail gegangen werden. Des Weiteren kann es notwendig sein, die Architektur rund um die zu entwickelnde Software sogar bis ins Detail zu vermitteln. Damit wird sichergestellt, dass der Anbieter während der Projektabwicklung nicht nur an der Spezifikation festhält, sondern bei Auftreten von Unklarheiten proaktiv an die Lösung der Fragestellung herangehen kann. Eine tiefe Ein-

führung in die Domäne der Software ist bei Erweiterungen von bestehender Software ohnehin unumgänglich. Dieser Know-how Transfer muss über die Laufzeit aufrecht erhalten werden, insbesondere wenn intern an den angebundenen Applikationen Änderungen vorgenommen werden. Jensen unterstreicht dies in seinen Best Practices mit der Aussage, dass auf einen kontinuierlichen Wissenstransfer Wert gelegt werden sollte. Darüber hinaus kann es ein schwerer Fehler sein, interne Know-how-Träger vom Offshoring-Projekt abzuziehen [Jensen, M. et al. (2007), 4]. Zu beachten gilt, dass der Anbieter möglicherweise mit unternehmenskritischen Informationen versorgt wird. Dies kann zu Problemen führen, da der Lieferant im hier betrachteten Fall des externen Offshorings natürlich auch andere Kunden bedient, die ebenfalls im gleichen Geschäftsfeld operieren (vgl. [Amberg, M. et al. (2006), 52]).

Im Laufe des Projekts muss im umgekehrten Fall dafür Sorge getragen werden, dass das Wissen, welches zusätzlich vom Anbieter aufgebaut wird, auch wieder in die eigene Organisation zurück fließt. Es besteht die Gefahr einer erhöhten Abhängigkeit vom Lieferanten, ein Wechsel des Anbieters wird erschwert (vgl. [Amberg, M. et al. (2005), 41]). Werden komplexe Individualprojekte offshore abgewickelt oder wird eine strategische Partnerschaft eingegangen, ist ein Wechsel des Anbieters aufgrund der hohen Anlaufkosten ohnehin nur begrenzt anzuraten.

3.2.5 Frühzeitige und regelmäßige Kontrolle der Projektergebnisse

Amberg verweist darauf, dass ein integriertes Qualitätsmanagement von enormer Bedeutung für den Projekterfolg ist. Bezüglich des Projektmanagements stellt er klar, dass die Kontrolle des Projektfortschritts mit Hilfe von Projektmanagementmethoden, wie ein detaillierter Projektplan oder das Führen von einem Pflichtenheft, unterstützt werden kann (vgl. [Amberg, M. et al. (2005), 40]). Im Prinzip läuft es darauf hinaus, dass auf Anbieterseite ein gewisser Grad an Prozessreife in der Softwareentwicklung erreicht sein sollte. Viele Lieferanten sind daher nach den gängigen Qualitätsstandards CMMI, ISO900x oder Six Sigma zertifiziert, um potentiellen Kunden diese Prozessreife zu signalisieren. Bei der Frage zur Bedeutung von verschiedenen Faktoren bei der Auswahl eines Offshoring-Partners wurde in der Studie von Bitkom der Einsatz von Qualitätsmanagementsystemen an prominenter 4. Stelle gereiht ([Deutsche Bank Research 2005, 19].

Zu beachten gilt, dass eine Zertifizierung zwar das Vorhandensein von gelebten Prozessen zum Zeitpunkt eines Audits vorgibt, aber nicht zwangsweise belegt, dass diese Prozesse auch voll umfänglich gelebt werden. Im umgekehrten Fall kann es durchaus sein, dass heimische Unternehmen nicht die Prozessreife von Lieferanten besitzen. Sind beide Unternehmen in Bezug auf diesen Level im Ungleichgewicht, kann es zu Problemen während des Projekts kommen. Der Auftraggeber könnte die Chance wahrnehmen, um in der eigenen Organisation die Prozesse zu verbessern.

Jensen fordert ebenfalls eine detaillierte, sorgfältige Qualitätskontrolle aller Deliverables. Ziel dabei ist, Fehler und Missverständnisse frühzeitig aufzudecken, um rechtzeitig eingreifen und Korrekturen anbringen zu können. Je später Fehler erkannt werden, desto schwieriger und kostspieliger wird es den Fehler und etwaige Folgefehler zu beheben [Jensen, M. et al. (2007), 6].

In der Praxis bedeutet das ein Anpassen der hauseigenen Softwareentwicklungsprozesse an die Gegebenheiten beim Anbieter und umgekehrt. Das Einführen eines Software-Entwicklungsmodells, sollte dies nicht bereits im Unternehmen etabliert sein, erscheint unumgänglich. Bestehen bereits gut strukturierte und gelebte Prozesse im Unternehmen, gilt es die Schnittstellen zum Lieferanten zu definieren und ein Regelwerk für die Zusammenarbeit auszuarbeiten. Aus meiner Sicht macht es wenig Sinn, wenn Anbieter und Kunde nach unterschiedlichen Modellen arbeiten. In der Praxis geschieht das häufig, wenn Anbieter vorgeben, nach allen gängigen Methodiken arbeiten zu können. In dem Fall liegt der Verdacht nahe, dass oberflächlich betrachtet auf die Anforderungen des Kunden eingegangen wird, hinter den Kulissen aber nach den eigenen Methoden gearbeitet wird.

3.2.6 Geographische Nähe des Anbieters

Der letztgenannte Faktor bezieht sich auf die geographische Distanz zum Anbieter. Steigende Distanz wirkt sich negativ auf die wesentlichen Projektmanagement-Aufgaben aus, nämlich Kommunikation, Koordination und Steuerung (vgl. [Nicklisch, G. et al. (2009), 33].

Fällt die Wahl auf einen Offshoring-Anbieter, wird es unter Umständen ungemein schwierig angemessene Reaktionszeiten auf Anfragen zu erlangen bzw. einen Teamkollegen im fernen Ausland direkt via Telefon oder Videokonferenz zu erreichen. Der

Grund dafür liegt in den unterschiedlichen Zeitzonen, in denen beide Vertragspartner operieren. Heimische Unternehmen müssen sich darauf einstellen nur kurze Zeitschlitze für Konferenzschaltungen vorzufinden und längere Wartezeiten für Antworten auf schriftliche Anfragen einzuplanen. Je weiter weg der Offshoring-Partner vom Heimatland operiert, desto schwieriger wird es einen kontinuierlichen Kommunikationsfluss aufrecht zu erhalten.

Eine ähnliche Argumentation gilt für das Zustandekommen persönlicher Treffen und dem damit eng verbundenen Wissenstransfer. Nearshoring-Anbieter können rascher auf kritische Situationen im Projekt reagieren und Projektmitarbeiter an den Ort des Auftraggebers entsenden und vice versa. Somit erschwert Distanz auch die Bildung von Teams. Distanz wirkt sich negativ auf die Zusammenarbeit und einen Know-How-Transfer aus.

Heimische Unternehmen wünschen sich wie oben erwähnt deutschsprachige Kontaktpersonen. Von Ausnahmen abgesehen, wird sich dies wohl eher im benachbarten Ausland bewerkstelligen lassen.

3.3 Delivery-Modelle unter Berücksichtigung der Erfolgsfaktoren

3.3.1 Onsite Delivery-Modell

Das Onsite Delivery-Modell bietet das höchste Maß an Kontrolle, da die Mitarbeiter des Dienstleisters vor Ort am Firmensitz tätig werden. Diese hohe Kontrollmöglichkeit wird allerdings mit dem zweithöchsten fiktiven Kostensatz erkauft. Für einen Vergleich setzen wir den Wert 80 an (vgl. Abbildung 3). Die Möglichkeit der direkten Face-to-Face Kommunikation hat positive Effekte auf den Projekterfolg, Sprachbarrieren können in der direkten Diskussion leichter überwunden werden. Es ist zu Beginn lediglich ein geringer Know-how Transfer nötig, da interne Mitarbeiter das nötige Spezialwissen über die Branche, Geschäftsprozesse und Systemlandschaft laufend beisteuern können. Das Onsite-Modell bietet sich daher grundsätzlich für zeitkritische Projekte, Prestigeprojekte, Geheimprojekte, Entwicklung strategisch wertvoller Softwareapplikationen (vgl. [Amberg, M. et al. (2006), 29]) und wichtiger Kundenprojekte an.

Ist zum Projektstart oder im Rahmen des Anforderungsmanagements absehbar, dass die Qualität der Requirements nicht ausreichend detailliert bereitgestellt werden kann, hilft

eine Onsite-Präsenz des Anbieters die Unsicherheit im Projekt zu minimieren. Der Auftragnehmer kann den Prozess begleiten und ist in der Lage auf die Bedürfnisse des Kunden direkt einzugehen und bei sich ändernden Anforderungen rasch zu reagieren (vgl. [Faiz, M. et al. (2007), 2]).

[Gadatsch, A. (2006), 47] spricht in diesem Zusammenhang von reinem Body-Leasing. Der Unterschied liegt allein in der Beauftragung von günstigeren Offshore-Anbietern. In der Literatur gehen viele Autoren meiner Wahrnehmung nach davon aus, dass es sich zwingend um einen Anbieter aus dem Ausland handeln muss. Wird der Begriff IT-Sourcing weit genug gefasst, ist es leicht denkbar, dass Anbieter aus dem gleichen Land Onsite anbieten (klassisches Outsourcing).

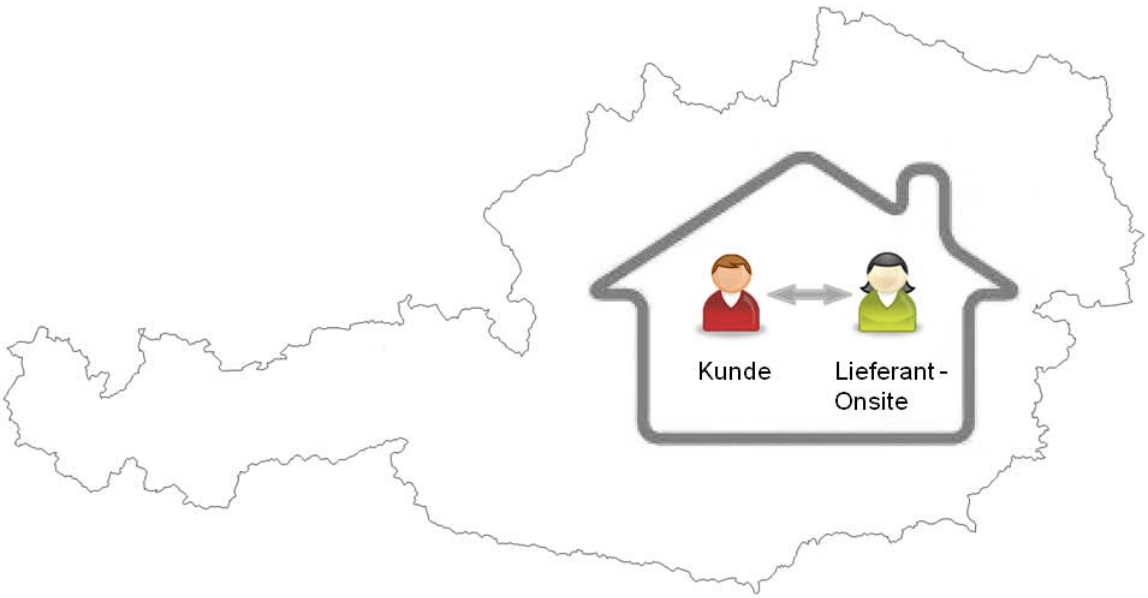


Abbildung 4: Onsite Delivery

3.3.2 Onshore Delivery-Modell

Dabei bietet ein Offshore-Dienstleister dem Kunden seine Services in dessen Land an. Der Anbieter betreibt eine Niederlassung, hat ein Unternehmen akquiriert oder entsendet Mitarbeiter aus seinem Mutterland. Idealerweise mietet sich der Anbieter im direkten Umfeld des Auftraggebers ein. Kontrollmechanismen greifen noch ausreichend gut,

obgleich die räumliche Distanz einen permanenten Austausch Face-to-Face nicht mehr zulassen. Diese Art der Kommunikation ist bei wenig komplexen Projekten keine zwingende Voraussetzung. Der Anbieter kalkuliert höher als im Fall von Onsite-Delivery. Der relative Kosten-Vergleichswert wäre die Referenz 100 (vgl. Abbildung 3). Der Grund warum Onshore teurer ist als Onsite liegt darin, da der Lieferant Räumlichkeiten und technische Infrastruktur anmieten muss, wenn der Kunde keine Möglichkeit hat, den Lieferanten am Firmensitz aufzunehmen. Deshalb kann der Lieferant nicht niedriger kalkulieren (vgl. [Faiz, M. et al. (2007),2] und [Böhm, C. (2004), 5] zu Multiplikatoren der Offshore-Rate bei Onsite- und Onshore-Präsenz).

Ein ausgeklügeltes, institutionalisiertes Kommunikationssystem schafft Abhilfe und ist in vielen Fällen ausreichend, z.B. wenn zu Projektbeginn bereits in einem höheren Ausmaß Klarheit über die umzusetzenden Anforderungen herrscht. Wie auch beim Onsite Delivery-Modell ist bei großzügiger Auslegung des Begriffs IT-Sourcing ein Dienstleister aus dem eigenen Land denkbar (vgl. [Ebert, C. (2006), 13]).

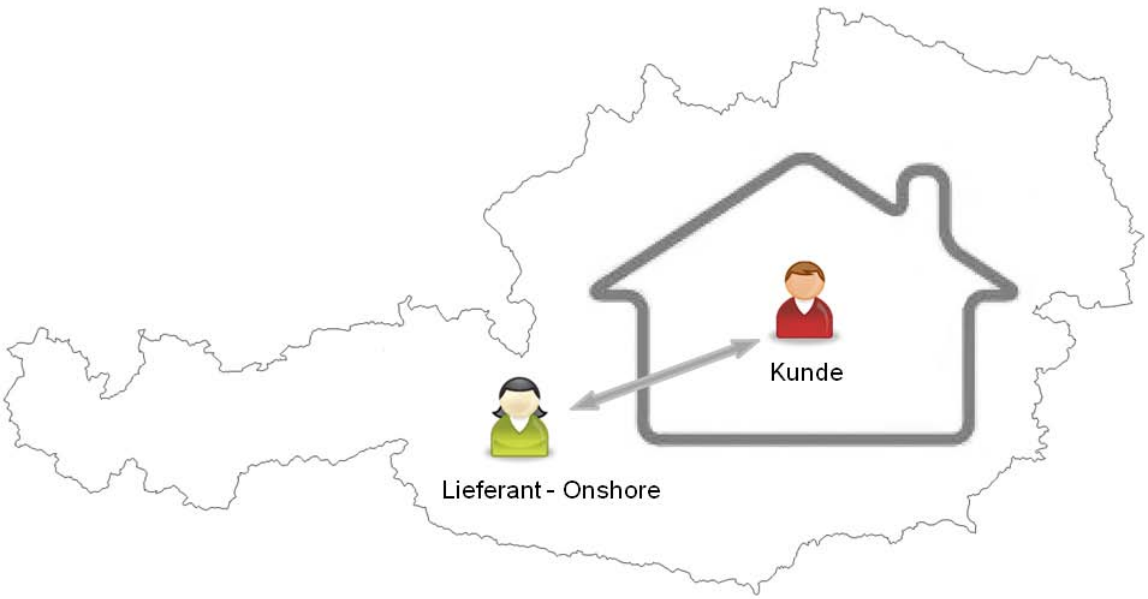


Abbildung 5: Onshore Delivery

3.3.3 Nearshore Delivery-Modell

Bei diesem Modell wird die gesamte Leistung im angrenzenden Ausland erbracht. Obgleich das Nearshore-Team nicht mehr unmittelbar greifbar ist, steht durch die geographische Nähe und damit einhergehende minimale Zeitzonenverschiebung ein angemessener Zeitschlitz für einen Informationsaustausch zur Verfügung. Persönliche Besuche sind beiderseits mit einem nicht unerheblichen Aufwand möglich und begünstigen den Projekterfolg, ansonsten müssen andere Wege wie z.B. Videokonferenzen und Online-Chats gefunden werden. Günstigere Raten im Vergleich zu Onsite- beziehungsweise Onshore-Delivery erkauft man sich hier mit Kontrollverlust, der Benchmark des Stundensatzes in diesem Fall wäre in etwa 60 (vgl. Abbildung 3).

Mit Sprachbarrieren muss natürlich gerechnet werden, obwohl die im Nearshoring bevorzugte Sprache Deutsch häufiger vorgefunden werden würde als im Offshoring.

Nach einer Studie von Moczadlo wird Nearshoring bevorzugt von kleinen und mittleren Unternehmen gewählt [Moczadlo, R. (2005), 3]. Da in kleineren Betrieben das Prozessdenken oftmals nicht dominant im Vordergrund steht, darf und soll auf eine formale Zertifizierung des Anbieters verzichtet werden. Der Unterschied in der Arbeitsweise wäre unter Umständen zu hoch, wie auch in weiterer Folge die Reibungsverluste während des Projekts. Um die Qualität zu sichern, sollte man sich auf einen Prozess und auf einige der wichtigsten Ergebnistypen bzw. –dokumente einigen und deren Bearbeitung sorgfältig überwachen. Nearshoring bietet sich auch als Einstiegsdroge für Unternehmen mit wenig Erfahrung in der Fremdvergabe von Softwareentwicklungen im In- und Ausland an, da Risiken bezogen auf Kommunikation, Kontrolle, eventuell der Sprache und kultureller Unterschiede abgeschwächt werden können.

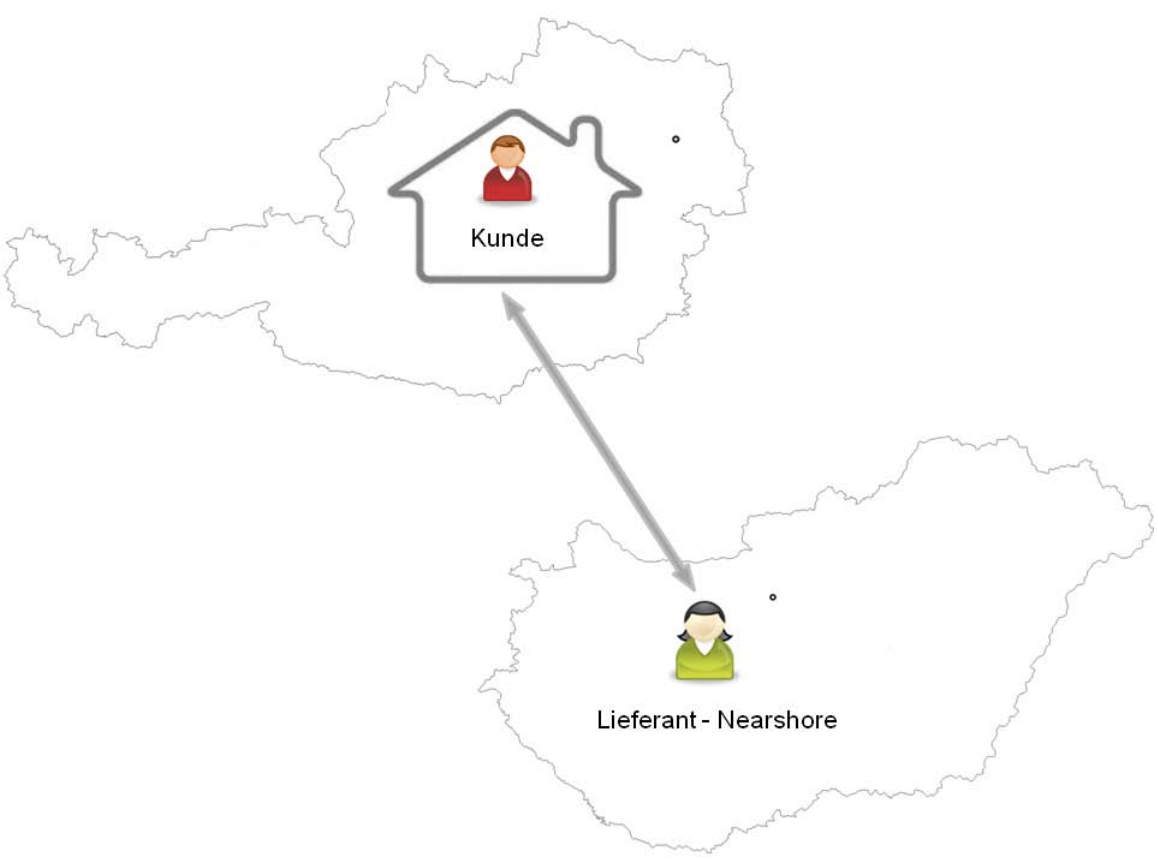


Abbildung 6: Nearshore Delivery am Beispiel Ungarn

3.3.4 Offshore Delivery-Modell

Beim Offshore Delivery-Modell wird die gesamte Leistung im fernen Ausland erbracht. Eine Kommunikation Face-to-Face ist lediglich sporadisch möglich, es steht nur der Weg zu allen technisch möglichen Kommunikationsmedien offen, sofern sie an beiden Standorten verfügbar sind. Trotzdem sei nochmals darauf hingewiesen, dass speziell zu Projektbeginn unbedingt Delegationen zwecks Austauschs abgestellt werden sollten. Günstige Raten von 20 im Vergleich zu 60 beim Nearshore erkauft man sich hier mit massivem Kontrollverlust (vgl. Abbildung 3). Dazu kommen hohe zeitliche Unterschiede, die das Management der Projekte erschweren und kulturelle Schwierigkeiten.

Für wenig komplexe und stark abgrenzbare Projekte ist dies ein gut denkbares Modell, die Voraussetzung dafür ist jedoch ein klares Verständnis der Anforderungen. Entweder ist der Kunde in der Lage nach international gängigen Methoden die Requirements zur Verfügung zu stellen, oder auf Auftraggeberseite besteht wenig bis keine Kenntnis über die Materie und der Offshoring-Partner ist in der Lage dieses Defizit durch seine speziellen Fähigkeiten zu kompensieren [Faiz, M. et al. (2007), 2].

[Amberg, M. et al. (2006), 30] verweist auf einige Autoren, die der Meinung sind, dass ein komplettes Offshore Delivery nicht möglich sei. Fraglich in diesem Zusammenhang ist, was die Autoren konkret unter kompletter Offshore Delivery verstehen, denn eine Mitarbeit des Kunden ist meiner Ansicht nach in jedem Projekt notwendig.

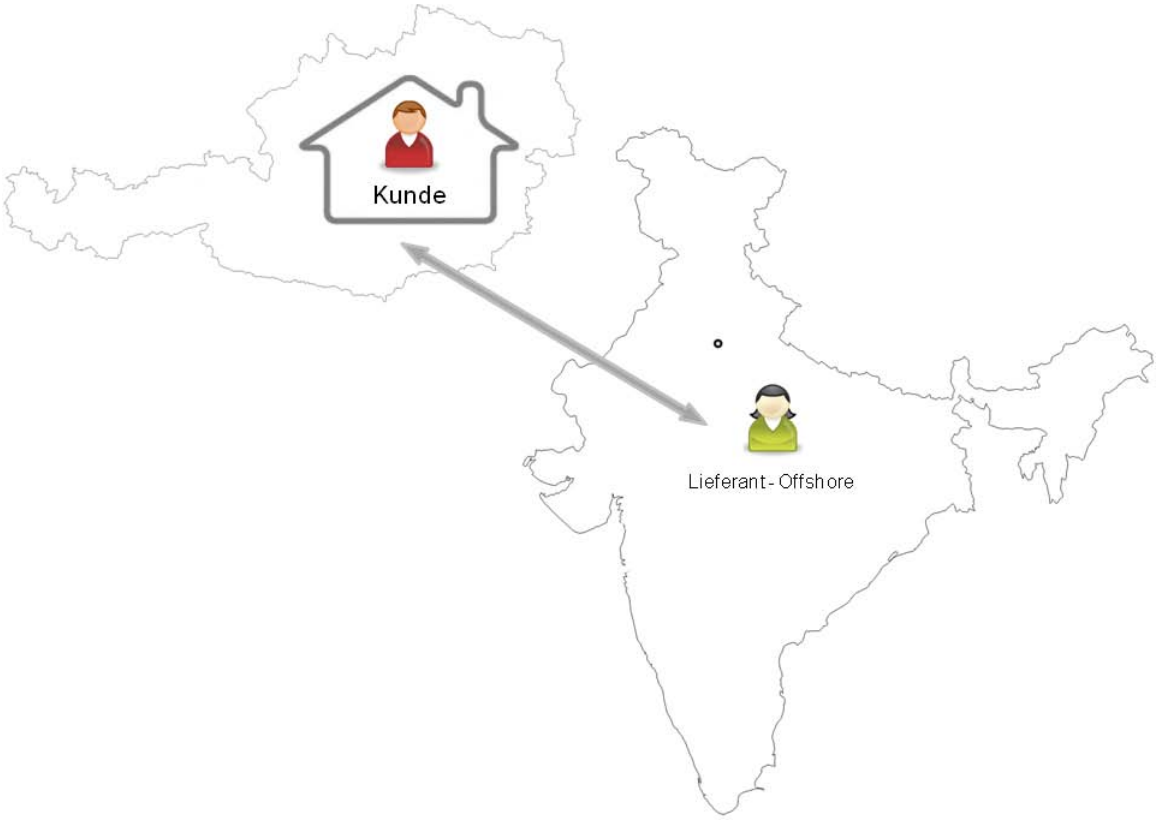


Abbildung 7: Offshore Delivery am Beispiel Indien

3.3.5 Delivery-Modelle mit Onsite- oder Onshore- Präsenz

Darunter fallen vier Modellvariationen mit jeweils einer zusätzliche Schnittstelle, nämlich Onsite-Nearshore, Onsite-Offshore, Onshore-Nearshore und Onshore-Offshore. Vernachlässigt wird in diesem Rahmen das Modell Onsite-Onshore, das bei genauer Betrachtung dem nationalen Sourcing, einer gewöhnlichen Kunde-Lieferant Beziehung, gleichkommt, außer ein Offshoring-Anbieter stellt seine Mitarbeiter Onshore bereit. Auswirkungen sind ähnlich wie oben beschrieben beim reinen Onsite- und Onshore-Modell. Anbieter erschließen den heimischen Markt, indem sie im Inland eine Unternehmensrepräsentanz aufbauen oder ein Unternehmen akquirieren. Sie tragen damit den vorherrschenden Ressentiments von Unternehmen Rechnung, die bisher Hemmungen gehabt haben, in die Welt des Offshorings einzusteigen. Insbesondere die geographische

Distanz soll in der Wahrnehmung des Kunden minimiert werden. In einer zweiten Variante entsendet der Lieferant für die Dauer der Zusammenarbeit Mitarbeiter in das Land des Kunden.

Der Anbieter kann so einen günstigen Mischsatz (Blended Rates zwischen 44 und 76) anbieten, der im Vergleich zu den reinen Formen zwar höher, aber immer noch günstiger als bei heimischen Vendoren sein wird.

Diese Modelle erfreuen sich mittlerweile hoher Beliebtheit, wie die Studie von Moczadlo zeigt. Demnach nutzen 40% der heimischen Unternehmen das Onsite-Offshore-Modell (im weiteren Sinne). Signifikante Unterschiede sind zwischen der Near- und Offshore- Variante zu beobachten. Demnach nutzen jene Unternehmen, die Leistungen in weit entfernten Ländern nachfragen zu ungefähr 60% das Onsite-Offshore Modell. Jene die im Nearshoring aktiv sind bevorzugen zu ca. 30% die Onsite-Präsenz des Anbieters [Moczadlo, R. (2005), 6].

Vorteilhaft erscheint diese Vorgehensweise bei Projekten mittlerer Komplexität, indem ein „Verbindungsoffizier“ ([Gadatsch, A. (2006), 35] und [Amberg, M. et al. (2006), 28]), auch „Brückenkopf“ genannt, während des gesamten Projektverlaufs als Schnittstelle zwischen den Vertragspartnern fungiert. Dieser Brückenkopf kommt idealerweise aus dem kulturellen Kreis des Anbieters oder kennt die kulturellen Eigenheiten und Organisationsstruktur ausgezeichnet. Es ist permanent ein Mitarbeiter vor Ort, der eine kontinuierliche Kommunikation zum Kunden aufrecht erhalten kann und bei inhaltlichen Problemen als Mediator auftritt.

Bei Projekten mit hoher Komplexität kommt es vor, dass dieser Brückenkopf aus mehreren Mitarbeitern vor Ort besteht oder sogar die komplette Projektorganisation gespiegelt wird.

Bei dieser Variante werden Mitarbeiter eingesetzt, die über verschiedenste Qualifikationen verfügen, wie z.B. Business Analysten, Architekten, Test-Ingenieure, die in der Lage sind mit dem Kunden gemeinsam Requirements aufzubereiten und das System Design zu entwickeln. Das ressourcenaufwendige Coding findet im entfernten Ausland statt.

Mit diesem Modell kann ein guter Wissenstransfer jedenfalls zum Onsite-Team gewährleistet werden. Die Kommunikation zur Offshore-Destination kann nicht direkt beeinflusst werden.

Jensen ortet Vorteile darin einen Mittelsmann vor Ort bereitzustellen, der technische Fragen aufnimmt und die Verbindung zwischen den richtigen Personen beim Kunden und Lieferanten offshore herstellt. Er stellt klar, dass in der Praxis die Zusammenarbeit besser funktioniert, wenn sich die Teammitglieder kennen. Daher tritt er dafür ein, dass im Laufe des Projekts jeder Mitarbeiter zumindest einmal beim Kunden einen Besuch abstattet [Jensen, M. et al. (2007), 7].

Obwohl ein ständiger Face-to-Face Austausch mit Vertretern des Anbieters stattfinden kann, liegt es doch im Interesse des Kunden, die Mitarbeiter jenseits der Landesgrenze ins Team einzubinden. An dieser Stelle sei darauf hingewiesen, dass es durch die höhere Anzahl an Schnittstellen zu Informationsverlusten kommen kann. Trotz Onsite-Präsenz wird eine strukturierte Online-Kommunikationsplattform (gemeinsame Repositories und Projektplattformen) angeraten. Durch die ständige Anwesenheit des Lieferanten über die Projektlaufzeit kann jedenfalls eine dauerhafte Qualitätssicherung der Arbeitsergebnisse erzielt werden.

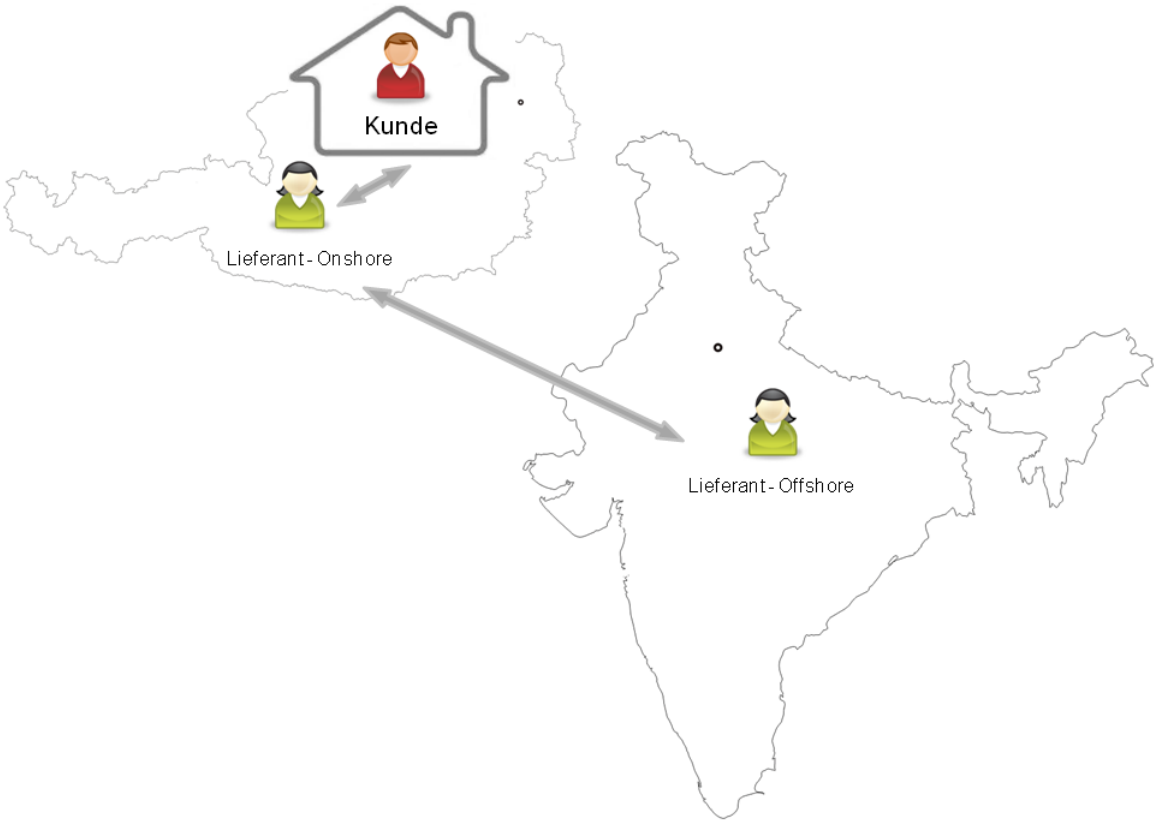


Abbildung 8: Onshore–Offshore Delivery am Beispiel Indien

3.3.6 Global Delivery-Modelle

Hier wird davon ausgegangen, dass diese Modelle mehr als 2 Schnittstellen zur Informationsweitergabe aufweisen. Es sind alle nicht genannten Kombinationen denkbar, dabei belaufen sich die Stundensätze im fiktiven Bereich von 40-72. Im Vergleich zu den kombinierten Delivery-Modellen mit zwei Schnittstellen, ergibt sich eine geringe Kostensenkung. Aus meiner Sicht gibt es zwei Ausprägungen zu beachten. Global agierende Beratungsunternehmen und Software-Entwicklungshäuser wie Accenture, Cap-Gemini, Unisys oder Siemens akquirieren Offshore-Dienstleister oder bauen selbst Facilities in Nearshore- und Offshore-Ländern auf. Dadurch sind sie theoretisch in der Lage, für jedes spezifische Projekt oder strategische Partnerschaft die bestmögliche Sourcing-Variante anzubieten.

Während der Laufzeit können beliebig Ressourcen hinzugezogen und wieder abgebaut werden, je nachdem, ob sie gerade gebraucht werden. Am Firmensitz ist meist ein Onsite-Team, das die Kommunikation mit dem Kunden übernimmt und die Koordination der Offsite-Teams aussteuert [Faiz, M. et al. (2007), 3].

In der zweiten Ausprägung wählt der Kunde gezielt Anbieter aus, die dem Anforderungsprofil für eine bestimmte Software-Komponente am besten entsprechen. Auch ein Engagement eines Dienstleisters, der etwa speziell auf das Testen spezialisiert ist, wäre denkbar. Auf ein Projekt bezogen bedeutet das, fähig zu sein, viele Human- Schnittstellen bedienen zu können.

Empfehlenswert ist diese Methode meiner Wahrnehmung nach nur für (Groß-) Unternehmen, die bereits über eine langjährige Erfahrung mit der Steuerung von Offshore-Projekten haben und ein rigide, prozessorientierte Vorgehensweise beherrschen.

Ein Vorteil dieser Vorgehensweise besteht darin, wenn die Voraussetzungen beim Kunden stimmen, dass Projekte rascher abgewickelt werden können. In der Literatur liest man von „Follow the Sun“-Entwicklung oder „Round the World-Engineering“. Beispielsweise beginnt ein Zyklus mit Requirements Engineering onsite, System Design nearshore, Entwicklung offshore in China, eine erste Qualitätssicherung nearshore in Ungarn, Deployment und Abnahme onsite beim Kunden (vgl. [Gadatsch, A. (2006), 33] und [Amberg, M. et al. (2006), 32]).

Da sich die Zusammensetzung des Teams jederzeit ändern kann, wirkt sich diese Art der Zusammenarbeit negativ auf einen kontinuierlichen Kommunikationsfluss und Wissenstransfer aus.

Global Delivery, Global Sourcing, Multi Sourcing sind alles Begriffe, die unter diesem Punkt subsummiert werden können. Amberg zählt sie nicht zu den Delivery-Modellen, sondern reiht sie unter Geschäftsmodelle, die durch die Besitzstruktur geprägt sind [Amberg, M. et al. (2006), 30]. Ich zähle sie zu den Delivery-Modellen, da es hier um Fremdvergabe geht, nicht um Joint Ventures und auch nicht um den Aufbau eines Tochterunternehmens.

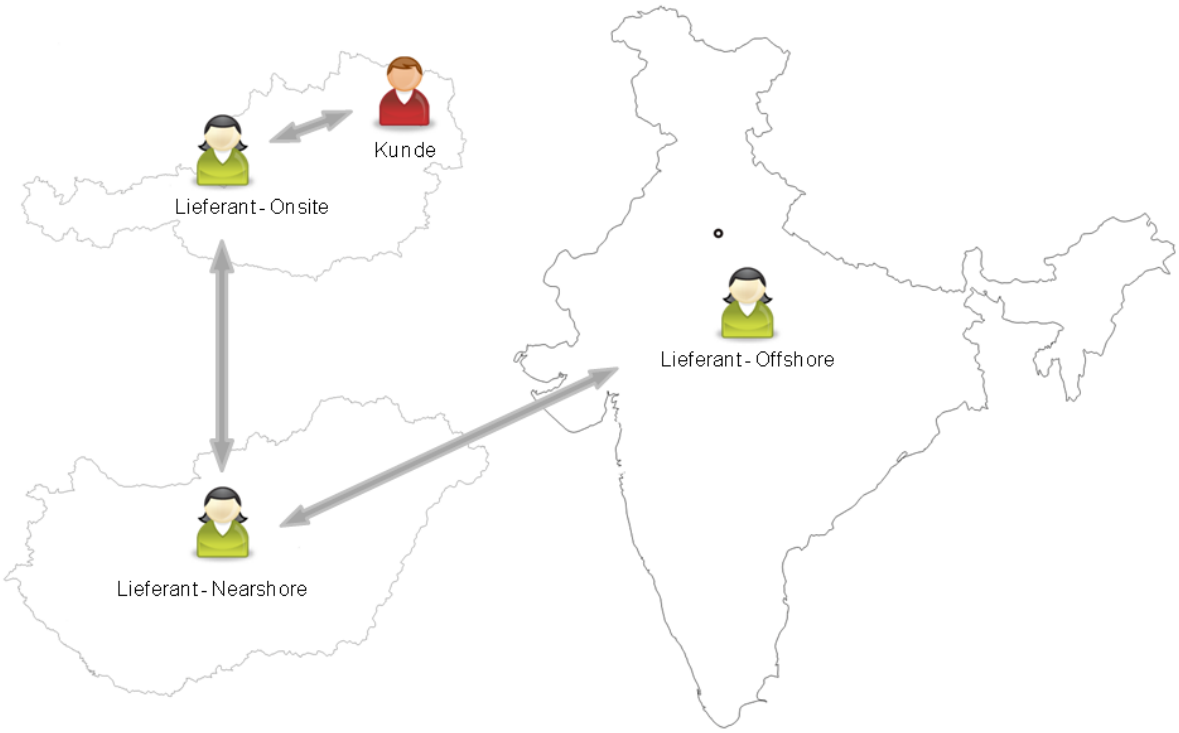


Abbildung 9: Global Delivery

4 Kollaborationsmodelle

Bisher haben wir beleuchtet, welche organisatorischen Kombinationen bezogen auf ein verteiltes Software Development-Projekt theoretisch möglich sind. Nun gehen wir tiefer ins Detail und betrachten welche Konstellationen bezogen auf Aktivitäten bzw. Software-Engineering Disziplinen sinnvoll möglich sind und in der Praxis gerne angewendet werden. Im Prinzip handelt es sich um eine Entscheidung wie früh im Software-Entwicklungsprozess der Anbieter mit einbezogen wird.

Die Gliederung orientiert sich gewissermaßen an der oben beschriebenen Klassifizierung und Unterscheidung nach dem Grad des externen Leistungsbezugs und ist des Weiteren eine Ausprägung des Software Development Sourcings in der Klassifikation nach dem Grad der Geschäftsorientierung (vgl. Abbildung 2).

In der Praxis werden zwei Extreme beobachtet, die „Extended Workbench“ und „Over the Fence“ (vgl. Abbildung 11 und Abbildung 12). In weiterer Folge ergibt sich eine Vielzahl an Varianten in der Arbeitsteilung, nicht zuletzt durch die Verwendung bestimmter Software-Entwicklungsmodelle.

In der Studie nach Moczadlo [Moczadlo, R. (2005), 5] wurden deutsche Unternehmen unter anderem nach der Rolle des Offshore-Partners in der Projektplanung im Vergleich zwischen Offshoring und Nearshoring befragt. Eine Frage zielt darauf ab, wie frühzeitig ein Anbieter in den Prozess eingebunden wird. Unabhängig von der Tatsache, ob es sich um Near- oder Offshore-Kontrakte handelt, kann interpretiert werden, dass jegliches Modell in etwa gleich oft in der Realität angewendet wird, jedoch sticht mit knapp über 30% eine Möglichkeit der Zusammenarbeit heraus: Der Offshore-Partner erhält einen fertigen Systementwurf und strukturiert den Programmentwurf. Das bedeutet, dass knapp ein Drittel der deutschen Unternehmen gerne Architekturentscheidungen im eigenen Haus belassen, genauso wie die Strukturierung des Systems. Auf der anderen Seite ist der Lieferant in 36% der Fälle maßgeblich bei der Erstellung des Sollkonzepts involviert, wobei in 18,4% der Fälle der Lieferant direkt am Requirements Engineering und System Modelling beteiligt ist. Der klassischen „Extended Workbench“-Methode folgen etwa 16,2% der Auftraggeber.

Interessant ist der Unterschied zwischen Nearshore- Destinationen und Indien als Offshore-Vertreter. Das weit entfernte Indien wird signifikant öfter in die Sollkonzepterstel-

lung eingebunden als Vertreter im nahe gelegenen Ausland. Ich führe das auf zwei Gründe zurück. Indien ist nach wie vor das führende Land am Outsourcing-Sektor und kann mit einem hohen Qualitätsstandard und Bildungsniveau, hoher technischer Kompetenz und vorzüglichen Englischkenntnissen aufwarten (vgl. [Amberg, M. et al. (2006), 127] und [Ebert, C. (2006), 47ff]). Demzufolge sind indische Anbieter außerordentlich gut geeignet, Aktivitäten mit dem Auftraggeber durchzuführen, die eine hohe Kritikalität und Kommunikationsaufwand bedürfen. In meiner beruflichen Laufbahn haben sich die Schritte rund um Requirements Engineering, System Modelling und Architectural Design zweifelsohne als projektkritisch herausgestellt. Als zweiten Grund sehe ich die stärker werdende Beliebtheit von kombinierten Delivery-Modellen mit Onsite-, oder Onshore-Präsenz und Global Delivery-Modellen. Wie zuvor erwähnt ermöglicht ein Unternehmenssitz des Lieferanten im Land des Auftraggebers eine direkte Bearbeitung des Marktes. Somit können heimische Unternehmer von den Fähigkeiten indischer IT-Experten profitieren. Die Unternehmen dürften sich auf die Fertigkeiten in Osteuropa, also im Nearshoring, insofern eingestellt haben, als dass sie hauptsächlich detailliert ausgearbeitete Spezifikationen bis zum Programmentwurf bzw. Software Design als Basis der Projektabwicklung heranziehen. Die Softwareindustrie in Osteuropa verspürt zwar ein starkes Wachstum, allerdings ist eine unzureichende Prozessorientierung und mangelndes Qualitätsbewusstsein einem Projekterfolg abträglich. „Unternehmen liefern, was verlangt wird, und das kann auch das Falsche sein. Unzureichende Spezifikationen werden spät entdeckt“ [Ebert, C. (2006), 55].

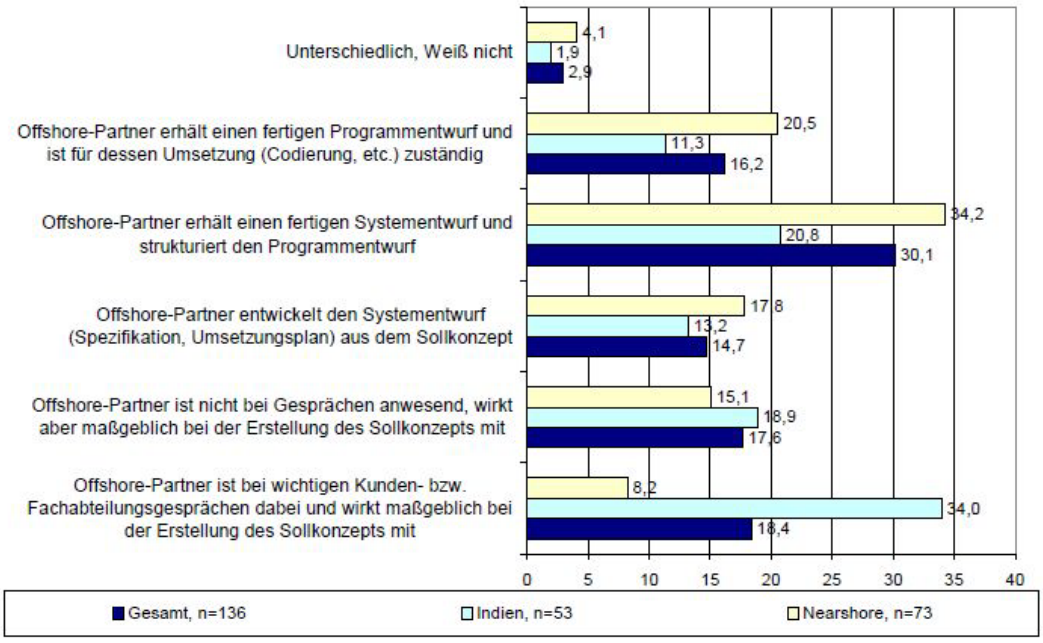


Abbildung 10: Offshore-Arbeitsteilung im Vergleich [Moczadlo, R. (2005), 6]

4.1 Kritische Erfolgsfaktoren

Mit Hilfe der bereits zur Bewertung der Delivery-Modelle herangezogenen exploratorischen Studie zu kritischen Erfolgsfaktoren von Michael Amberg können wir Faktoren isolieren, die für eine Bewertung äußerst hilfreich sind. Zur Erinnerung wurden die identifizierten Erfolgsfaktoren nach internen und externen Eignungs- sowie Managementfaktoren klassifiziert. Um herauszufinden welche Aktivitäten besser im eigenen Unternehmen durchgeführt werden sollen und welche besser im Offshore-Modus abgewickelt werden, ist die interne Sichtweise grundsätzlich maßgeblich und es wurden aus den dreizehn Erfolgsfaktoren dieser Kategorie acht isoliert (vgl. [Amberg, M. et al. (2005), 29ff.]

Interne Eignungsfaktoren sind nach Amberg „Erfolgsfaktoren hinsichtlich der Offshore-Readiness des Auftraggebers“ und interne Managementfaktoren sind „Erfolgsfaktoren hinsichtlich der Planung eines Offshore-Software-Entwicklungsprojekts“ (vgl. [Amberg, M. et al. (2005), 31]).

Die Auswahl einer dieser Optionen hängt von Merkmalen beim Auftraggeber ab, wenn davon ausgegangen werden darf, dass der Anbieter ein komplettes Entwicklungsprojekt abwickeln kann und daher über die notwendigen Skills verfügt. Aus diesen Merkmalen werden Fragen abgeleitet, die sich ein Unternehmen in diesem Zusammenhang stellen sollte.

4.2 Die isolierten Erfolgskriterien im Einzelnen:

4.2.1 Umfassende Erfahrung mit IT-Outsourcing-Projekten

Vorteile orte ich bei jenen Unternehmen, die in der Vergangenheit bereits Software-Projekte extern vergeben haben. Die Mechanismen im Offshoring verhalten sich ähnlich wie im klassischen Outsourcing bezogen auf das Lieferantenmanagement. Besonders kleinere Unternehmen ohne eigene umfangreiche IT sind hier im Vorteil, da sie gezwungenermaßen ihre Software-Entwicklungsmaßnahmen bisher auch schon extern vergeben mussten. Bei großen Unternehmen hängt es davon ab, welche Strategie in den letzten Jahren verfolgt wurde. Insbesondere Großkonzerne beherbergen oft ausgezeichnet ausgestattete IT-Bereiche. Die IT stellt hier zwar keine Kernkompetenz dar, das Un-

ternehmen ist aber in der Lage, beinahe jegliche Anforderung der Fachbereiche an Softwareprodukte im eigenen Haus abwickeln zu können. Ein Vorteil ist, dass diese Unternehmen über umfängliche Kenntnisse verfügen, wie Entwicklungsprojekte optimal abgewickelt werden sollen, jedoch mangelt es in der Praxis am Lieferantenmanagement und im Umgang mit den Teammitgliedern eines Offshore-Anbieters während der Projektdurchführung. Eine Wahrnehmung aus meinem beruflichen Umfeld ist, dass Mitarbeiter des Vendors oftmals als Arbeiter zweiter Klasse angesehen werden, die einfach ihren Vertrag erfüllen sollen. Größere Unternehmen mit eigenem IT-Bereich, die intern ein hohes Maß an Durchführungskompetenz aufweisen und zudem zwecks Skalierbarkeit bereits Entwicklungstätigkeiten an einen inländischen Lieferanten ausgelagert hatten, sind ebenfalls klar im Vorteil.

4.2.2 Angemessenes technisches Verständnis auf Kundenseite

Gehört IT Software Entwicklung nicht unbedingt zu den Kernkompetenzen eines Unternehmens, ist es doch eine zwingende Voraussetzung, dass auf Kundenseite die Komplexität eines solchen Vorhabens bewusst ist. Insbesondere grundlegende Kenntnisse zur Vorgehensweise in Software-Entwicklungsprojekten und vertiefend über Methoden im Bereich Requirements Engineering sind hilfreich. Sind diese Basisfähigkeiten gegeben, steht grundsätzlich ein „Over the fence“-naher Ansatz zur Verfügung. Dies trifft am ehesten in kleineren Unternehmen zu, wenn keine Systemspezialisten wie System Analysten und Architekten beschäftigt werden und/oder der Betrieb der Applikation ebenfalls durch den Lieferanten stattfinden soll. Der Lieferant übernimmt in der Regel die gesamte Projektleitung und führt durch den Software-Entwicklungsprozess unter Miteinbeziehung des Kunden.

4.2.3 Erstellen einer detaillierten Projektspezifikation

Der Begriff Projektspezifikation ist in der Literatur nicht hinreichend definiert. In Amberg [Amberg, M. et al. (2005), 35] ist davon die Rede, dass die „Formulierung einer detaillierten Aufgabenspezifikation“ unerlässlich sei und „innerhalb der Projektbeschreibung zudem eine umfassende Beschreibung der Verantwortlichkeitsbereiche erfolgen sollte“. Ich pflichte dem uneingeschränkt bei, durchsucht man aber das IEEE Standard Glossary of Software Engineering Terminology nach dem Wort „Specification“ finden sich unzählige Treffer.

Eine Spezifikation ist demnach: „A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied" [IEEE Std 610.12 (1990), 69].

Diese Definition stellt auf den Endzustand einer Dokumentation ab, denn eine Spezifikation ist erst dann komplett, wenn alle Ungereimtheiten im Laufe des Entwicklungsprozesses ausgeräumt sind. Das ist in der Regel erst zu fortgeschrittener Projektlaufzeit der Fall (vgl. [Bullinger, H.-J. (1999), 133] zum Vorhandensein einer Grobspezifikation bei Vertragsabschluss).

Oftmals wird Projektspezifikation mit Pflichtenheft, Lastenheft und anderen Dokumenten gleichgesetzt oder in einem Atemzug mit einem Statement of Work genannt. Ein Statement of Work betitelt eine Leistungsbeschreibung im Rahmen der Vertragsgestaltung zwischen Kunde und Lieferant. Verantwortungsbereiche wie oben erwähnt und etwa Vertragsart, Projektorganisation, grober Terminplan sind Teil der Leistungsbeschreibung und Teil des Vertrags. Auch Richtlinien des Kunden für die Systemmodellierung oder Architekturrichtlinien können (und sollen) darin gelistet sein.

In diesem Kontext reicht es aus zu sagen, der Kunde sollte in der Lage sein Anforderungen an ein System in geeigneter Art und Weise zu erarbeiten. Daraus resultiert eine „Spezifikation" und zwar aus dem Schritt Requirements Engineering eine strukturierte Darstellung von Benutzeranforderungen, eine Software Requirement Specification. Verfügt der Kunde über tiefer gehendes Know-how, wäre er auch in der Lage die Software zu strukturieren und etwa das System Design vorzugeben. Requirements Engineering und System Modelling sind aus meiner Sicht die Kerngebiete der Software Engineering Disziplinen, um den Grundstein für einen erfolgreichen Projektabschluss zu legen.

4.2.4 Schaffen einer kulturellen Sensibilität

Mitarbeiter des Kunden sollten sich darüber im Klaren sein, dass es Unterschiede in den Kulturen beteiligter Länder gibt. Das Unternehmen ist dafür verantwortlich, dass sich zukünftige Teammitglieder des Offshore-Projekts damit auseinander setzen können, noch bevor ein Projekt startet. Gleiches würde natürlich auch für den Anbieter zutreffen, der Kunde hat darauf anfangs allerdings keinen Einfluss. Im Projektverlauf kann der Auftraggeber sehr wohl steuernd das Risiko kultureller Unterschiede minimieren, indem

das „Team“ per Job-Rotation durcheinander gewürfelt wird. Dies setzt allerding voraus, dass die Projektorganisation derart gestaltet ist, dass auf beiden Seiten gleichgeartete Rollen verfügbar sind. Nur so kann ein Offshore-Developer gegen einen Onsite-Developer, ein Offshore-Tester gegen einen Onsite-Tester und so fort, getauscht werden. Das unter diesem Punkt Gesagte trifft auch auf Unterschiede in den Strukturen von Unternehmen gleicher Kulturen zu.

4.2.5 Internationale Unternehmenskultur

Eine internationale Unternehmenskultur entsteht nicht auf dem Reißbrett und ist oft nur multinationalen Unternehmen vorbehalten, die Vertriebsniederlassungen oder Tochterunternehmen im Ausland betreiben. Ist zum Beispiel Englisch als Sprache für Besprechungen und bestimmte Dokumente manifestiert, ist schon einmal das Sprachproblem in vielen Fällen gelöst. Hat eine Organisation bereits häufig mit anderen Kulturen zu tun gehabt, ist jedenfalls eine gewisse Sensibilisierung vorhanden, die es erleichtert mit Menschen anderer Kulturen umzugehen. Auf der anderen Seite ermöglicht es kleineren und schlankeren Unternehmen rascher und flexibler auf Erfordernisse am Markt zu reagieren, da Strukturen nicht so stark eingefahren sind, wie dies oftmals in Großunternehmen der Fall ist.

Zusammenfassend muss sich das Unternehmen ernsthaft die Frage stellen, welche Fähigkeiten im Bereich Software Engineering, Projektmanagement, Lieferantenmanagement und Soft-Skills, wie Einfühlungsvermögen in andere Welten, gegeben sind. Des Weiteren sollten sich Entscheider die Frage stellen welche Aktivitäten in der Hoheit des Kunden verbleiben sollen, um festlegen zu können, wie eine Zusammenarbeit mit einem Offshore-Lieferanten aussehen kann.

Die Frage lautet: Welche Skills sind im Unternehmen verfügbar?

4.2.6 Effiziente interne Organisationsstruktur

In erster Linie ist in diesem Zusammenhang eine schlagkräftige Projektorganisation zu nennen. Ob eine reine Projektorganisation oder Matrixorganisation im Unternehmen ihre Anwendung findet ist sekundär, allerdings ist dies immer im Zusammenspiel mit der Kritikalität des Projekts zu beurteilen. Grundsätzlich gilt, je kritischer ein Projekterfolg einzustufen ist, desto eher sollte die Personalverantwortung dem Projektleiter auf-

getragen werden und die Projektmitarbeiter aus ihrer Linienfunktion herausgelöst werden (z.B. Task Force).

Das Zusammenspiel zwischen strategischer und operativer Ebene spielt gerade bei anlaufenden Offshore-Partnerschaften eine Rolle. Eine anhaltende Managementunterstützung ist unerlässlich und wird mit angemessenen Kommunikationsmitteln wie regelmäßigen Projekt-Steering-Meetings erreicht. Insbesondere zu Beginn einer Partnerschaft laufen bis zum Erlangen eines eingeschwungenen Zustands viele Gespräche auf strategischer (Management-) Ebene zwischen den Vertragspartnern ab. Deren Ergebnisse sollten umgehend in die Projektarbeit zurück fließen.

Für die Auswahl einer geeigneten Projektorganisation im Offshoring gehört neben den vorhandenen Skills wie oben beschrieben und etwaigen Erkenntnissen durch Auslagerung von Projektteilen auf inländische Dienstleister auch die Auswahl eines Entwicklungsmodells, das den Gegebenheiten beim Kunden entspricht. Die Auswahl des Entwicklungsmodells geht wiederum einher mit der Gestaltung der Arbeitsteilung. Ist der Kunde eher mit klassischen Modellen vertraut, wird ein Ansatz im Rahmen von „Over the Fence“ und der „Extended Workbench“ Variationen eingesetzt werden können. Dabei werden die Teams nach den Aktivitäten oder Phasen zusammengestellt. Beispielsweise sind die Business Analysten vor Ort, die System Designer nearshore und Programmierer offshore. Eine weitere Möglichkeit besteht darin, die Teams nach deren technologischem Wissen zu strukturieren. Dabei befinden sich Entwickler von Frontends an einem Standort, während Datenbankspezialisten an einem anderen Standort sitzen. Auch eine Kombination dieser beiden Ausprägungen ist möglich, wenn etwa System Analysten oder Tester wiederum an einem anderen Standort eingesetzt werden.

Bei agilen Methoden zählen Teamzusammenhalt und eine häufige, direkte Kommunikation. Es wird eher eine Arbeitsteilung zum Tragen kommen, die sich an den geforderten Funktionalitäten des Auftraggebers orientiert. Diese Teams werden Featureteams genannt, da diese in der Lage sein sollen ein Feature, also eine Funktionalität, End-to-End bereitzustellen. Ein Featureteam besteht aus Spezialisten aller für eine Funktionalität benötigten Rollen und folgt dem Prinzip von selbst organisierenden Teams. [Eckstein, J. (2009), 25ff].

4.2.7 Standardisierte und dokumentierte Prozesse

Wenn zwei Unternehmen in einem Projekt zusammenarbeiten sollen, stoßen sie unweigerlich auf die Herausforderung unterschiedliche Welten von Prozessen verheiraten zu müssen. Strukturen passen lediglich auf einer gewissen Abstraktionsebene zusammen. Beide Unternehmen haben Geschäftsprozesse und Supportprozesse definiert, die sie unter Umständen in ihrem eigenen Geschäftsfeld vergleichbar machen. Idealerweise sind diese Prozesse standardisiert und dokumentiert. Gewöhnlich besteht in größeren Unternehmen das Bedürfnis, Projekte die dem für uns maßgeblichen Software-Entwicklungsprozess unterliegen, zu monitoren und zu steuern. Die Konzepte dazu sind in der IT-Governance zusammengefasst. Kleinere Unternehmen neigen dazu, aufgrund der geringeren Komplexität ihrer Organisation, weniger Tätigkeiten zu standardisieren. Dies kann Hemmnis oder Vorteil zugleich sein, je nachdem, wie strukturiert im Vergleich der Anbieter arbeitet. Ebert [Ebert, C. (2006), 95] schreibt als Teil des Erfolgsrezepts: „Je spezieller und chaotischer ihre Entwicklungsumgebung und –prozesse sind, desto schwieriger gestaltet sich das spätere Outsourcing-Projekt“. Er schlägt außerdem vor, dass der Lieferant zumindest Level 4 nach dem Reifegradmodell CMMI zertifiziert sein sollte und das eigene Unternehmen einen CMMI-Reifegrad von zumindest Level 3 aufweisen sollte (vgl. [CMMI Product Team (2010)]). Amberg meint im Rahmen seiner Studie, dass maximal ein Unterschied von zwei Stufen nach CMMI angestrebt werden sollte [Amberg, M. et al. (2005), 33]. Nun sind insbesondere indische Anbieter und Töchter hiesiger Beratungsunternehmen oftmals CMMI 4 oder gar 5 zertifiziert, um den potentiellen Kunden einen Entscheidungsgrund für die Zusammenarbeit in Sachen Fähigkeit zur Ausführung und ihrem Qualitätsstandard zu kommunizieren. CMMI sagt allerdings wenig über die Qualität der erzeugten Produkte und Services aus, sondern lediglich, dass die eigenen dokumentierten Prozesse verfolgt werden. Es sagt auch wenig darüber aus, wie es um die Fähigkeit letztendlich beider Vertragspartner bestellt ist, gemeinsam ein Projekt erfolgreich abzuwickeln. Das soll nicht bedeuten, dass eine Zertifizierung wertlos ist, daher folgt weiter unten eine Darstellung, wie eine CMMI Zertifizierung einem Unternehmen geholfen hat die Performance zu verbessern. Ich denke nicht, dass es zweckmäßig ist, dass der Kunde verhältnismäßig hohe Anstrengungen unternimmt, um nach CMMI zertifiziert zu werden. Ich denke aber, dass es speziell für kleinere Unternehmen hilfreich ist, ein Mindestmaß an strukturierter Vorgehensweise anzustreben, oder wenn bereits Abläufe standardisiert sein sollten, die Prozessfähigkeit zu erhöhen. Auf der anderen Seite hilft es großen Unternehmen mit schwergewichtigen,

auf ihre Besonderheiten angepassten Prozessen, diese zu verschlanken und sich näher an (Mindest-) Standards zu bewegen.

Hohe CMMI Levels sind gut mit klassischen Entwicklungsmodellen vereinbar, da sie sehr dokumentationsintensiv sind. Interessanterweise findet sich im SEI-Standard „CMMI for Development“ eine Passage, die darauf hinweist, dass Erfordernisse aus CMMI mit agilen Methoden verknüpft werden können und in weiterer Folge immer wieder Hinweise auf welche Punkte bei Anwendung agiler Methoden geachtet werden sollte [CMMI Product Team (2010), 58].

Obwohl es im ersten Moment etwas gewöhnungsbedürftig erscheint, dass ein prozessoptimierender Standard mit agilen Werten harmonieren kann, quantifiziert Sutherland einerseits Prozessverbesserungen aufgrund CMMI Level 5 Zertifizierung und andererseits Erfahrungen in Verbindung mit agilen Methoden bei Systematic, einem dänischen Software-Anbieter für Defense, Healthcare und Intelligence & National Security [Sutherland, J. et al. (2007), 1ff].

Nachdem Systematic CMMI Level 5 erreicht hatte, reduzierten sich Nachbesserungen um 42%, die Präzision von Schätzverfahren waren mit einer Unschärfe von weniger als 10% belegt und es wurden 92% aller Meilensteine erreicht. Gleichzeitig wurden zusätzliche Arbeiten signifikant reduziert. Systematic war es dadurch möglich, die Anforderungen der Kunden zeitgerecht, mit der geforderten Qualität und ohne Überziehung des Budgets zu liefern. Dabei konnte der eigene Aufwand um 31% im Vergleich zu CMMI Level 1 Unternehmen gesenkt werden. Systematic hat es geschafft, CMMI mit Scrum zu verbinden. CMMI liefet dabei Anhaltspunkte, welche Prozesse für eine hoch standardisierte Organisation benötigt werden, während Scrum für effizientes Projektmanagement im Sinne von hoher Flexibilität und Anpassbarkeit steht. Durch Scrum werden Prozesse effizient implementiert, CMMI sorgt dafür, dass alle nötigen Prozesse betrachtet werden. Sutherland versichert, dass seit der Integration von Scrum in den Rahmen von CMMI der Aufwand bei gleicher Prozesstreue bezogen auf die Einführung von CMMI Level 5 um 50% gesunken ist.

Agile Methoden sind mit CMMI vereinbar, wenngleich das Wertesystem eigentlich weit auseinander klafft. Folgend vier Unterschiede, die das verdeutlichen sollen [Eckstein, J. (2009), 189]:

– CMMI ist von Prozessen, während die Agilität von Menschen getrieben ist.
– CMMI erfordert strikte Einhaltung des Prozesses, während Agilität vom Team fordert, den verwendeten Prozess regelmäßig zu hinterfragen.
– CMMI verlangt intensive Aktivitätsnachverfolgung, während bei der Agilität das Erreichen der Ziele im Mittelpunkt steht.
– CMMI fordert umfangreiche Dokumentation, während Agilität leichtgewichtige Dokumentation erwartet.

Ein hoher Reifegrad beeinflusst die Arbeitsweise und birgt daher eine Wechselwirkung mit dem Modell der Arbeitsteilung und des Software-Entwicklungsprozesses.

Neben CMMI sind aus meiner Erfahrung weitere Rahmenwerke bekannt, die einen ähnlich lenkenden Effekt haben. Ist etwa Application Management nach ITIL im Unternehmen breit institutionalisiert, existiert ein Spannungsfeld zwischen Release Management, Change Management und agilen Methoden. Change Management nach ITIL fordert eine sorgfältige Planung, Prüfung, Freigabe und Kontrolle von Änderungen in den IT-Systemen. Agile Methoden forcieren die rasche (unbürokratische) Bereitstellung von Funktionalitäten.

Unterliegt ein Unternehmen dem Sarbanes-Oxley-Act oder dem URÄG 2008 ist ein internes Kontrollsystem IKS im Rahmen der IT-Governance aufzubauen. Dieses Kontrollsystem folgt meist dem CObIT-Rahmenwerk [The IT Governance Institute (2005)]. Eine Ausprägung davon ist Strategic Alignment. Strategic Alignment fordert, dass die Leistungen der IT auf die Anforderungen des Unternehmens abgestimmt sind. Dies wird durch ein Demand Management erreicht, jedoch müssen diese Demands formal, das heißt in geeigneter Weise dokumentiert, freigegeben und durch Wirtschaftsprüfer nachvollziehbar sein.

4.2.8 Definition von Projektstandards

Ob nun Kunde und Lieferant eine „reife“ Organisation besitzen, Prozesse generell standardisiert verfolgen und überwachen kann für Offshoring hilfreich sein und ist die Basis für das Festsetzen von Projektstandards. Darin wird geregelt, wie die Zusammenarbeit konkret aussieht, indem Richtlinien vereinbart werden. Zu allererst ist hier die Wahl eines geeigneten Software-Entwicklungsprozesses zu nennen. In weiterer Folge muss auf einzelne Phasen, Aktivitäten oder Ähnliches herunter gebrochen und Verantwortlichkeiten definiert werden.

Folgend beispielhaft Fragen, die hier erörtert werden:

- Welche Methoden zur Erhebung von Anforderungen und System Design werden eingesetzt?
- Welches Schätzverfahren zur Aufwandsabschätzung soll eingesetzt werden?
- Welche Schnittstellen (im Sinne von Übergabepunkten) zwischen Kunde und Lieferant werden gewählt?
- Welche Programmier- und Testrichtlinien sollen eingehalten werden?
- Welche Infrastruktur wird verwendet und wo ist diese lokalisiert?
- Wie sieht die Integrationsstrategie aus?
- Wie wird mit Change Request umgegangen?
- Welche Artefakte werden erstellt?
- Wie sieht das Configuration Management aus?
- Welche Werkzeuge werden eingesetzt (Modellierungswerkzeuge, Repositories, Versionsverwaltung)?
- Wie ist die Verteilung der Rollen und Verantwortlichkeiten (RACI)

Basierend auf einem Assessment der eigenen Organisationsstruktur bezüglich des Reifegrads und der eigenen gelebten Prozesse lässt sich die dritte Frage ableiten:

Die Frage lautet: Welche Prozesse und Methoden sollen angewendet werden?

4.3 Ausgewählte Kollaborationsmodelle unter Berücksichtigung der Erfolgsfaktoren

4.3.1 Extended Workbench

Bei der „Extended Workbench“ wird der Anbieter erst spät in den Prozess eingegliedert. Im Extremfall erst beginnend mit der Implementierung bzw. Codierung. Der Lieferant erhält vom Auftraggeber einen fertig spezifizierten Programmentwurf, während der Auftraggeber fertige Module oder Komponenten des Systems bereitgestellt bekommt. Normalerweise führt der Lieferant Modul- oder Komponenten-Tests durch, die Integration kann vom Auftraggeber durchgeführt werden.

Frage: Welche Skills sind im Unternehmen verfügbar?

In erster Linie steht diese Variante jenen Unternehmen zur Verfügung, die über eine eigene IT-Abteilung verfügen und sämtliche Aktivitäten der Kategorien Software Specification und Software Design abdecken können. Finanziell wirkt sich diese Variante

eher gering aus und wird oftmals zur Spitzenabdeckung bei fehlenden internen Ressourcen eingesetzt. Dies wird insbesondere bei größeren Unternehmen der Fall sein, die im Bereich Lieferantenmanagement strukturelle Vorteile aufweisen. Auch als Einstieg in die Sourcing-Welt ist die Extended Workbench gut geeignet, um dem Sourcing-Partner die Umwelten begreiflich zu machen und einen Wissenstransfer sicher zu stellen. Unterstützend kann Job Rotation unter den Entwicklern helfen, kulturelle Barrieren zu mildern.

Frage: Welche Prozesse und Methoden sollen angewendet werden?

Als Projektorganisation bietet sich ein Setup mit Projektleiter und Projektteam beim Kunden und ein schlankes Schattenprojekt beim Lieferanten an. Dabei geht die Gesamtprojektsteuerung vom Kunden aus, im Projektteam befinden sich sämtliche Rollen, wie bei einem internen Projektablauf üblich. Das sind Business Analysts, Software Architects, System Designer, Tester und eventuell Software Developer, falls der Offshoring Partner lediglich zur Skalierung dient. Der Lieferant stellt ebenfalls einen Projektleiter, der die Entwickler auf Lieferantenseite koordiniert.

Durch die späte Einbindung des Lieferanten zu Implementierungsbeginn kommen nur sequentielle Software-Entwicklungsmodelle zum Einsatz. Es wurde bis zu diesem Zeitpunkt bereits einiges an Vorarbeit geleistet, somit sind agile Methoden nicht mehr effizient einsetzbar.

Ist die Prozessfähigkeit bei beiden hoch und eine Standardisierung weit fortgeschritten, müssen lediglich die Schnittstellen gemeinsam definiert werden. In weiterer Folge werden Projektstandards festgelegt, insbesondere Codierungsrichtlinien und die Form erforderlicher Dokumentation. Dies orientiert sich meist an den Prozessen beim Kunden.

Ist die Prozessfähigkeit beim Kunden niedriger als beim Offshore-Dienstleister, sollten vor Projektstart (Ramp-Up-Phase) Erfahrungen des Dienstleisters beim Kunden einfließen. Dabei hat das Unternehmen die Chance vom Lieferanten zu profitieren und für weitere Projekte zu lernen.

Durch die starke Steuerung des Projekts beim Auftraggeber kommen prinzipiell alle Projekttypen, vom kleinen Webauftritt bis zur komplexen betrieblichen Anwendung in Betracht.

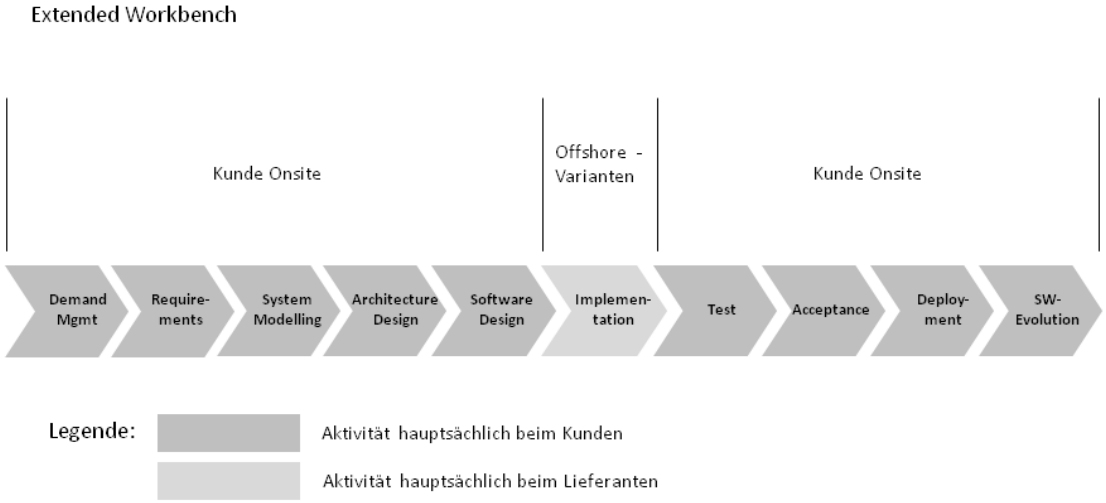


Abbildung 11: Extended Workbench

4.3.2 Over the Fence

Beim zweiten Extrem „Over the Fence“ wirft der Auftraggeber im wahrsten Sinne des Wortes seine Anforderungen an ein System über den Zaun zum Lieferanten. Der Lieferant ist für die komplette Projektabwicklung vom Systementwurf bis zum Deployment und nach Abschluss des Projekts für die Wartung verantwortlich. Vor dem Deployment erfolgt unter Anwendung von Acceptance-Tests beim Auftraggeber die formelle Abnahme der Software.

Frage: Welche Skills sind im Unternehmen verfügbar?

Prinzipiell werden im auftraggebenden Unternehmen bis auf eine Ausnahme keine besonderen Kenntnisse in Bezug auf Softwareentwicklung benötigt. Um einen Projekterfolg zu gewährleisten müssen die Anforderungen aber in einer hohen Güte vorliegen. Deshalb sind auf Kundenseite Kenntnisse im Bereich Business Analyse vonnöten. Die Konzeption und Umsetzung des Projekts liegt dennoch gänzlich in der Hand des Lieferanten. Man hat keinen Einfluss auf die Architektur des Systems und deren Umsetzung. Dies erschwert jedenfalls einen Wechsel des Anbieters, da das gesamte Wissen nicht im eigenen Unternehmen liegt. Eine solche Software wäre einem Legacy-System gleichzusetzen und sollte wenn möglich vermeiden werden.

Da beinahe alle Aktivitäten ausgelagert sind, scheint es so, als wäre dies finanziell die beste Lösung. Allerdings ist der Erfolg des Projekts äußerst fraglich. Nicklisch bestätigt

die Auffassung, dass „gerade bei innovativen Zielen der klassische und meistens zu optimistische – Anforderungen über den Zaun, fertige Software zurück – Ansatz nicht mehr funktioniert“ [Nicklisch, G. et al. (2009), 38].

Unternehmen ohne eigene IT sollten bereits mit inländischen Lieferanten Erfahrungen gesammelt haben. Die Komplexität steigt mit Offshore-Anbietern gewöhnlich stark an.

Somit beschränkt sich die Verwendung auf kleinere, wenig komplexe Anwendungen oder die Integration und Konfiguration von Standardsoftware.

Jene Unternehmen mit eigener IT und der Fähigkeit generell Software-Projekte gänzlich durchzuführen, können diese Form ebenfalls anwenden. Voraussetzung ist, dass sich die Kunde-Lieferanten Beziehung in einem eingeschwungenen Zustand befindet. Der Lieferant kennt genau den Geschäftsbereich in dem der Kunde operiert. Architektur- und Codierungsrichtlinien sind bekannt und der Kunde fordert diese auch ein. Beim Kunden wird die ausgelieferte Software auch inhaltlich verstanden, ein Wissenstransfer vom Lieferanten zum Auftraggeber findet statt. Anhand der vereinbarten Richtlinien bekommt man die oben angeführten Probleme bezüglich Architekturkonformität und Wissensmängeln beim Auftraggeber in den Griff. Was bleibt ist eine gewisse Unsicherheit über die Auslegung einzelner Punkte in den Requirements, die allzu oft in ermüdenden Diskussionen über kostenpflichtige Change Requests enden. Was bleibt ist ein schaler Nachgeschmack in der Beziehung zwischen Kunde und Lieferant.

Frage: Welche Prozesse und Methoden sollen angewendet werden?

Die Projektorganisation beinhaltet einen Projektmanager beim Kunden und einen Gegenpart beim Lieferanten. Der Projektmanager des Lieferanten steuert sein Team aus Business Analysts, Software Architects, System Designer, Developer und Tester. Dem Projektmanager auf Kundenseite stehen idealerweise bei Bedarf Experten zur Seite, die zwecks Einhaltung von Richtlinien und für Verständnisfragen partiell zur Verfügung stehen. Gibt es keine IT beim Auftraggeber, die diese temporären Rollen einnehmen kann, wird es wie eingangs beschrieben zu Problemen kommen.

Bei der Wahl des Entwicklungsmodells gibt es bei „Over the Fence“ Einschränkungen in Bezug auf agile Ansätze. Beim Einsatz agiler Methoden kommt es darauf an, wie hoch die Freiheitsgrade für den Lieferanten sind und ob der Kunde gewillt ist über die Projektlaufzeit ständig für Fragen und Entscheidungen bereit zu stehen. Darüber hinaus wäre ein Rückschritt in die Aktivität Requirements Engineering nötig (Stichwort: User

Stories, Priorisierung der Anforderungen). Ein Rückschritt kann eine Projektverzögerung bedeuten, birgt aber auch die Chance, dringend benötigte Funktionalitäten früher nutzen zu können. Dieser Ansatz wird unter „Over the Fence light“ betrachtet.

Beim Grad der Prozessfähigkeit gilt es zu beachten, dass für kleinere Unternehmen bei diesem Projektsetup in Kombination mit einem renommierten IT-Dienstleister ein Ungleichgewicht besteht. Dabei stehen wenigen Mitarbeitern auf Kundenseite (im Extremfall nur der Projektmanager) eine Vielzahl von Spezialisten beim IT-Dienstleister gegenüber [Gadatsch, A. (2006), 34].

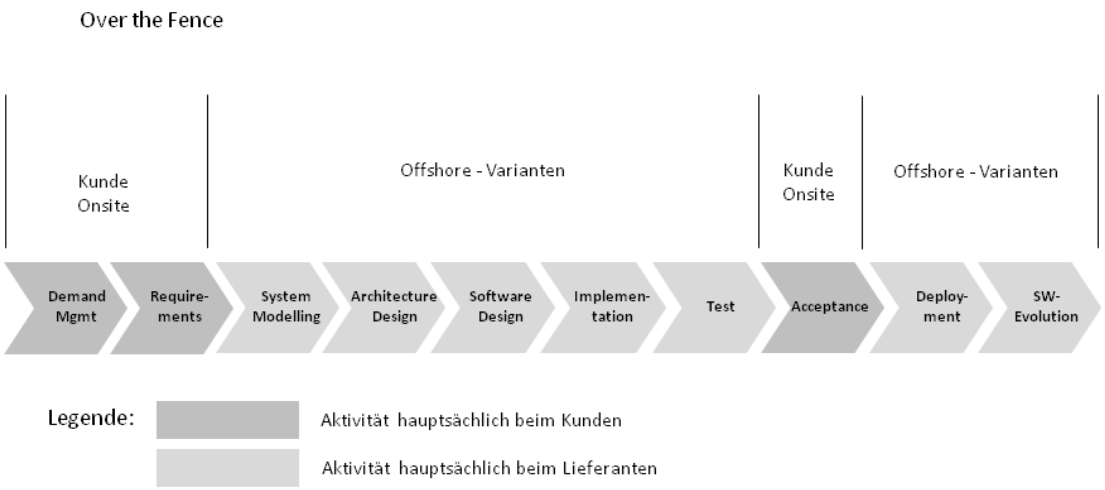


Abbildung 12: Over the Fence

4.3.3 Variante: Erweiterte Extended Workbench

Dabei übernimmt der Sourcing-Partner neben der Implementierung auch den Programmentwurf, das Software Design. Das Architecture Design wird also noch vom Kunden durchgeführt. Das macht vor allem dann Sinn, wenn der Kunde durch eine komplexe Systemlandschaft geprägt ist und er aus strategischer Sicht architekturelle Entscheidungen im Haus belassen möchte.

Frage: Welche Skills sind im Unternehmen verfügbar?

Hier gilt ähnliches wie bei der Extended Workbench, das Unternehmen sollte eine gut organisierte IT-Abteilung besitzen. Nachdem zwischen den Sourcing-Partnern Vertrauen durch erfolgreich abgewickelte Projekte hergestellt wurde, kann diese Form eine

logische Fortführung der Offshoring-Bestrebungen sein. Der Sourcing-Dienstleister konnte bereits Erfahrung sammeln und ist bereit sein Angebot auszudehnen.

Frage: Welche Prozesse und Methoden sollen angewendet werden?

Es bietet sich an, die Projektorganisation zu spiegeln. Das bedeutet, beim Kunden und beim Anbieter sind die gleichen Rollen vertreten, allerdings mit teils unterschiedlichen Aufgaben. Die Spiegelung hilft bei Verständnisproblemen Lösungen zu finden, dabei hat jeweils ein Part die Ausführungs-, der andere Part die Beratungskompetenz.

Sequentielle Modelle können jedenfalls zur Anwendung kommen, iterativ, inkrementelle Modelle bis zu agilen Methoden sind allerdings auch anwendbar. Die Frage hierbei ist nur, in welcher Granularität die Aktivitäten bis zum „Übergabepunkt“ bereits durchgeführt worden sind. Dabei sind die Anforderungen grob modelliert und das Architektur-Design fungiert lediglich als Vorgabe in dessen Rahmen die Umsetzung geschehen soll.

Die erforderliche Prozessfähigkeit ist auf beiden Seiten als hoch einzustufen. Jedenfalls sollten zu hohe Unterschiede zwischen den Vertragspartnern vermieden und ein diszipliniertes, strukturiertes Vorgehen angestrebt werden.

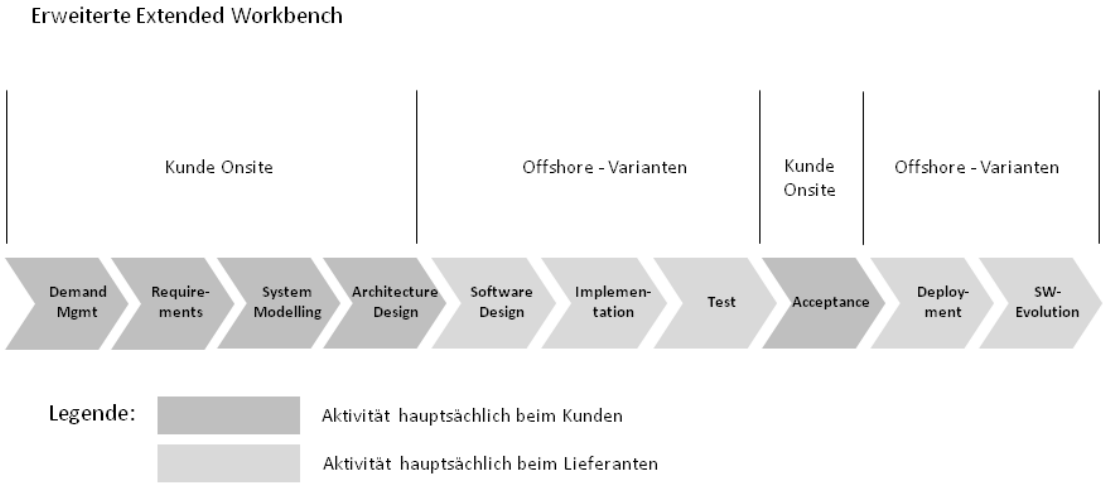


Abbildung 13: Erweiterte Extended Workbench

4.3.4 Variante: Over the Fence light

Der Schritt Requirements Engineering wird vom Kunden und Lieferanten gemeinsam durchgeführt. Im Vergleich zu Over the Fence werden Verständnis- und Auslegungsdefizite bereits zu Beginn bekämpft. Der übrige Software-Engineering Prozess wird wieder vom Lieferanten durchgeführt.

Frage: Welche Skills sind im Unternehmen verfügbar?

Prinzipiell sind keine Software-Engineering Kenntnisse auf Auftraggeberseite nötig. Der Lieferant führt durch den Requirements Engineering Prozess und gewährleistet durch seine hohe Durchführungskompetenz eine ordnungsgemäße Abwicklung des Projekts.

Geeignet für Unternehmen ohne eigene IT, die im Bereich Lieferantenmanagement einiges an Erfahrung mitbringen. Unternehmen mit eigenem IT-Bereich steht die Variante auch zur Verfügung. Kritisch anzumerken ist die Tatsache, dass die Business Analyse aus strategischer Sicht generell eine Stärke des Auftraggebers sein sollte. Ein Lieferant kann nur bedingt das Geschäftsfeld und die gelebten Prozesse im Unternehmen kennen. Diese Sicht muss in hohem Maße vom Kunden eingebracht werden. Demzufolge wäre es für einen Projekterfolg maßgeblich, wenn beim Kunden diese Rolle wahrgenommen werden kann.

Frage: Welche Prozesse und Methoden sollen angewendet werden?

Als Projektorganisation kommt ein ähnliches Setup wie bei „Over the Fence“ zum Tragen. Unterschied ist die zusätzliche Rolle Business Analyst beim Lieferanten.

Dieses Vorgehen eignet sich hervorragend für iterativ, inkrementelle Prozessmodelle und agile Methoden, da der Lieferant sehr früh im Prozess mitwirkt. Bei Anwendung von agilen Methoden sollte auf Kundenseite ein Nutzer des Systems für die Projektlaufzeit bereit gestellt werden, der bezüglich der Anforderungen Entscheidungen treffen kann und darf.

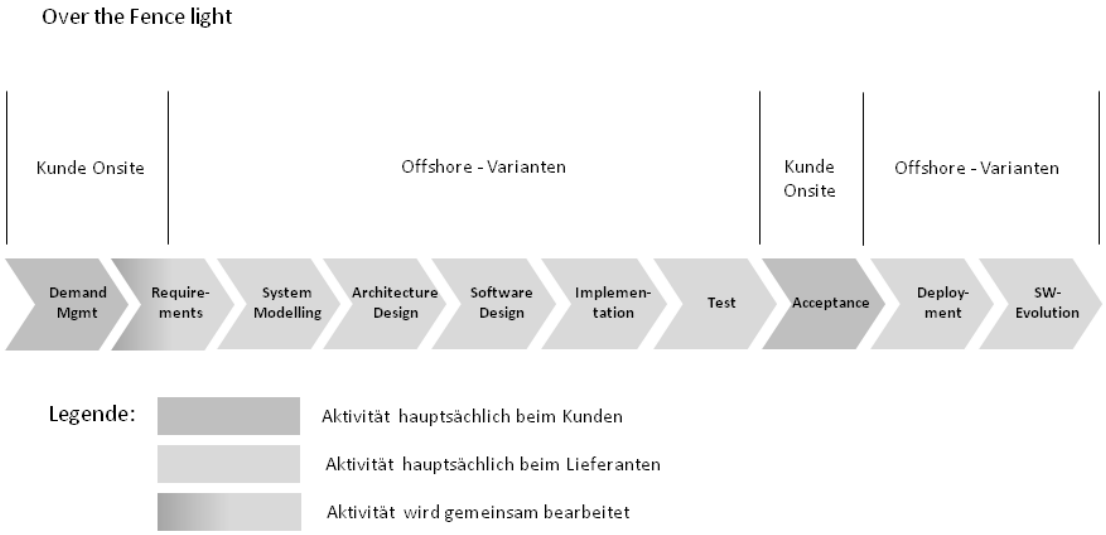


Abbildung 14: Over the Fence light

4.3.5 Setup im agilen Umfeld

Wie im agilen Manifest beschrieben streben alle agilen Vorgehensweisen nach sich selbst organisierenden Teams. Eines der zwölf Prinzipien lautet: “The best architectures, requirements, and designs emerge from self-organizing teams.” [Cunningham, W. et.al (2001)]. Mit agilen Methoden sollen Funktionalitäten in kurzen Iterationszyklen bereitgestellt werden. Demzufolge macht es Sinn in einem Team sämtliche Rollen vertreten zu haben, um dies zu erreichen. Dabei orientiert man sich weniger an den Aktivitäten, sondern an den Funktionalitäten. Solche Teams werden Featureteams genannt und beinhalten neben den Rollen Business Analyst, Software Architect, System Designer, Software Developer und Tester auch Domänen-Experten wie Datenbankspezialisten, User Interface Designer, Infrastrukturexperten und ähnliches mehr. Die Zusammenstellung des Teams orientiert sich dabei an den Rollen, die gerade eben für die Bereitstellung eines Features benötigt werden (vgl. [Eckstein, J. (2009), 33]). Die Zusammenstellung kann also je nach Feature variieren. Aus Sourcing Sicht ist die Verteilung Kunde-Lieferant eher zweitrangig, wichtig dabei ist der Teamgedanke.

Frage: Welche Skills sind im Unternehmen verfügbar?

Beispielhaft wird eine Variante beschrieben, bei der Kunde und Lieferant gemeinsam jede Aktivität im klassischen Sinne im Projekt bearbeiten, die Domänenexperten stellt der Kunde bereit. Implizit wird unterstellt, dass beim Kunden eine IT-Abteilung mit der Fähigkeit Software-Entwicklungen unter der Nutzung agiler Methoden existiert. Der

andere Fall, bei dem der Kunde über keine ausreichende IT verfügt ist unter „Over the Fence light“ zusammengefasst. Dabei führt der Lieferant durch das Projekt.

Die Schwierigkeit im Lieferantenmanagement liegt im Spannungsfeld zwischen vertraglichen Vereinbarungen und dem partnerschaftlichen Teamgedanken. Demzufolge sollte der Auftraggeber danach trachten, Vertragsklauseln nicht auf Punkt und Beistrich einzufordern, sondern dem Lieferanten Freiheiten zuzugestehen. Für den Fall, dass in einem Projekt auf die Vertragsgrundlage referenziert werden muss, ist wahrscheinlich bereits so viel schief gelaufen, dass der Teamgedanke schwer wieder herzustellen sein wird. Es wird darauf hingewiesen, dass herkömmliche Vertragsarten wie Fixed Price Contracts oder Time & Material einem guten Klima im Projekt abträglich sind.

Frage: Welche Prozesse und Methoden sollen angewendet werden?

Das Projektteam besteht aus einzelnen Featureteams, die an den Funktionalitäten des Systems ausgerichtet sind. Die Disziplinen des Software Engineerings werden mehr oder weniger gemeinsam durchgeführt. In den ersten Iterationen macht es Sinn, die Projektorganisation zu spiegeln, um das Teamgefüge zu etablieren und einen Wissenstransfer sicher zu stellen. In den folgenden Iterationen können je nach strategischer Ausrichtung aus Sourcing-Sicht budgetintensive Tätigkeiten mehr und mehr zum Sourcing-Partner verlagert werden. Beispielsweise findet eine Verschiebung der Aktivität Software Design und Implementierung zum Sourcing Partner statt, während Business Analyse und die Architektur weiterhin stärker vom Kunden betreut werden. In diesem Projektumfeld werden strenge Richtlinien des Kunden existieren, wie das System gebaut werden muss. Zu Einhaltung dieser Richtlinien bietet sich eine Rochade von Teammitgliedern des Kunden gegen Mitarbeiter des Vendors an. Somit bleibt der Wissensstand innerhalb der Teams konstant hoch. Spezialthemen wie User Interface-Design, Datenbank-Fragen, Netzwerkthemen und ähnliches werden vom Kunden betreut. Spezialisten auf diesen Gebieten können bei Bedarf in ein Team für die Iteration gezogen werden. Es besteht auch die Möglichkeit separate Teams mit Kenntnissen zu einer Domäne zu bilden und „on demand“ den Featureteams bereit zu stellen. In agilen Modellen kommt selten die Rolle des Projektmanagers vor. Je höher die Komplexität und das ist im Sourcing-Umfeld mit verteilten Teams jedenfalls der Fall, macht es Sinn diese Rolle (beim Kunden) zu besetzen (vgl. [Eckstein, J. (2009), 44]). Auf Lieferantenseite wird auch formal ein Projektleiter bestellt, der für die Belange seiner Mitarbeiter verantwortlich ist.

Standardisierte und dokumentierte Prozesse stehen vermeintlich im Widerspruch zu agilen Werten. Ist die Prozessfähigkeit bei beiden Partnern hoch, erfolgt in der Ramp-Up-Phase ein Mapping der agilen Vorgehensweise mit verpflichtend einzuhaltenden Prozessen auf Kundenseite. Im Projekt selbst muss darauf Wert gelegt werden, dass diesen Prozessen Genüge getan wird. Dies erfordert trotz aller Agilität eine disziplinierte Vorgehensweise.

Die Komplexität agil durchgeführter Offshoring-Projekte sollte nicht allzu hoch sein. Die Durchführung kritischer Projekte sollte auf diese Weise vermieden werden. Für intern genutzte Support-Systeme mit geringer Auswirkung im Fehlerfall ist diese Vorgehensweise gut geeignet, für Systeme mit erhöhter Außenwirksamkeit, wie Verrechnungs-Systeme, sollte lieber eine andere Art der Abwicklung gewählt werden.

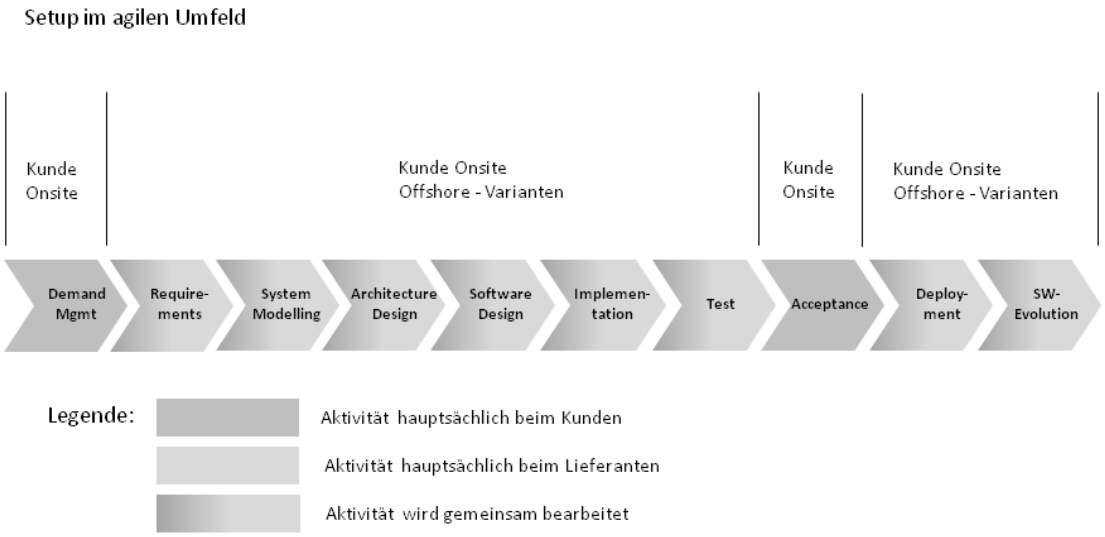


Abbildung 15: Setup im agilen Umfeld

5 Software-Engineering

In diesem Kapitel beleuchten wir Aspekte im Rahmen von Software Engineering, im speziellen die unterschiedlichen Disziplinen der Software-Erstellung. Diese Disziplinen finden sich mehr oder weniger deutlich herausgebildet in Software-Entwicklungsmodellen wieder. Wir wollen anhand von Beschreibungen in der Literatur diese Modelle auf ihre Praxistauglichkeit im Offshoring überprüfen.

5.1 Einführung

Obwohl es mir fernliegt, die gesamte Geschichte der Softwaretechnik aufzuarbeiten, erscheint es wesentlich, einige grundlegende Anmerkungen dazu festzuhalten. Bis in die späten 1960er Jahre hinein wurde Software-Entwicklung durch zwei essentielle Schritte charakterisiert, eine Art Analyse und die folgende Codierung. Darauf folgte wiederum die Analyse, ob die gewünschten Funktionalitäten korrekt implementiert werden konnten, wenn nicht, wurde wieder codiert. Diese Vorgehensweise, die sich auf den Kern der Entwicklung beschränken, wird „Code and Fix“ genannt, weil nach der Programmierung eine Fehlerkorrektur (Bug Fixing) folgt. Diese Vorgehensweise stieß an ihre Grenzen, als Programme umfangreicher und Anforderungen daran komplexer wurden. Außerdem war die Anwendung für den internen Gebrauch beschränkt, die Entwickler mussten genau wissen, was benötigt wird.

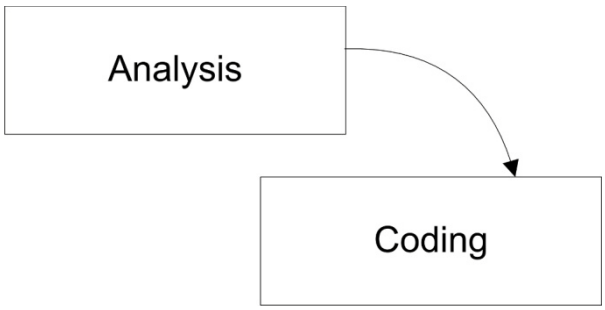


Abbildung 16: Code and Fix [Royce, W. (1970), 1]

In [Ludewig, J. et al. (2010), 153] werden Vor- und Nachteile dieser Vorgehensweise gelistet.

Vorteile:

- Das Vorgehen entspricht unserem Drang schnell voranzukommen und das uns gestellte Problem zu lösen.
- Die Arbeit liefert schnell ein lauffähiges Programm.
- Die auszuführenden Tätigkeiten, Codieren und spontanes Testen, sind relativ einfach.

Nachteile:

- Das Projekt ist nicht planbar, weil nie rational entschieden wurde, was in welcher Qualität hergestellt werden soll.
- Die Arbeiten können nicht über mehrere Personen oder Gruppen verteilt werden.
- Da die Anforderungen nicht systematisch erhoben wurden, werden sie vom Resultat in der Regel nicht erfüllt.
- Allen Prüfungen fehlen die Soll-Vorgaben.
- Die entstehenden Programme sind meist schlecht strukturiert und nur mit größtem Aufwand zu warten.
- Der Aufwand für Korrekturen ist unangemessen hoch, da Mängel erst spät (nämlich im Einsatz) entdeckt werden.
- Wichtige Konzepte und Entscheidungen sind nicht dokumentiert, sondern lediglich in den Köpfen der Entwickler vorhanden. Dieses Wissen kann kaum transferiert werden.

Software Engineering bedeutet ein systematisches, diszipliniertes und quantifizierbares Vorgehen zur Entwicklung, Betrieb und Wartung von Software (vgl. [IEEE Std 610.12 (1990), 67]).

Je nach Art der Software stehen eine oder mehrere Charakteristika im Vordergrund [Sommerville, I. (2011), 7]:

- Maintainability: Software muss so gebaut werden, dass sie zum Beispiel im Fehlerfall (Incident) einfach zu warten und bei weiterführenden Anforderungen der Fachbereiche leicht zu erweitern ist.
- Dependability: Software muss zu einem gewissen Grad zuverlässig und sicher sein. Sie muss angemessenen Schutz gegen bösartigen Zugriff bieten.
- Efficiency: Software soll nicht nur effektiv Funktionalitäten bereitstellen, sondern auch eine ausreichende Performance bieten. Dies gilt in Bezug auf Reaktionszeiten, Speicherbelegung und ähnliches.
- Acceptability: Software muss für User verständlich, leicht zu bedienen und mit anderer Software kompatibel sein.

Ungeachtet einer Verwertbarkeit für Offshoring, ist leicht zu erkennen, dass in der heutigen Zeit Code & Fix nicht mehr zeitgemäß ist. Eine gute Wartbarkeit verlangt etwa, dass vorab eine Architektur gewählt wird, in der Änderungen leicht durchführbar sind. Zuverlässige Software lässt sich jedenfalls besser in Umgebungen bewerkstelligen, die gut strukturiert und dokumentiert wurden.

Mit zunehmender Komplexität der Software, steigt auch die Notwendigkeit nach mehr oder weniger stark ausgeprägten, dem eigentlichen Programmieren vorgelagerten Aktivitäten. Diese Aktivitäten wurden ab ca. 1970 in Vorgehensmodellen, oder auch Software Entwicklungsprozessen zusammengefasst, welche die Abfolge dieser Aktivitäten in Relation setzen. Je nach Modell verlagert sich der Aufwand vor und nach der eigentlichen Codierung. Bei Code and Fix fließt scheinbar 100% in die Codierung. Dies ist natürlich nicht ganz korrekt, zumal man nur durch (unstrukturiertes) Testen der Funktionalität Fehler findet. Für phasenorientierte Modelle ist eine Verteilung von 40/20/40 (vor/für/nach Codierung) gegeben, bei modernen Modellen wird der Aufwand weiter nach vorne verlagert. Tätigkeiten zur Spezifizierung der Software steigen, der Aufwand für Tests sinkt (60/15/25) (vgl. [Ludewig, J. et al. (2010), 61]; Anm.: „Moderne Modelle sind nicht näher definiert).

5.2 Aktivitäten im Software- Engineering

Alle bisher veröffentlichten Software-Entwicklungsmodelle beinhalten zumindest vier Aktivitäten, die in einem Software-Projekt durchgeführt werden müssen [Sommerville, I. (2011), 36].

- Software specification
- Software design and implementation
- Software validation
- Software evolution

5.2.1 Software specification

In der Literatur finden sich mehrere Begrifflichkeiten zu diesem Thema. Im englischen Sprachgebrauch ist wohl Requirements Engineering am Gängigsten, im Deutschen Anforderungsanalyse, Analyse und Spezifikation der Software. Im Begriff Requirements

Engineering steckt bereits die Notwendigkeit, Anforderungen an ein System erarbeiten zu müssen. In den wenigsten Fällen liegen diese klar auf der Hand.

Sommerville beschreibt vier Aktivitäten, die den Requirements Management Prozess im Kern ausmachen [Sommerville, I. (2011), 37 und 82ff.]:

Feasibility study

Anhand von initial vorliegenden Anforderungen an ein System wird eine Entscheidungsgrundlage erarbeitet, ob mit dem Projekt fortgefahren werden soll oder nicht. Entscheidungskriterien sind hierbei wirtschaftliche Interessen (Business Case) und technologische Grenzen.

Requirements elicitation and analysis

Anforderungen an ein System können durch verschiedene Methoden erhoben werden. Gibt es bereits eine Art Dokumentation durch den Kunden, wird dies eine Basis dafür sein. In diesem Stadium ist es unbedingt nötig, mit allen Stakeholdern zu sprechen, mit ihnen Szenarien durchzuspielen, wie dies etwa im Extreme Programming durch User Stories geschieht. Use-Case-Diagramme im Rahmen von UML sind eine weit verbreitete Methode, um Interaktionen grafisch mit Usern (Actors) darzustellen. Existiert ein System beim Kunden, das wenigstens Teile der Anforderungen abdeckt, muss dieses ebenfalls analysiert werden. Es stellt neben den gelebten Geschäftsprozessen einen Teil des Istzustandes dar. Gewöhnlich sprechen User im Interview von den Dingen, die in der bestehenden Applikation schlecht sind und erwähnen nicht, welche Funktionalitäten durch das bestehende System gut abgedeckt sind [Ludewig, J. et al. (2010), 358]. Abhängigkeiten zu Systemen, die mit dem neu zu entwickelnden System in Verbindung stehen, werden dadurch aufgedeckt. Es ist daher notwendig den Istzustand zu erheben, bevor der Sollzustand eruiert wird.

Requirements specification

Darunter wird die Dokumentation von Anforderungen verstanden, ungeachtet dessen auf welche Art dies geschieht. In klassischen Entwicklungsmodellen werden meist ein oder mehrere Dokumente verfasst, oder durch CASE-Werkzeuge in Repositories abgelegt, während bei agilen Methoden Anforderungen auf Kärtchen geschrieben werden. User Requirements sollen stets so verfasst werden, dass funktionale und nicht funktionale Anforderungen von allen Stakeholdern verstanden werden können. System Requi-

rements beinhalten zusätzlich Informationen für System Engineers, wie die Anforderungen vom System bereitgestellt werden sollen. Eine deutsche Umschreibung dafür wäre Lastenheft und Pflichtenheft, letzteres käme bei einem sequentiellen Entwicklungsmodell als Vertragsbestandteil in Betracht.

Requirements validation

Die Spezifikation wird gegen die realen Anforderungen des Kunden geprüft. Dies geschieht unter anderem durch Reviews, Prototyping bzw. bei agilen Methoden oft durch Definition von Testfällen für jede Funktionalität (TDD, Test Driven Development). Eine Spezifikation ist idealerweise inhaltlich den Erwartungen des Kunden entsprechend, vollständig, widerspruchsfrei und objektivierbar, das heißt, es ist möglich gegen die Spezifikation zu testen [Ludewig, J. et al. (2010), 375]. Das Attribut vollständig wird bei iterativen, inkrementellen Modellen und insbesondere bei agilen Methoden erst spät im Prozess erreicht.

5.2.2 Software design and implementation

Aktivitäten im Rahmen von Software Design übertragen die Ergebnisse der Software Spezifikation in eine strukturierte technische Abbildung des zu bildenden Systems. Werden iterative oder inkrementelle Entwicklungsmodelle angewandt, geschieht dies im Wechselspiel mit der Entwicklung der Software Spezifikation. Agile Methoden verzichten oft auf die Erstellung eines Dokuments und binden das Software Design in den Code mit ein. Als kreative Tätigkeit wird ein stabiles Software Design nicht in einem Wurf erreicht, der Software Analyst oder Designer wird mehrere Schleifen in seine Arbeit einbinden, um alle Abhängigkeiten optimal abzubilden. Je nach Art der Software werden in unterschiedlichem Ausmaß folgende Aktivitäten durchgeführt:

Architectural Design

Sommerville unterscheidet zwei Abstraktionsebenen „architecture in the large and architecture in the small“[Sommerville, I. (2011), 148]. Architecture in the large betrachtet die Interaktion im Rahmen der komplexen Unternehmensarchitektur, wenn die Software oder Teile davon verteilt bereitgestellt werden. Architecture in the small befasst sich mit dem Aufbau des zu entwickelnden Programmes selbst. Welche Komponenten und Module sollen implementiert werden und wie stehen sie in Relation zueinander.

Interface Design

Komponenten müssen miteinander kommunizieren können, dafür werden Interfaces definiert. Schnittstellen sind Grenzen zwischen zwei kommunizierenden Komponenten. Die Schnittstelle stellt Aktionen der Komponente für ihre Umgebung zur Verfügung und umgekehrt [Ludewig, J. et al. (2010), 406].

Component Design

Festlegen des Verhaltens einzelner Komponenten und Module ist Ziel dieses Schritts. Das Ausmaß reicht von der Darstellung einer Konfiguration bereits existierender Komponenten (component reuse) bis zum detaillierten Design Modell neu zu entwickelnder Teile.

Database Design

Hier wird festgelegt, ob und welches Datenbankmodell zum Einsatz kommen soll, z.B. relational oder objektorientiert, und wie die Daten strukturiert werden sollen.

5.2.3 Software validation

Aktivitäten rund um das Testen von Software beziehen sich auf die Verifizierung und Validierung. Bei der Verifizierung wird gegen die Software Spezifikationen geprüft, ob Funktionalitäten richtig implementiert worden sind. Die Validierung bezieht sich darauf, ob das System auch die Erwartungen der Kunden erfüllt. Testen beginnt somit eigentlich bereits im Stadium vom Requirements Engineering, in dem überprüft wird, ob die geforderten Funktionalitäten auch dementsprechend in der Spezifikation festgehalten und weiter, ob im Software Design die Requirements Specification richtig gedeutet wurde. Sommerville zitiert Barry Boehm: „Validation: Are we building the right product?“, “Verification: Are we building the product right?” [Sommerville, I. (2011), 207].

Die eigentlichen Softwaretests im Rahmen des Developments werden nach ihrer Komplexität unterteilt in [Sommerville, I. (2011), 210ff]:

Unittests

Bei Unittests werden singuläre Einheiten wie einzelne Funktionen oder Objektklassen getestet. Alle möglichen Eingabeparameter, Ausgabewerte oder Statusveränderungen sind Ziel dieser Tests.

Componenttests

Hierbei werden einzelne Objekte, “Units” zu Komponenten zusammengefasst, also integriert und das Verhalten der gesamten Komponente geprüft. Die Tests werden gegen das Interface der Komponente gerichtet und nicht mehr auf die einzelnen Objekte. Es darf davon ausgegangen werden, dass die Objekte die Unittests positiv durchlaufen haben. Es handelt sich also um Integrationstests.

Systemtests

Ein lauffähiges Softwaresystem zu erstellen ist Ziel dieser Tests. Dabei werden die einzelnen Komponenten integriert. Im Vergleich zu Componenttests gibt es zwei Unterschiede. Beim Zusammenstellen des Systems werden neu entwickelte Komponenten möglicherweise mit zugekauften (COTS – commercial off the shelf) und wiederverwendeten Komponenten integriert und Komponenten, entwickelt von verschiedenen Teams, werden integriert.

5.2.4 Software evolution

Dieser Schritt ist in den meisten Entwicklungsmodellen wenig beachtet, wird er doch mit Betrieb, Wartung, Maintenance gleichgesetzt. Software Projekte werden in den seltensten Fällen von Null aufgebaut, sondern es wird auf Bestehendem aufgesetzt, Systeme umgebaut und erweitert. Prinzipiell können drei Typen unterschieden werden [Sommerville, I. (2011), 243]:

– Bug fixing: Das System arbeitet nicht mehr wie erwartet und der Fehler muss behoben werden.
– Adaptierungen aufgrund sich ändernder technischer Rahmenbedingungen: Das System Environment hat sich geändert, dadurch muss die Software adaptiert werden, um lauffähig zu bleiben.
– Änderungen aufgrund neuer Anforderungen: Der Fachbereich benötigt neue oder modifizierte Funktionalitäten.

Es ist ratsam Software evolution als kontinuierliche Verbesserung in den Lebenszyklus mit einzubeziehen. Die drei Typen können nach ihrer Dringlichkeit klassifiziert werden. Grundsätzlich ist die Frage bei Bug Fixes und neuen User Requirements, ob die Änderungen außerhalb von geplanten Wartungsfenstern in eine Produktivumgebung eingespielt werden sollen. Da Änderungen der Systemumwelt planbar sind, werden Changes

dieser Art, als auch weniger kritische Bug Fixes und User Requirements release-basiert abgewickelt werden können.

6 Software-Entwicklungsmodelle

Ein Software-Entwicklungsprozess gibt den Rahmen vor, wie Softwareprojekte abgewickelt werden sollen. Den Kern eines jeden Modells bilden die oben angeführten Disziplinen. Sie regeln die Abfolge von Aktivitäten ausgehend von einem Bedürfnis des Kunden bis zum fertigen Produkt.

Ein Software- Entwicklungsprozess ist nach dem IEEE Standard Glossary of Software Engineering Terminology demnach [IEEE Std 610.12 (1990), 67]:

“The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. Note: These activities may overlap or be performed iteratively.”

6.1 Chronologische Einordnung

Code & Fix stand am Anfang der Software Prozess-Evolution mit zwei einfachen, aufeinander folgenden Aktivitäten. Im Laufe der Zeit sind eine Vielzahl an Modellen in der Literatur niedergeschrieben worden, einige wenige sind oder waren flächendeckend in der Software-Industrie im Einsatz.

Auf der Suche nach einer chronologischen Aufstellung wurde ich beim US-Amerikaner Mark Kannaley fündig, der in seinem Buch eine Vision zur künftigen Software Entwicklung erörtert.

Um einen Einblick in die Evolution zu gewinnen folgt eine vereinfachte und um die europäische „V-Modell“ -Prozessfamilie erweiterte Darstellung der Entwicklung gängiger Modelle [Kannaley, M. (2010), 18].

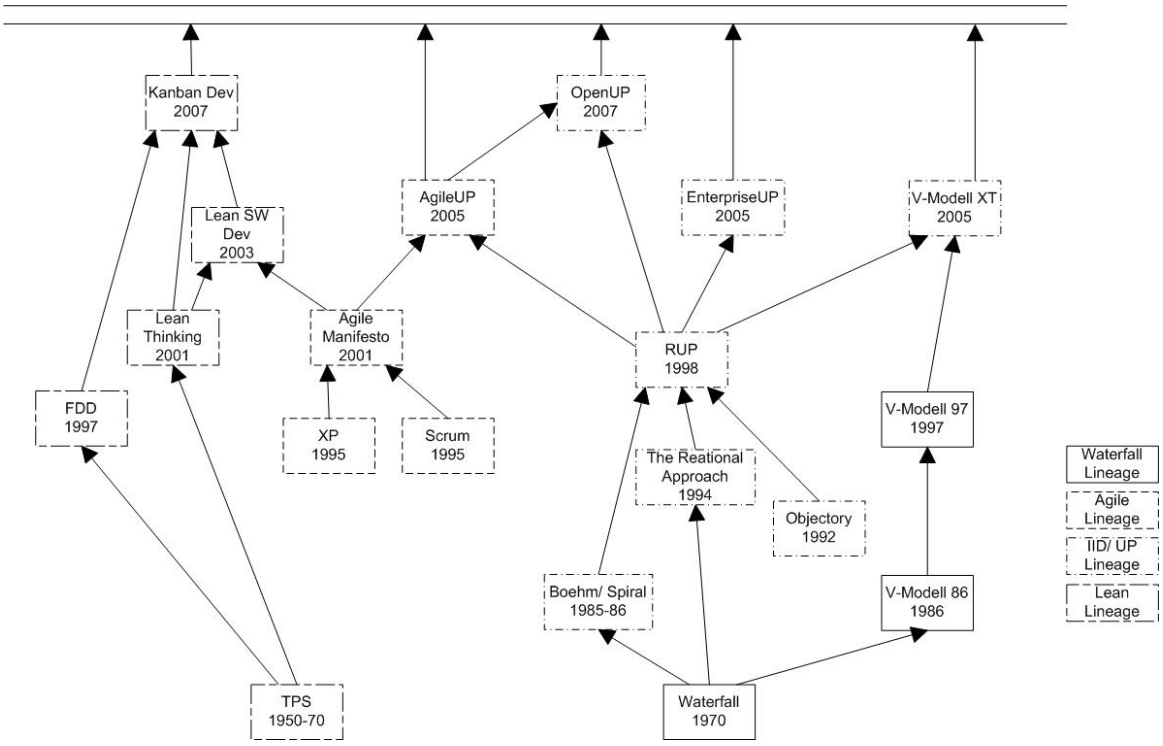


Abbildung 17: History of modern Software Engineering Methods [in Anlehnung an: Kannaley, M. (2010), 18]

Kannaley ortet vier nahezu isolierte Stoßrichtungen, nämlich den Wasserfall, die iterativ, inkrementelle Modellfamilie (Iterative and Incremental Development, IID und Unified Process, UP), Agile Software Development und Lean Software Development. Unter SDLC 1.0 (Software Development Life Cycle 1.0) fallen der Wasserfall und somit auch das V-Modell, unter SDLC 2.0 interessanterweise sämtliche iterativ, inkrementellen Modelle inklusive Agile Development und Lean Development. Das Ziel wäre aus seiner Sicht eine Baseline aller Praktiken, um die Erkenntnisse „Best of Bread“ zu SDLC 3.0 zusammen zu fassen. Er argumentiert, dass die Vielzahl der Modelle nicht automatisch aufgrund von Bedürfnissen aus Projekten heraus entwickelt wurden, sondern oftmals kommerzielle Interessen der Industrie zu Weiterentwicklungen derselben geführt haben (vgl. [Kannaley, M. (2010), 17ff.]. Der interessierte Leser mag unter anderem das Spiralmodell vermissen. Es handelt sich hier nach gängiger Lehre um ein Metamodell und scheint deshalb in meiner Darstellung nicht auf [Ludewig, J. et al. (2010), 177].

6.2 Grundlegende Differenzierung

6.2.1 Vorgehensmodell – Prozessmodell

Beide Begriffe werden in der Literatur synonym verwendet. Ein Vorgehensmodell beschreibt die Aktivitäten, die während eines Softwareentwicklungsprojekts abgehandelt werden sollen. Prozessmodelle gehen einige Schritte weiter und definieren zusätzlich Aspekte um handelnde Rollen, Projektmanagement, Qualitätsmanagement und ähnliches mehr. Das Wasserfallmodell und Phasenmodell gehören zu den Vorgehensmodellen, Unified Process und V- Modell hingegen wären zu den Prozessmodellen zu zählen (Anmerkung: das V-Modell wird offiziell als Vorgehensmodell bezeichnet) [Ludewig, J. et al. (2010), 91].

6.2.2 Lineares – nicht lineares Vorgehensmodell

Ein lineares Vorgehensmodell ist gekennzeichnet durch eine streng sequenzielle Abfolge von Aktivitäten oder Phasen. Am Ende des Prozesses steht die lauffähige Software, die mit einem sogenannten „Big Bang Approach“ gelauncht wird. Das bedeutet, dass erst am Ende des Projekts die erstellte Software in Betrieb genommen wird und diese der Kunde erst ab diesem Zeitpunkt nutzen kann.

Nichtlineare Modelle erlauben hingegen eine zyklische Abwicklung des Projekts. Nach Ludewig unterscheidet man folgende Ausprägungen [Ludewig, J. et al. (2010), 169ff]:

Rapid Prototyping

Hierbei wird eine Software, die im Groben der bereits vorhandenen Spezifikation entspricht, zur weiteren Klärung der Anforderungen gebaut. Der Prototyp sollte anschließend verworfen werden, da er den zugrunde liegenden Qualitätskriterien nicht entspricht.

Evolutionäre Entwicklung

Wiederum steht die Klärung von Anforderungen im Vordergrund. Im Unterschied zu oben wird die Software nicht verworfen, sondern so lange verfeinert, bis ein gebrauchsfähiges System vorhanden ist. Nachteil dieser Vorgehensweise ist eine vergleichsweise schlechte Architektur. Evolutionäre Entwicklung wird bei agilen Methoden angewandt. Schwächen in der Architektur können, falls vorgesehen, mittels speziell zur Bereini-

gung vorgesehenen Zyklen entgegnet werden (Anmerkung: Man spricht hier von Refactoring).

Iterative Entwicklung

Bei der iterativen Entwicklung wird ein umfangreiches Projekt in kleinere Zyklen unterteilt. Der Gesamtumfang wird dabei nicht aus dem Auge verloren, die Anzahl der Iterationen vorab definiert. In jeder Iteration wird analysiert, designt, implementiert und getestet. Nach jeder Iteration liegt eine verbesserte Version des zu entwickelnden Systems vor. Am Ende der letzten Iteration steht das fertige Produkt zur Verfügung.

Inkrementelle Entwicklung

Hier wird die Software in Ausbaustufen realisiert, wobei jede Ausbaustufe auch ausgerollt wird. Das erste Inkrement bietet dabei die absolut notwendigen Kernfunktionalitäten, während die folgenden Ausbaustufen zusätzliche Features bereitstellen. Meist ist der Endzustand des Produkts vorab nicht klar. Vergleichbar ist diese Vorgehensweise mit Release-basierter Entwicklung, ein umfangreiches Vorhaben wird in mehrere, oftmals noch nicht quantifizierbare Teilprojekte zerlegt.

Treppenmodell

Der inkrementellen Entwicklung sehr ähnlich, wird die Kernfunktionalität rasch bereit gestellt. Der Unterschied liegt in der verschachtelten Abwicklung der einzelnen „Teilprojekte“. Ist die Analyse der ersten Ausbaustufe abgeschlossen und kann mit dem System-Design und in weiterer Folge mit der Implementierung begonnen werden, beginnt parallel die Analyse der zweiten Ausbaustufe und so fort. Bedingung für diese Art der Entwicklung ist allerdings eine gesamthafte Planung, um die inhaltlichen Abhängigkeiten und terminliche Komplexität in den Griff zu bekommen.

6.2.3 Klassische – agile Prozessmodelle

Klassische Modelle orientieren sich an einer Abfolge von Aktivitäten oder Phasen. Am Ende einer Phase wird ein bestimmtes Set an Deliverables in einer definierten Güte geliefert. Dieser Output ist gleichzeitig ein Input für die folgenden Aktivitäten. Aktivitäten orientieren sich streng am Prozess, ein Abweichen davon ist nicht gewünscht oder wird nicht toleriert. Agile Modelle konzentrieren sich auf die Kernaktivitäten Design und Implementierung und bauen Tätigkeiten, die in der klassischen Vorgehensweise explizit

angeführt werden, in die Kernaktivitäten mit ein. Die Erstellung einer Spezifikation ist grundsätzlich Nebensache, sie kann aber im Nachhinein verfasst werden.

Anders gesagt werden im klassischen Modell innerhalb einer Aktivität mehrere Iterationen durchlaufen bis der Reifegrad für einen Abschluss einer Aktivität oder Phase gegeben ist. Bei agilen Modellen werden die Iterationen über mehrere Schritte hinweg und in kleineren Inkrementen durchgeführt [Sommerville, I. (2011), 62].

Agile Prozesse entstanden als Gegenbewegung zu den als starr und unflexibel erlebten, umfangreichen Prozessmodellen. Im ersten Moment der Frustration auf Entwicklerseite wurde alles bisher Etablierte als schlecht dargestellt. Insbesondere Vorgehensweisen, die nicht zwingend zum Projekterfolg beitragen sollten, standen in der Kritik. Daraus resultierte eine Ablehnung von jedweder Dokumentation, die nicht unmittelbar mit dem übergeordneten Ziel, der lauffähigen Software, zu tun hatten.

Im oft zitierten Agilen Manifest konnten sich Jahre später führende US-Experten auf Grundsätze der agilen Softwareentwicklung einigen, die nicht mehr ganz der extremen Haltung entsprach, sondern vielmehr ein neuartiges Wertesystem verknüpft mit 12 Prinzipien vermitteln sollte [vgl. Cunningham, W. et.al (2001)].

Hruschka beschreibt das agile Wertesystem folgendermaßen [Hruschka, P. et al. (2009), 3ff.]:

eher angemessen als extrem

Wie wir im Kapitel 7.3 XP - Extreme Programming noch sehen werden, gilt der Grundsatz „im Source-Code ist genügend Dokumentation enthalten“ für die wenigsten Projekte. Für jedes Projekt soll sorgfältig abgewogen werden, welche Art der Dokumentation zweckmäßig ist. Wie viel zusätzlicher Aufwand neben der eigentlichen Entwicklung soll betrieben werden und welche Vorteile werden dadurch erreicht? Ein Aspekt der durchaus auf klassische Modelle umgelegt werden kann. Wenn es das Rahmenwerk ermöglicht, könnten für kleinere Projekte nur relevante Bausteine aus einem Bauchladen verwendet werden, ohne Gefahr zu laufen in einer Sackgasse zu landen. Arbeitsprodukte müssten entweder gar nicht oder nur in geringerem Ausmaß durchgeführt werden.

eher ergebnisorientiert als prozessorientiert

Oberstes Projektziel ist die Herstellung einer lauffähigen, nutzenbringenden Software. Gewisse Rahmenbedingungen erfordern zusätzliche Aktivitäten, um geforderte Ergebnisse in der Zielehierarchie zu befriedigen.

Bei nachhaltig entwickelten Systemen mit längerer Laufzeit empfiehlt es sich daher Dokumentation in höherer Qualität bereit zu stellen und von der Prämisse, Wissen alleine im Team zu verteilen, abzuweichen. Dies gilt auch für größere Projekte, wenn es darum geht, Informationen über mehrere Teams oder Standorte zu verteilen. Im Falle von Kunde-Lieferant Beziehungen ist es unumgänglich ein Mindestmaß an Dokumentation sicher zu stellen. Schließlich gilt es die Ergebnisse auf Basis einer vertraglichen Grundlage zu bewerten.

eher Best Practices aus Erfahrung als verordnete Vorgaben

Die Welt ist komplex. So empfiehlt es sich, nicht starr einem Vorgehensmodell zu folgen, sondern auf erprobte Methoden zurück zu greifen, die in ähnlichen Konstellationen zum Erfolg geführt haben. Es kommt stark auf die Gegebenheiten im Unternehmen an, welche Wege zur Zielerreichung frei gegeben und welche strikt zu befolgen sind. Starke Teams finden ihren optimalen Arbeitsstil, der möglicherweise nicht auf andere übertragbar ist. Sie finden Best Practices, die sie regelmäßig hinterfragen und mit denen sie sich identifizieren. Zentral vorgegebene Praktiken bringen unter Umständen nicht die Akzeptanz wie selbst definierte.

eher miteinander reden als gegeneinander schreiben

Persönliche Kommunikation ist dem Austausch von Dokumenten vorzuziehen. Wenn immer es geht, sollten sich Projektbeteiligte miteinander unterhalten und offene Fragestellungen erörtern. Selbst wenn es gelingen sollte, formal korrekt und leicht verständlich Sachverhalte nieder zu schreiben, lässt sich im persönlichen Gespräch vermeintlich Klares weiter präzisieren und hilft Irrtümer zu reduzieren. In vielen Modellen wird dies durch ein Projekt-Setup bewerkstelligt, in dem alle Beteiligten an einem Ort, im radikalen Fall in einem Raum sitzen. Dies beinhaltet auch Benutzer des Systems, die permanent Rede und Antwort stehen sollen.

eher offen für Änderungen als starres Festhalten an Plänen

Dies umfasst zwei Aspekte. Einerseits die Nutzung einer iterativen, inkrementellen Vorgehensweise, allerdings verkürzte Iterationszyklen im Vergleich zu herkömmlichen Modellen. Die Auslieferung lauffähiger Funktionalitäten in kurzen Intervallen bietet den Mehrwert früher und schafft Transparenz über den Projektfortschritt. Dies unabhängig davon, ob der Benutzer davon sofort profitiert, oder „nur" die Auftraggeber beruhigt werden sollen. Andererseits begünstigt es eine rasche Aufnahme von geänderten und neuen Anforderungen in jeder Iteration. Um den Fokus nicht aus den Augen zu verlieren, empfiehlt sich eine grobe Planung über die gesamte Projektlaufzeit hinweg.

eher Menschen und Kommunikation als Prozesse und Tools, eher Vertrauen als Kontrolle

Projektteammitglieder sollen ermutigt werden alternative Wege zu gehen, solange keine zwingenden Regulative dagegen sprechen und Best Practices einfließen zu lassen, anstatt strikten Vorschriften nachzugehen. Die Kontrolle soll in das Team verlagert werden, anstatt auf formale Kontrollmechanismen wie Fortschrittsberichte zu bestehen. Ein vertrauensvoller Umgang unter den Vertragspartnern ist wichtiger, als stetig auf jede Klausel auf Punkt und Beistrich zu verweisen. Voraussetzung dafür ist eine bedarfsgerechte Ausgestaltung der Verträge, die erst dann präzisiert werden, wenn beispielsweise eine neue Iteration ansteht.

6.2.4 Schwere – leichte Prozessmodelle

Getrieben durch die agile Bewegung gelangte die Unterscheidung zwischen schweren und leichten Prozessen in den Sprachgebrauch. Einerseits zielt die Unterscheidung auf die Art ab, wie ausführlich oder komplex in Bezug auf seine Ausprägungen ein Modell beschrieben wurde. Andererseits werden alle dokumentenbasierenden Modelle mit „schwer" assoziiert. „Ein Prozess, der sich selbst dient und nicht dem übergeordneten Ziel, mit minimalem Aufwand qualitativ ausreichende Software zu bereitzustellen, hat mehr mit Kafka als mit den traditionellen Ingenieurswerten zu tun" [Ludewig, J. et al. (2010), 218]. Diese Aussage beschreibt treffend den Overhead in Projekten, wenn strikt nach Vorschrift auf Basis von umfassenden Prozessmodellen gearbeitet wird. Allerdings kann sie nicht auf alle Projekte umgelegt werden. Man neigt dazu agile Werte dazu zu verwenden, um den Dokumentationsaufwand von schweren Prozessen zu umgehen. Es kommt auf das Vorhaben an, welche Art der Dokumentation zweckmäßig ist.

Die Devise lautet: So wenig Dokumentation wie möglich, so viel Dokumentation als nötig. Beispielsweise wird für hoch sichere und hoch verfügbare Systeme eine andere Herangehensweise zu wählen sein, als für schnelllebige Internetapplikationen.

7 Ausgewählte Software-Entwicklungsmodelle

Im folgenden Abschnitt werden einige Vorgehens- bzw. Prozessmodelle beschreiben, die aus meiner Sicht häufig in der Praxis angewendet werden. Bei der Auswahl wurde darauf Wert gelegt, dass jeweils ein sinnvoller Vertreter der erwähnten Differenzierungsarten beschrieben ist.

Die Wahl fiel auf das Wasserfallmodell als einen Vertreter der klassischen, streng sequentiellen Vorgehensmodelle. Um die nächste Evolutionsstufe der Prozessmodelle abzudecken, wird in weiterer Folge der Unified Process beschrieben. Diese Auswahl hat zum Ziel, die nicht linearen Modelle in die Überlegungen mit einzubeziehen. Wie wir später sehen werden, deckt der UP neben iterativ, inkrementeller Entwicklung auch das Phasenmodell und prototypische Entwicklung mit ab. Es bestand die Möglichkeit, anstatt dem Unified Process als generisches Modell, das konkretisierte Prozessmodell RUP (Rational Unified Process) zu durchleuchten. Der RUP beschreibt den Unified Process nur detaillierter und baut ihn an den Stellen, an denen es von den Autoren für notwendig erachtet wurde, weiter aus. Der Grundgedanke ist im Unified Process ausreichend beschrieben, außerdem erschwert die Granularität des RUP ein Nebeneinanderstellen der Modelle.

Als Vertreter der agilen, leichtgewichtigen Modelle fiel die Wahl einerseits auf Extreme Programming, andererseits auf Scrum. Extreme Programming beschreibt agile Entwicklungspraktiken, während Scrum den Fokus auf Projektmanagementmethoden legt. In der Praxis werden beide Ansätze oftmals zusammen angewendet, deshalb finden in dieser Arbeit zwei agile Methoden ihren Platz. Dies darf allerdings nicht als Wertung für agile und gegen andere Methoden gesehen werden.

Aufgrund der Komplexitätsreduktion wurde auf die Familie des Lean Software Development verzichtet. Das V-Modell wurde erst zuletzt ausgeschieden, da es in seiner ursprünglichen Ausgestaltung dem Wasserfall vom Ablauf her sehr nahe kommt. Größter Unterschied ist wohl die Tatsache, dass im V-Modell der Validierung und Verifikation ein hoher Stellenwert beigemessen wird. Jeder Aktivität im Software Engineering wird eine Aktivität der Kategorie Test gegenübergestellt. Dadurch entsteht eine V-förmige Darstellung, die dem Modell seinen Namen gibt. Die Aktivität „Testen“ zieht sich wie ein roter Faden durch das Modell. Es gibt sehr detaillierte Vorgaben zu Ergebnistypen

und Rollen, es erinnert vom Umfang an RUP. Dadurch ist das V-Modell insbesondere für große Projekte und Embedded Systems geeignet, bei denen Verifizierung und Validierung einen hohen Stellenwert einnimmt [Bundesministerium für Verteidigung (1997)].

Im Rahmen der Diskussion ausgewählter Modelle wird der Bogen zu signifikanten Erfolgsfaktoren im Sourcing-Umfeld gespannt, um Erkenntnisse zu deren Anwendbarkeit im Offshoring-Kontext zu gewinnen. Die Ausführungen werden durch Anwendungsfälle ergänzt.

- Aussage zu Software Engineering Disziplinen: In Kapitel 3 Delivery-Modelle und Kapitel 4 Kollaborationsmodelle hat sich gezeigt, dass es für einen Projekterfolg auf die der Codierung vorgelagerten Aktivitäten ankommt, insbesondere auf Software Specification und etwas nachrangig Software Design. Deshalb wird darauf der Fokus gelegt.
- Kommunikation, Koordination, Steuerung: Eine Vielzahl der Erfolgsfaktoren im Sourcing Kontext beziehen sich auf diese wesentlichen Projektmanagement-Agenden. Zusätzlich wird auf Wissenstransfer und Change Request Handling eingegangen.
- Anwendungsfälle: Bezugsgrößen für eine Bewertung sind Projektgröße, Risiko und Komplexität des Vorhabens sowie Kriterien bezüglich der Projektart

7.1 Das Wasserfallmodell

Winston Royce beschrieb 1970 das wahrscheinlich bekannteste und heute trotz seiner Schwächen noch in Verwendung befindliche Modell [Royce, W. (1970), 1ff.].

Als Weiterentwicklung zu „Code & Fix“ streicht der Artikel drei wesentliche Punkte heraus [vgl. auch Royce, W. (1998), 6]:

- Software-Entwicklung beinhaltet zwei Basisaktivitäten, nämlich Analyse und Implementierung
- Zusätzlich sind einige vor- und nachgelagerte „Overhead“-Schritte nötig, um ein Software Projekt kontrolliert abhandeln zu können.
- Das beschriebene Modell ist besser als Code & Fix, minimiert aber nicht Risiken und begünstigt Fehler.

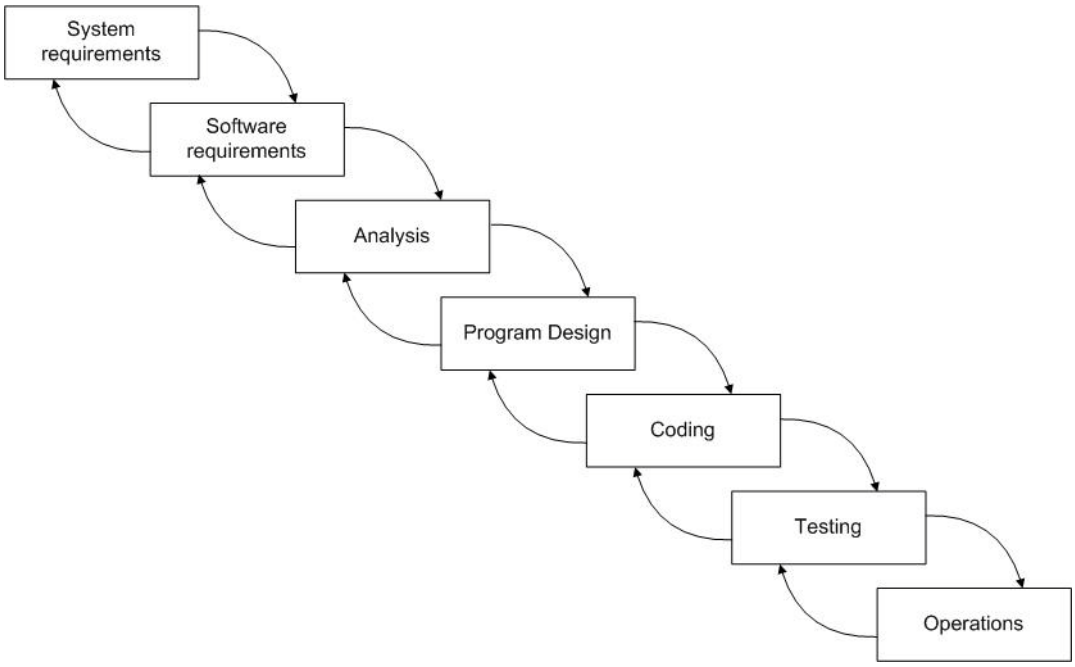


Abbildung 18: Wasserfallmodell [Royce, W. (1970), 2]

7.1.1 Eigenschaften

Das Wasserfallmodell gibt Aktivitäten in der Reihenfolge vor, in der sie durchzuführen sind. Es beginnt links oben und endet rechts unten. Für größere Projekte ist es unmöglich in einer der frühen Aktivitäten alle Eventualitäten zu berücksichtigen. Daher kann man bei Auftreten von Widersprüchen in die vorherigen Schritte zurückkehren. Im Prinzip stellen die angeführten Aktivitäten jene dar, die zumindest in einem Software Projekt behandelt werden müssen. In der Praxis wurde häufig das „strenge Wasserfall- oder Einbahnstraßenmodell“ verfolgt. Dabei wurde festgelegt, dass am Ende eines Schrittes jeweils eine formale Abnahme der Deliverables (Dokumente) erfolgen muss und die Inhalte ab diesem Zeitpunkt als fixiert anzusehen sind. Dadurch wurden die Rückschritte außer Kraft gesetzt, das Modell wurde dadurch unflexibler.

Zum Thema Risikominimierung bietet der Artikel von Winston Royce fünf Praktiken, welche die Risiken minimieren sollen (aber bei den Praktikern selten Anwendung fanden):

Program Design Comes First

Im Fokus steht der Schritt “Prelimenary Program Design” der nach dem Schritt „Software Requirements“ und vor dem Schritt „Analysis“ eingebettet werden sollte. Damit sollten Fehler aufgrund mangelhafter Architektur umgangen werden.

Document the design

Als dokumentengetriebenes Modell verlangt es nach umfangreichen Spezifikationen in Schriftform. Zum Zeitpunkt der Erstellung des Artikels ein adäquates Verlangen, da zu diesem Zeitpunkt noch keine ordentliche Werkzeugunterstützung und Methoden vorhanden waren. Kernaussage ist allerdings, dass komplexe Systeme ausreichend dokumentiert sein sollen, damit allen betroffenen Stakeholdern eine umfangreiche Wissensbasis zur Verfügung steht.

Do it twice

Wird unbekanntes Terrain betreten, schlägt Royce vor, den kompletten Zyklus zwei Mal zu durchlaufen. Im ersten Durchlauf ist die Aufgabe, grob ein System zu bauen, das dem endgültigen bereits im Wesentlichen entspricht. Ziel ist die gewählte Architektur und Alternativen auszutesten. Der Aufwand im Vergleich zum zweiten Durchlauf ist bewusst kurz zu wählen.

Plan, control and monitor testing

Software Validierung findet erst sehr spät im Prozess statt. Royce schlägt vier Punkte vor, nämlich ein unabhängiges Test-Team, Review des Source-Codes , Prüfen jedes logischen Pfades und Abnahmetest in der Produktivumgebung.

Involve the customer

Gefordert wird ein Einbinden des Kunden an verschiedenen Punkten im Prozess. Zuerst bei der Erstellung der Benutzeranforderungen im Schritt „System Requirements“, eine Abnahme des „Preliminary Program Design“, Reviews und Abnahme im Schritt „Program Design“ und ein „Final Acceptance“ am Ende des Schritts „Testing“ bevor die Software produktiv gesetzt wird.

7.1.2 Diskussion

Aussage zu Software Engineering Disziplinen

Das Modell ist anforderungsgetrieben, es wird im ersten Schritt versucht, sämtliche Requirements zu erfassen und zu definieren. In weiterer Folge werden genau diese implementiert. Es erfolgt grundsätzlich keine Priorisierung der Funktionalitäten, um die Architekturtreiber zu lokalisieren. Dadurch leidet der Programmaufbau [Royce, W. (1998), 15].

Im Projektverlauf werden vermeintlich durch abgenommene Dokumente Erfolge erzielt. Durch das Paradigma erst spät im Prozess mit der Codierung zu beginnen, kann es zu Problemen mit der Integration der einzelnen Teile und Interfaces kommen. Schwächen im Software Design werden möglicherweise erst jetzt erkannt und führen wegen des hohen Termindrucks zu suboptimalen Anpassungen (Abbildung 20). Als Resultat käme eine nicht stabile, schwer wartbare Architektur zu Tage [Royce, W. (1998), 12].

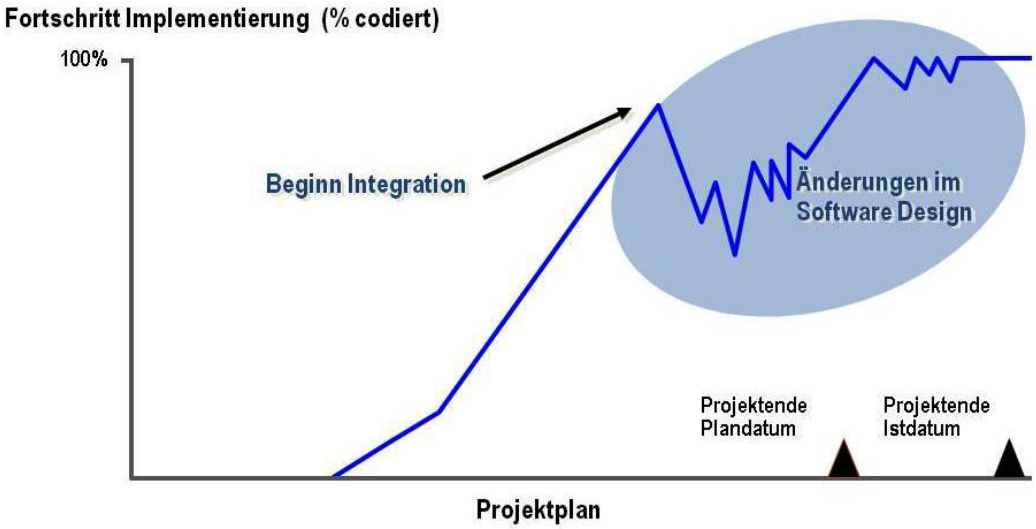


Abbildung 19: Fortschritt eines Projekts [Royce, W. (1998), 12]

Kommunikation, Koordination, Steuerung

Als reines Vorgehensmodell macht der Wasserfall keine Aussage über Projektmanagement-Aktivitäten.

Im strengen Einbahnstraßenmodell ist der Anforderer lediglich im Rahmen der Erhebung von Requirements und bei den Abnahmetests mit einbezogen. Die IT übernimmt die Durchführung aller verbleibenden Aktivitäten, streng nach Prozess, ohne weitere Einbindung des Kunden.

Im ursprünglich angedachten Modell findet Royce, wie oben erwähnt, mehrere Punkte in denen Kunden mit einbezogen werden sollen. Allerdings beschränkt sich dies meist auf Reviews von (umfangreichen) Dokumenten, deren Aussagekraft von zweifelhafter Gestalt ist.

In einer Kunde-Lieferanten Beziehung läuft es darauf hinaus, dass ein hohes Maß an Ressourcen zur Sichtung von Unmengen an Dokumentation bereitgestellt werden muss.

Durch die vertragliche Bindung ist ein Lieferant darauf bedacht sich abzusichern. Viel effektiver wäre es, den Fokus auf das Wesentliche zu lenken, die eigentliche Umsetzung. [Royce, W. (1998), 17].

Ein Risikomanagement ist schwer zu implementieren, da die Teilsysteme und somit ausführbarer Code zur Verifizierung und Validierung erst sehr spät zur Verfügung stehen. Das bedeutet eine kontrollierte Risiko-Steuerung ist erst während der Testphase möglich. Zuvor liegt das Gefährdungspotential allgemein auf einem sehr hohen Niveau [Royce, W. (1998), 14].

Zu der Zeit als Winston Royce das Wasserfallmodell entwickelte, gab es mit hoher Wahrscheinlichkeit keine verteilte Entwicklung. Um einen Wissenstransfer zu gewährleisten sind Dokumente grundsätzlich geeignet, sofern sie aktuell gehalten sind und einen angemessenen Umfang haben. Speziell neu ins Projekt zu integrierende Personen könnten davon profitieren. Es spricht außerdem nichts dagegen, werkzeugunterstützt eine Aktualität durch CASE-Tools zu gewährleisten. Hilfreich ist auch, wenn das Modell durch häufige und regelmäßige Reviews der erzielten Ergebnisse ergänzt wird.

Neue oder sich ändernde Anforderungen während eines laufenden Projekts sind durch das Wasserfallmodell grundsätzlich nicht abgedeckt. Der Output jeder Aktivität ist umfangreich und muss präzise formuliert sein. Jede Änderung an den Anforderungen bedingt eine Überarbeitung der Dokumente in den bereits beendeten Aktivitäten. Je später ein Change Request eingemeldet wird, desto höher ist das Risiko in Bezug auf Projektlaufzeit und Kosten.

Anwendungsfälle

Unter folgenden Bedingungen kann das Wasserfallmodell eingesetzt werden (eigen und in Anlehnung an [Sommerville, I. (2011), 63], [Thompson, K. (2012)], [Hastie, J. (2008)]):

- Die Anforderungen sind vorab in einer hohen Qualität klar definierbar und bleiben in hohem Maße konstant.
- Für reine Migrationsprojekte, bei der die funktionalen Anforderungen gleich bleiben
- Architektur und Infrastruktur ist wohl verstanden bzw. bleibt im Rahmen des Projekts konstant [Munassar, N. et al. (2010), 97].
- Systeme, die bereits häufig vom Lieferanten in ähnlicher Weise implementiert worden sind, wie ERP-Systeme und ähnliches mehr.

- für sicherheitskritische Anwendungen
- für hoch verlässliche Systeme, z.B. Kundenverrechnung
- Für Systeme, die hochgradig durch externe Regulatoren beeinflusst werden
- Geeignet für Projekte mit eher mittlerer bis hoher Kritikalität und mittlerem Umfang und geringer bis mittlerer Komplexität beziehungsweise für kleinere Anpassungen mit geringer Kritikalität, Umfang und Komplexität (Enhancements).

7.2 Der Unified Process

Mit der Objektorientierung begannen die Schwierigkeiten mit den bis dahin geltenden Paradigmen der sequentiellen Prozessmodelle. Viele konventionelle Methoden für Analyse und Design, Implementierung und Test mussten neu arrangiert werden.

Der Unified Process hat seine Wurzeln in Objectory (1987-1992) von Ivar Jacobson, einer objektorientierten Entwurfsmethode, die in den Objectory-Prozess, einem Use-Case getriebenen Vorgehensmodell mündete (1992). Nach dem Zusammenschluss von Objectory mit Rational wurde der „Rational Unified Process“, kurz RUP, vorgestellt (1998). Maßgeblich beteiligt daran waren Grady Booch, James Rumbaugh und Philippe Kruchten.

1999 wurde der „Unified Software Development Process“ von Jacobson, Booch und Rumbaugh vorgestellt, der wieder eine allgemeine Sicht darstellt. Der RUP ist eine konkrete Ausprägung des Unified Process [Ludewig, J. et al. (2010), 202].

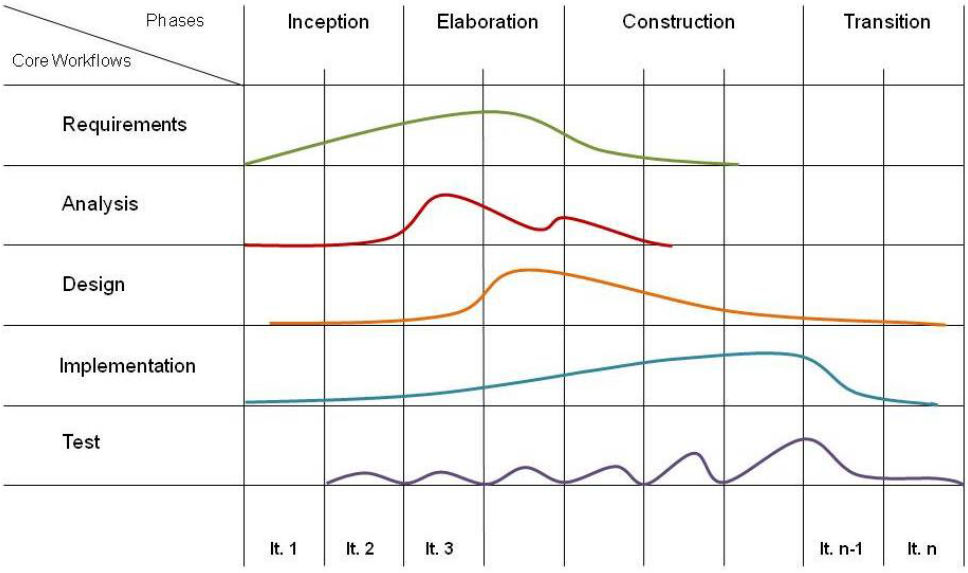


Abbildung 20: Unified Process [Jacobson, I. et al. (1999), 104]

7.2.1 Eigenschaften

Der Unified Process ist ein Phasenmodell mit vier definierten Abschnitten. Jede Phase endet mit einem Meilenstein. Den Durchlauf aller vier Phasen nennt man Zyklus. Am Ende eines Zyklus steht immer ein Release, egal ob sie internen Zwecken dient oder den Kunden zu Verfügung gestellt wird.

Der Unified Process ist iterativ und inkrementell in dem Sinn, als dass jede Phase in mehrere Iterationen unterteilt werden kann. In jeder Iteration werden alle nötigen Arbeitsschritte wiederholt durchlaufen. Als Ergebnis steht am Ende jeder Iteration ein ausführbares Stück Software. Die Schwerpunkte in den Iterationen sind unterschiedlich, ein Release entsteht inkrementell [Ludewig, J. et al. (2010), 203]. Der Unified Process unterstützt inkrementelle Entwicklung auch in der Hinsicht, dass ein Zyklus mehrmals durchlaufen werden kann [Sommerville, I. (2011), 51]. Am Ende des letzten Zyklus steht das fertige Produkt dem Kunden zur Verfügung.

Der Unified Process ist Use-Case getrieben und verwendet UML (Unified Modelling Language) zur Beschreibung von User Requirements. Als Resultat steht ein Use-Case Modell zur Verfügung, welches im Prozessverlauf die Basis für Design- und Implementation- Modelle darstellt. Letztendlich baut das Test-Modell ebenfalls auf den Use-Cases auf [Jacobson, I. et al. (1999), 34]).

Der Unified Process ist architekturzentriert, da der Aufbau des Programmes frühzeitig im Prozess mitbetrachtet und stetig weiterentwickelt wird. Jede Software wird durch Funktion und Form bestimmt, Use-Cases beschreiben die Funktion, die Architektur die Form. Eine Wechselwirkung besteht, da Use-Cases in die Architektur passen müssen und die Architektur genügend Raum für weitere Use-Cases bieten soll. Darum sollten Use-Case Erstellung und Architekturüberlegungen immer parallel entwickelt werden [Jacobson, I. et al. (1999), 6].

7.2.2 Phasen

In Abbildung 21 sind die vier Phasen des Unified Process ersichtlich, die absichtlich nicht nach Aktivitäten oder den Workflows benannt sind, da in jeder Phase Aktivitäten in unterschiedlicher Granularität durchgeführt werden. Inception und Elaboration bilden die Engineering Stage, Construction und Transition die Production Stage. [Royce, W. (1998), 74ff]

Beispielhaft und idealisiert sei der Erfüllungsgrad bei der Use-Case Erstellung über die Phasen hinweg in Abbildung 22 dargestellt.

Erfüllungsgrad in %	identifiziert	beschrieben	analysiert	designed, implementiert, getestet
Inception	50	10	5	3
Elaboration	>80	40-80	20-40	10
Construction	100	100	100	100
Transition				

Abbildung 21: Use Case Evolution (vgl. [Jacobson, I. et al. (1999), 358])

Inception Phase [Royce, W. (1998), 76], [Jacobson, I. et al. (1999), 358]

Übergeordnetes Ziel dieser Phase ist unter allen Stakeholdern eine gemeinsame Sicht auf das Projekt zu erarbeiten.

Dies beinhaltet:

- Project Scope, Aufbauen einer priorisierten Liste mit Requirements-Kandidaten (Feature List)
- Herausarbeiten der kritischen Requirements und damit der Use-Cases (Design-Treiber), eine erste Version des Use-Case Modells.
- Herausarbeiten von Architekturszenarios, die kritische Use-Cases abdecken
- Eventuell Implementierung eines Proof-of-Concept Prototyps
- Grobe Projektplanung für das gesamte Projekt, detaillierte Planung für die Elaborations-Phase
- Risikomanagement aufsetzen

Elaboration Phase [Royce, W. (1998), 78], [Jacobson, I. et al. (1999), 380]

Herausragendes Ziel in dieser Phase, die gleichzeitig das Ende der Engineering Stage bedeutet, ist das Festlegen der Zielarchitektur. Des Weiteren sollen die verbleibenden Anforderungen identifiziert worden sein, obgleich Änderungen in denselben nach wie vor zu einem bestimmten Grad möglich sein sollen.

Dies beinhaltet:

- Architektur Baseline, das heißt alle kritischen Use-Cases müssen in der gewählten Zielarchitektur abgebildet werden können. Requirement Changes dürfen keine gröberen Änderungen in der Architektur hervorrufen.
- Vision Baseline, ist ein Dokument für alle Stakeholder und beinhaltet u.a. Informationen über die Features mit Qualitätskriterien und Architektur, die schlussendlich in der Construction Phase umgesetzt werden. Es ist mit einem Lastenheft vergleichbar und kann als Basis für ein Fixpreisangebot gesehen werden.
- Grobe Projektplanung für das gesamte Projekt, detaillierte Planung für die Construction-Phase

Construction Phase [Royce, W. (1998), 79], [Jacobson, I. et al. (1999), 394]

Diese Phase stellt den Beginn der Production Stage dar, das Hauptziel der Construction Phase ist Implementierung, Integration und Test der noch nicht eingebauten Features.

Dies beinhaltet:

- Erreichen eines qualitativ hochwertigen, ausführbaren Systems (Beta-Release)
- Aktualisieren der unvollständigen Dokumentation
- detaillierte Planung für die Transition-Phase

Transition Phase [Royce, W. (1998), 80], [Jacobson, I. et al. (1999), 407]

Übergeordnetes Ziel der Transition ist die Überführung des Systems in die Umgebung des Kunden, der Nutzer des Systems.

Dies beinhaltet:

- Herstellen der finalen Konfiguration, die von den Stakeholdern abgenommen werden kann.
- Auslieferung wenn nötig mehrerer Releases unter Berücksichtigung von User-Feedback
- Iterationen beinhalten hauptsächlich Bug Fixes und Minor Enhancements
- Vervollständigen von User-Dokumentationen und Trainings

7.2.3 Core Workflows

Abbildung 21zeigt die Core Workflows oder Arbeitsabläufe des generischen Unified Process über alle Phasen hinweg.

Diese lauten:

- Requirements
- Analysis
- Design
- Implementation
- Test

Die Kurven verdeutlichen schematisch die Ausprägung der Workflows in der jeweiligen Phase bzw. Iteration.

Jeder Workflow wird durch eine Anzahl von Aktivitäten mit dazugehörenden Workern bestimmt. Ein Worker ist im Sprachgebrauch des Unified Process eine Rolle. Aufgrund der Überschneidung zu UML wurde dafür ein unterschiedlicher Begriff gewählt. Ein Worker kann mehrere Rollen einnehmen.

Abbildung 23 zeigt beispielhaft eine Darstellung des Requirement Worflows mit Aktivitäten und den dazu gehörenden Workern.

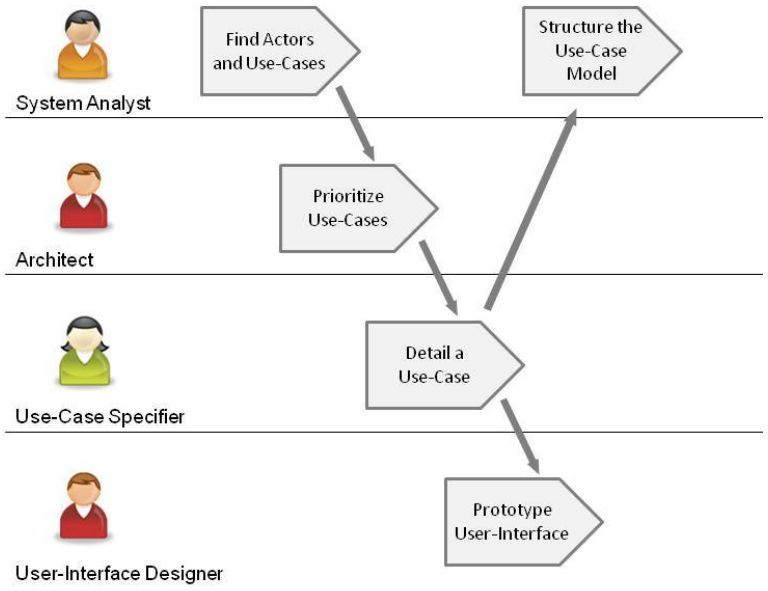


Abbildung 22: Requirements Workflow [Jacobson, I. et al. (1999), 143]

7.2.4 Diskussion

Aussage zu Software Engineering Disziplinen

Eng mit dem Auftraggeber werden Anforderungen iterativ und inkrementell strukturiert erhoben, priorisiert und als Use Cases visualisiert. Durch die Standardisierung der Dokumentation wird eine objektive Sicht auf die Anforderungen gewährleistet [Royce, W. (1998), 65]. Die Kenntnis der UML- Notation auf Kundenseite ist allerdings Voraussetzung für ein gemeinsames Erarbeiten der Requirements in dieser Form. Die Definition von Use Cases erfolgt vom Groben zum Feinen, ein Überarbeiten ist daher grundsätzlich erlaubt.
Als architekturzentriertes Modell gilt es in der Engineering Stage rasch einen stabilen Programmaufbau zu definieren. In der Inception Phase basierend auf den wichtigsten Use-Cases, in der Elaboration-Phase wird der Aufbau soweit fixiert, damit die Architektur als stabil und robust gelten kann [Jacobson, I. et al. (1999), 91]. Durch dieses Vorgehen wird das Risiko minimiert, spät im Projektverlauf das Software Design zu Ungunsten von Zeit und Kosten revidieren zu müssen.
Wichtigster Meilenstein in diesem Zusammenhang ist der Life-Cycle Architecture Milestone am Ende der Elaboration Phase.
Durch die iterativ, inkrementelle Vorgehensweise ist es dazu möglich im Projektteam, aber noch viel wichtiger, mit dem Kunden, eine einheitliche Sicht auf die Anforderungen und deren Umsetzung zu erhalten. Da zumindest am Ende jeder Iteration ein weiterer informeller Meilenstein (minor milestone) angesetzt wird, erhält der Auftraggeber frühzeitig Kenntnis über den Fortschritt in Form einer lauffähigen Systemumgebung.

Kommunikation, Koordination, Steuerung

Der iterative Life Cycle an sich minimiert Risiken gegenüber traditionellen, sequentiellen Methoden. Kritische Fragen werden frühzeitig im Prozess adressiert und nachhaltig gelöst.

Späten Änderungen im Software Design und damit verbundenes Rework der bereits implementierten Anforderungen werden ebenso begegnet, wie der Erstellung letztendlich inadäquater Requirements. Letzteres bezieht sich auf die Tatsache, dass eine Definition der Anforderungen in größeren Projekten vorab ohne ständige Überarbeitung kaum möglich ist.

Der UP fordert regelmäßig vorzeigbare Ergebnisse in Form von lauffähiger Software und fördert dadurch ebenfalls Verständnis für Requirements im Projektteam und zwischen den Stakeholdern, darüber hinaus zeigen sich frühzeitig Design-Fehler.

Zum Thema Qualitätsmanagement fordert Royce den Fortschritt und die Qualität für alle Stakeholder sichtbar zu machen und zwar speziell in Bezug auf Change Management [Royce, W. (1998), 187]. Bei iterativer Entwicklung geht es primär um die Möglichkeit Änderungen zuzulassen. Erst im zweiten Schritt sind absolute Qualitätskriterien maßgeblich. Positiv anzumerken ist die Tatsache, dass einfache Metriken definiert wurden, die mit entsprechender Werkzeugunterstützung automatisiert werden sollten.

Durch den hohen Grad an Standardisierung und die grafische Notation der Artefakte wird ein rascher Wissenstransfer in der Anlaufphase sowie während eines Projekts begünstigt. Die iterativ/inkrementelle Vorgehensweise in Verbindung mit häufigen Reviews zumindest am Ende der Iterationen ermöglichen zudem rasche Feedbackschleifen.

Change Requests werden grundsätzlich zugelassen bzw. gelten bis zum Abschluss der Elaboration-Phase nicht als solche. Einschränkungen dafür gibt es insbesondere in der Production Stage, wenn sie auf die Architektur massiven Einfluss haben.

Anwendungsfälle

Unter folgenden Bedingungen kann der UP eingesetzt werden (eigen und in Anlehnung an [Sommerville, I. (2011), 53], [Thompson, K. (2012)], [Hastie, J. (2008)]):

- Umfangreiche Projekte mit vielen inhaltlichen Unklarheiten
- Hoch innovative Projekte mit starker Geschäftsprozessorientierung
- Prestigeprojekte und wichtige Kundenprojekte
- Geeignet für Projekte mit hoher Kritikalität und Umfang bei gleichzeitiger hoher Komplexität beziehungsweise für mittlere Kritikalität, Umfang und Komplexität, wenn agile Methoden nicht angewendet werden können. Dies ist bei steigender Komplexität und verminderter Verfügbarkeit der handelnden Personen zum raschen Informationsaustausch gegeben.

7.3 XP - Extreme Programming

Im Kapitel 6.2 Grundlegende Differenzierung wurde die Unzufriedenheit mit herkömmlichen Vorgehensmodellen, die in der Entwicklung agiler Modelle mündete, bereits be-

schrieben. 1998 formulierte Kent Beck mit Extreme Programming einen aufgrund seiner Praxisrelevanz bedeutenden Vertreter dieser Gattung.

Beck beschreibt mit „Four Values“ die Werte, die Extreme Programming ausmachen. Dabei werden die Individuen in den Vordergrund gestellt und beeinflussen bei einer Einführung im Unternehmen dessen Wertesystem grundlegend.

Diese Werte lauten [Beck, K. (1999), 32ff]:

Communication

XP fordert von allen Beteiligten aktive und persönliche Kommunikation. Dies wird durch eine Vielzahl von XP-Praktiken (siehe unten) erreicht, die ohne direkte Kommunikation nicht funktionieren würden.

Simplicity

Einfachheit steht in allen Belangen im Vordergrund. Jede Funktionalität wird auf die einfachste Art und Weise implementiert. Vorausschauendes Handeln ist in dem Sinne nicht erwünscht, als dass eventuell in der Zukunft auftretende Änderungen in der Lösungsfindung keine Berücksichtigung finden. XP geht davon aus, dass das Einbeziehen von bekannten, aber erst später zu implementierenden Funktionalitäten vergeudete Zeit ist, da sich diese häufig noch ändern oder neue Funktionen höher priorisiert werden.

Feedback

Eng mit Kommunikation verbunden zielt Feedback über den gesamten Projektverlauf darauf ab, erzielte Ergebnisse den relevanten Stakeholdern zu vermitteln und eine Bewertung abzuholen. Beispielsweise erhält der Anforderer rasche Rückmeldung über den Aufwand neuer Anforderungen, der Programmierer gibt Feedback über die Güte von User Stories. Durch vorab definierte Testfälle erhält der Programmierer zudem Feedback, ob eine User Story verstanden wurde bzw. nach Implementierung durch Ausführen der Test-Cases, ob Fehler aufgetreten sind.

Courage

Die Teammitglieder sind aufgefordert auftretende Probleme kritisch zu hinterfragen und wenn nötig neue Wege zu gehen. Dies beinhaltet auch Wegwerfen von Code, wenn Refactoring keine Ergebnisse bringt. Offene Kommunikation erfordert Mut. Einem Auf-

traggeber sofort mitzuteilen, dass die Arbeit der letzten Tage verloren ist und dies Auswirkung auf das Releasedatum hat erfordert Mut.

Die vier Grundwerte hängen voneinander ab, etwa erfordert laufendes Feedback häufige Kommunikation und Einfachheit auch in gewissem Sinne Mut.
Im Kapitel 7.3.3 XP - Grundpraktiken lernen wir die 12 Grundpraktiken von XP kennen, den Kern des Modells. Diese „Practices“ waren in der Welt von Software Engineering nicht neu. Das Neuartige daran ist die Tatsache, dass bewährte Methoden auf „extreme“ Art und Weise durchgeführt werden sollen (vgl. [Beck, K. (1999), 7]):

- Permanente Code Reviews ersetzen mehrmalige (pair programming)
- Ständiges Testen ersetzt häufiges Testen (Unit Testing, Functional Testing)
- Design wird ständig hinterfragt und angepasst (Refactoring)
- Einfachheit bedeutet ein System mit einem Design, das gerade eben die bisherigen Funktionalitäten abbildet (Simple Design)
- Die Architektur des Systems wird ständig neu überarbeitet anstatt vorab definiert (Metaphor)
- Integration und Testen mehrmals täglich ersetzt konventionelle Integrationsstrategien (Continuous Integration)
- Extrem kurze Iterationszyklen ersetzen mehrwöchige, mehrmonatige Iterationen (the Planning Game)

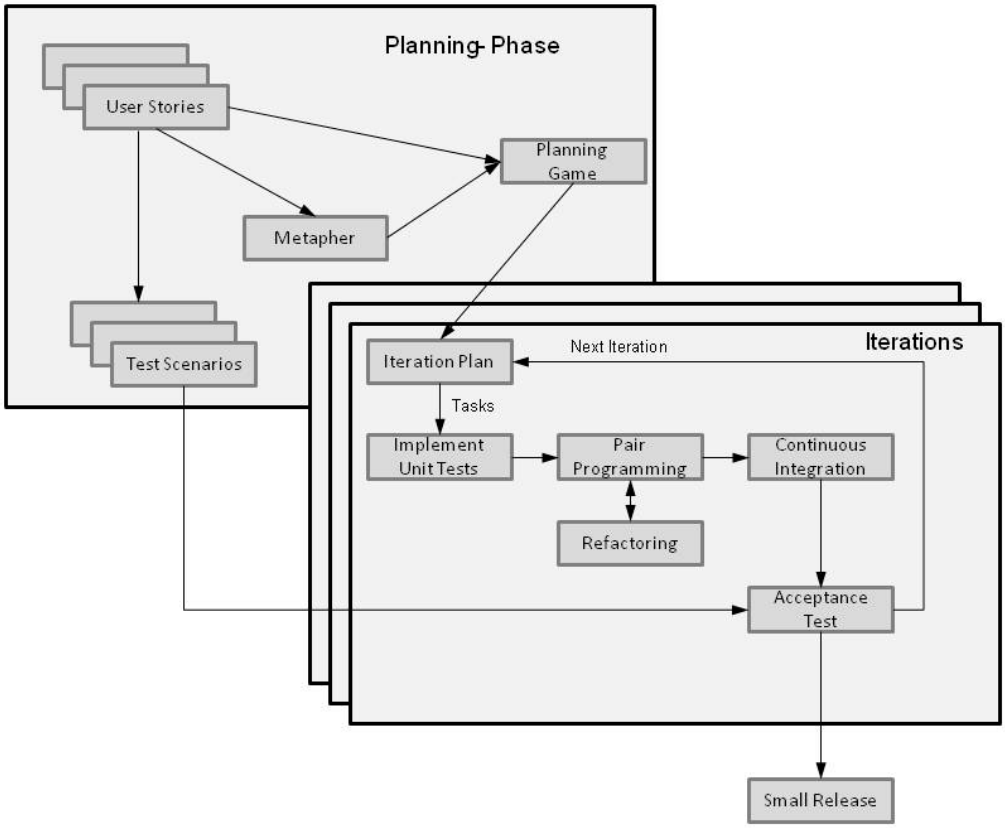


Abbildung 23: XP-Vorgehensmodell [in Anlehnung an Wells, D. (2009); Bunse, C. et al. (2008), 117]

7.3.1 Eigenschaften

In der ursprünglichen Fassung von Extreme Programming wurde kein Vorgehensmodell definiert, sondern nur Methoden und Praktiken, die miteinander in Verbindung stehen. Begründet wurde dies mit dem Argument, dass die Programmierer so wenig als möglich eingeschränkt werden sollten. Erst später hat Beck feststellen müssen, dass ein prozessorientiertes Handeln sinnvoll ist und ein rudimentäres Vorgehensmodell mit vielen Freiheitsgraden vorgestellt (vgl. Abbildung 24). Als Begründung führt Bunse etwa Schwierigkeiten bei der Zertifizierung von Unternehmen nach CMMI oder langwierige Einarbeitung von neuen Mitarbeitern an [Bunse, C. et al. (2008), 116].

7.3.2 XP- Vorgehensmodell

Prinzipiell kann ein XP-Projekt als eine Abfolge von kurzen Iterationen (1-3 Wochen) gesehen werden. Das hier betrachtete XP-Vorgehensmodell beinhaltet eine vorangestellte Planungsphase und eine Iterationsphase zur Umsetzung.

In der Planungsphase werden alle vorhandenen Anforderungen in Form von User Stories niedergeschrieben. User Stories sind in sich geschlossene Pakete, die dem Anwender Wert vermitteln. Sie werden in Schriftform verfasst und sind an keine bestimmte Form gebunden. In XP ersetzen die User Stories Requirements-Dokumente im weiteren Sinne. Im gleichen Atemzug definieren Anforderer und Programmierer Testfälle, welche die Funktionalitäten widerspiegeln. Sie werden später in den Iterationen für Abnahmetests benötigt und geben wertvolles Feedback, ob die User Stories von beiden Parteien gleichermaßen verstanden wurden. Danach kann von den Programmierern ein erster einfacher Architekturentwurf (Metaphor) entwickelt werden, um die Komplexität abschätzen zu können.

Ziel der Phase ist ein stimmiges Bild über den Scope, Timeline und Budget des Projekts. Wichtiger Bestandteil ist der Release-Plan, der im Rahmen des „Planning-Games“ (siehe Grundpraktiken) erstellt wird. Die wichtigsten Anforderungen werden zuerst entwickelt.

Jede Iteration beginnt wiederum mit einer Planung, einem „Planning-Game“ im Kleinen. Dabei werden die zu diesem Zeitpunkt wichtigsten User-Stories und eventuell neue User-Stories aufgenommen. Konnten in der vorangegangenen Iteration nicht alle Stories positiv implementiert werden, können diese in die neue Iteration einfließen oder auch verworfen werden. Die Entscheidung obliegt dem Kunden. Auch die Dauer der Iteration wird neuerlich geschätzt und abgestimmt. Das bedeutet, dass eine Iteration nicht an eine bestimmte Dauer gebunden sein muss.

Die umzusetzenden User-Stories werden dann in Tasks heruntergebrochen und unter den Programmierern (Paare) verteilt. (In weiterer Folge wird der Einfachheit halber nur mehr von User Stories gesprochen.) Für jede User-Story entwickeln die Programmierer zuerst automatisiert ausführbare Unit-Tests. Einerseits um die Aufgabe zu verstehen, andererseits um bereits erstellten Code zu schützen. Erst danach wird die eigentliche Funktionalität implementiert, getestet und integriert. Durch das Durchlaufen aller relevanten bereits bestehenden Test-Cases wird sichergestellt, dass das System nach wie vor wie gewünscht funktioniert.

Ist die letzte Iteration bezüglich der Akzeptanz-Tests positiv abgeschlossen worden, kann diese Release und somit des XP-Projekt abgeschlossen werden. Die Relation der Aufwände für die einzelnen Aktivitäten bleibt über alle Iterationen hinweg nahezu

gleich. Hauptaugenmerk fällt bei XP auf die Kernaktivitäten Implementierung und Test. Schematisch wird dies in Abbildung 25 verdeutlicht.

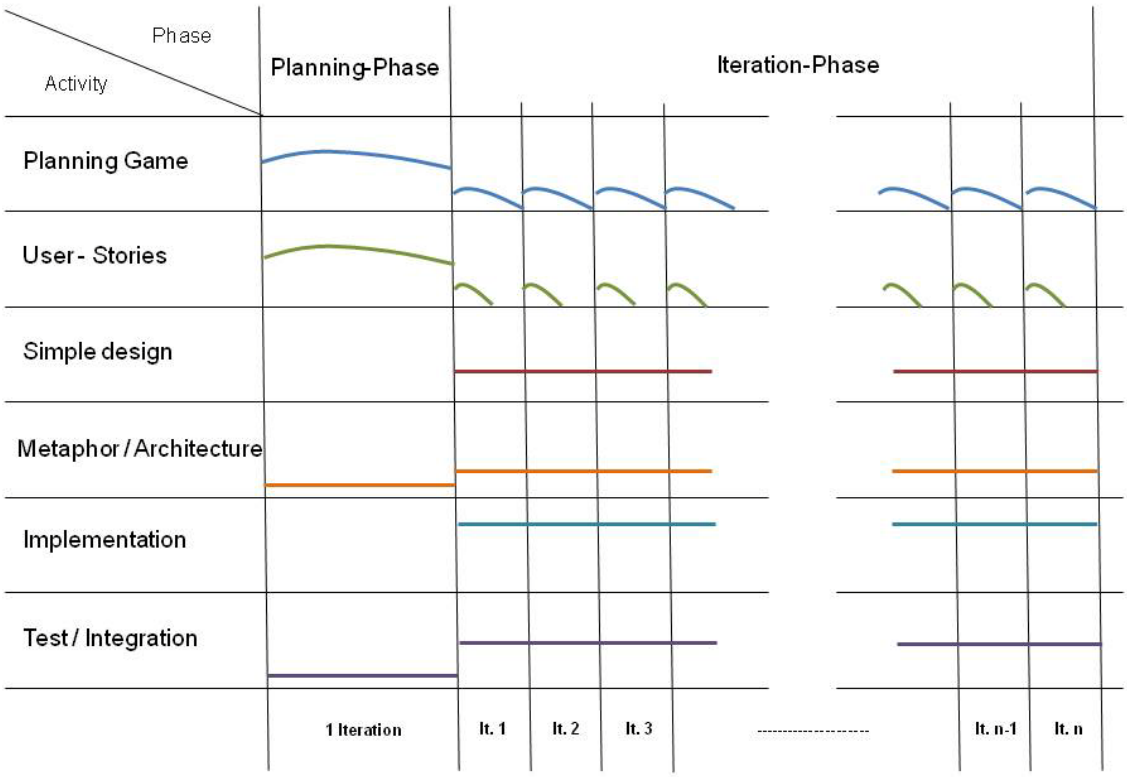


Abbildung 24: Auslastung nach Aktivitäten, schematisch

7.3.3 XP - Grundpraktiken

The Planning Game

Im Rahmen der Releaseplanung diskutieren Anforderer und Entwickler die einzelnen Anforderungen (User Stories). Der Anforderer ist aus Business-Sicht grundsätzlich für den Scope und die Priorisierung der User-Stories, den Inhalt von Releases und Launchterminen zuständig. Entwickler benötigen allerdings zur Risikominimierung die Freiheit, Architekturtreiber zuerst zu implementieren. Der Anforderer muss darüber hinaus auch seine Sicht über das Mindestmaß an Funktionalität je User Story beitragen. Dabei schätzen die Entwickler den Aufwand jeder Anforderung unter Berücksichtigung von Alternativen und Abhängigkeiten.

Small releases

Die Releasezyklen sind bewusst kurz zu wählen. Einerseits um rasch eine einsatzbereite Applikation bereitstellen zu können, andererseits um von den Anwendern Feedback über die Funktionalitäten zu erhalten. Man spricht hier von einer Größenordnung von wenigen Wochen bis Monaten. Prämisse für die Wahl der Releasedauer ist der Wert in der Wahrnehmung der Anwender. Es macht wenig Sinn Funktionalitäten zu launchen, die in diesem Stadium für sich alleine nicht nutzbar sind. Theoretisch könnte nach jeder Iteration, auch wenn sie noch so kurz ist, neue Funktionalität in die Produktivumgebung geladen werden. Die Entscheidung obliegt dem Kunden.

Metaphor

Wenn im Rahmen von XP von einer Metapher gesprochen wird, ist eigentlich die Architektur gemeint. Die Architektur entsteht ebenso inkrementell wie das System selbst. Metaphern sollen den Beteiligten im Projekt auf anschauliche Art und Weise vermitteln, was das System ausmacht.

Simple Design

Der Metapher folgend definiert Beck einfaches Design wie folgt: Bestehe alle Tests, bereinige jede doppelte Logik, folge dem kleinsten gemeinsamen Nenner an Klassen und Methoden. Der Ansatz ist konträr zur Meinung "Implement for today, design for tomorrow". Design soll nur die derzeit in Umsetzung befindlichen Anforderungen widerspiegeln.

Testing

In XP formuliert der Anforderer unter Mithilfe der Programmierer für jede User-Story funktionale Testfälle. Einerseits erhalten Programmierer dadurch neue Erkenntnisse über die Funktionalität, andererseits fungieren die Testfälle als Abnahmekriterium. Entwickler definieren automatisierte Unit-Tests und implementieren erst dann die Funktionalität. Sind alle Unit-Tests eines Inkrements positiv durchlaufen worden, gilt der Implementierungsschritt als abgeschlossen (Test Driven Development). Dabei ist darauf achtzugeben, dass zuvor implementierte User-Stories nicht korrumpiert werden. Die Unit-Tests sind also integraler Bestandteil von permanenten Regression-Tests. Eine Automatisierung ist daher unumgänglich.

Refactoring

Die Programmierer sind angehalten, den Code ständig einer Verbesserung zu unterziehen. Nachdem ein Feature implementiert und erfolgreich getestet worden ist, versucht man die Funktionalität einfacher zu gestalten. Dies allerdings nur, wenn augenfällig die Struktur (Architektur, Design) beeinträchtigt wurde und somit gegen die Praktik „Simple Design“ verstoßen worden ist (Beispiel: Logik musste dupliziert werden.). Dies gilt auch für den Code, der nicht selbst geschrieben wurde.

Pair programming

Jedes Feature wird durch ein Zweier-Team implementiert, einer codiert, der zweite hinterfragt kritisch die Aktivitäten das aktiven Programmierers und bringt eine andere Sichtweise ein. Die beiden Rollen werden häufig getauscht, genauso die Zusammensetzung der Teams. Dadurch soll eine breitere Streuung von Wissen über das System erreicht werden, die Kommunikation im Team wird erhöht.

Collective ownership

Jeder Mitarbeiter im Team darf und soll den Code ändern, das System gehört allen gleichermaßen. Umgekehrt hat jeder Programmierer auch die Verantwortung über den Code. Voraussetzung hierfür ist ein Mindestmaß an Kenntnis über alle Programmbereiche hinweg. Dieses Prinzip wird durch Pair Programming unterstützt und ist wiederum Voraussetzung für Refactoring von nicht selbst erstellten Programmteilen.

Continuous integration

Jede fertig implementierte User Story wird sofort oder zumindest am Ende des Arbeitstages in das System integriert. Dafür wird eine Umgebung benötigt, die eine kontinuierliche Integration auch zulässt. Erreicht wird dies durch eine eigene Integrationsumgebung, die von der Entwicklungsumgebung physisch getrennt ist, ein Konfigurations- und Änderungsmanagement ist unumgänglich. Die Entwickler können nur sequentiell ihren Code einchecken und Integrationstests unterziehen. Werden nicht alle Tests positiv durchlaufen und kann das Problem nicht gelöst werden, wird der ursprüngliche Zustand wieder hergestellt. Da mehrere Teams parallel am Code arbeiten, können nur kleinste Inkremente integriert werden.

40 hour week

Diese Praktik zielt darauf ab, die Belastung des Teams um leistungsfähig zu bleiben auf einem langfristig erträglichen Niveau zu halten.

On-site customer

XP fordert einen ständig für Fragen verfügbaren Nutzer der zu erstellenden Software und zwar im gleichen Raum. Dieser Kunde sollte auch ermächtigt sein, Entscheidungen im Projekt zu fällen, etwa in Fragen zur Priorisierung oder bei Unklarheiten über die Art, wie eine User-Story umgesetzt werden sollte.

Coding standards

Refactoring und Collective Ownership verlangen nach Regeln, nach denen die einzelnen Programmierpaare arbeiten. Man spricht von Programmierrichtlinien, mit denen das Format des Quellcodes genau dokumentiert ist. Durch diese Standards soll zusätzlich eine leichte Interpretation des Codes gewährleistet werden, ohne eine umfangreiche Dokumentation verfassen zu müssen.

7.3.4 Diskussion

Aussage zu Software Engineering Disziplinen

Zwei der vier Core Values beeinflussen Software Specification in einer äußerst positiven Art und Weise. Communication und Feedback erzwingen einen hochgradig kollaborativen Requirements-Management-Prozess. Durch den ständigen Austausch von Anwender und Developer oder Kunde und Lieferant ist die Wahrscheinlichkeit hoch, die wertvollsten Features in angemessener Qualität implementieren zu können.

Nachteilig wirkt sich aus meiner Sicht die lediglich rudimentär geforderte Dokumentation der User-Stories aus. Speziell in verteilten / verstreuten Projektumgebungen ist es sinnvoll, zusätzliche Methoden zur Anwendung zu bringen, beispielsweise ein Zurückgreifen auf die UML-Notation oder ein regelmäßiges, dokumentenbasiertes Zusammenfassen der User-Stories und Design-Artefakte.

Ebenfalls erachte ich es als hilfreich, eine über die Metapher hinausgehende architekturelle Überlegung vorab zu diskutieren. Dies ist besonders im Offshoring notwendig, wenn die Komplexität des Vorhabens im Sinne von Umfang und Risiko relativ hoch ist.

Obwohl Software Validation in der Diskussion grundsätzlich nicht betrachtet wird, ist „Testing“ doch integraler Bestandteil von XP und speziell im Requirements Engineering. Erst durch dieses durchgängige Konzept von Test Driven Development wird gewährleistet, dass auch in verteilten Projektumgebungen eine kontinuierliche Integration sicher gestellt werden kann.

XP setzt einen kulturellen Change in Bezug auf das Rollenbild eines „Teammitglieds“ voraus. Ein Programmierer muss demnach in der Lage sein, sämtliche Software Engineering Disziplinen zu beherrschen. Das Skill-Profil erstreckt sich nicht nur auf technische Fertigkeiten, sondern auch auf Managementaufgaben, wie sie in der Business–Analyse und im Requirements Engineering gefragt sind.

Kommunikation, Koordination, Steuerung

Die Prämisse Menschen in den Vordergrund zu stellen ist wie bei allen anderen agilen Methoden besonders herauszustreichen. In XP erfolgt die Steuerung über das Planning Game, mit dem der Scope einer Release und jeder Iteration kollaborativ festgelegt wird. Falls es über die Projektlaufzeit zu Problemen mit der Implementierung von Features kommen sollte, wird in erster Linie der Scope angepasst, nicht aber die Qualität beeinflusst. Erst in weiterer Folge werden Überlegungen hinsichtlich Kosten und Projektlaufzeit angestellt.

Die Koordination geschieht durch das Team selbst. Problematisch wird diese Praxis bei umfangreichen Vorhaben und speziell im Offshoring-Kontext. Hier ist es umso wichtiger, dass die Teammitglieder sich dessen bewusst sind und geeignete Kommunikationsstrukturen nützen. Die geforderte ständige Face-to-Face Kommunikation ist nicht möglich, daher ist die Grundpraktik On-site Customer differenziert zu sehen. Da XP nicht ausschließt einen Projektmanager zu benennen, kann dieser die koordinativen Aufgaben übernehmen und verschiedene Standorte zusammenhalten. Der Projektmanager soll aber nicht die Bedingung von sich selbst organisierenden Teams beeinflussen.

Ein Wissenstransfer wird durch mehrere Mechanismen erreicht. Durch Pair programming und Refactoring bekommen Developer einen guten Überblick über das System während des Projekts. In der Anlaufphase ist es hilfreich die Paare gemischt zu besetzen. Das bedeutet ein Developer wird vom Kunden gestellt, ein weiterer vom Lieferanten. Dadurch wird gewährleistet, dass Wissen vom Kunden zum Lieferanten transferiert wird. Bei strikter Einhaltung der agilen Werte erfolgt keine umfangreiche Dokumentati-

on. Im Offshoring-Kontext ist es meiner Meinung nach trotzdem nützlich, ein Mindestmaß an Dokumentation sicherzustellen. Einerseits weil sonst Wissen lediglich implizit im Projektteam vorhanden ist, andererseits weil in Kunde-Lieferanten Beziehungen außer ausführbarem Code auch andere Artefakte zur Bewertung der Zielerreichung maßgeblich sind.

Change Requests kommen im Sprachgebrauch von XP eigentlich nicht vor. Jede Anforderung wird gleich behandelt und kann jederzeit vom Kunden eingebracht werden. Die neuen Anforderungen werden im Rahmen der Release- und Iterationsplanung wertfrei aufgenommen und einer Priorisierung zugeführt. Allerdings ist meiner Ansicht nach darauf Bedacht zu nehmen, dass jeder weitere Request den Scope des gesamten Projekts ändern, das Budget sprengen oder die Laufzeit verlängern kann.

Anwendungsfälle

Unter folgenden Bedingungen kann XP eingesetzt werden (eigen und in Anlehnung an [Sommerville, I. (2011), 63], [Thompson, K. (2012)], [Hastie, J. (2008)], [Beck, K. (1999), 117ff]):

- Individualsoftware für den Gebrauch innerhalb des Unternehmens, wie CRM-Systeme
- Für nicht geschäftskritische Individualsoftware, wie Internet-, Intranet-Applikationen und andere Support-Systeme
- Business Intelligence Projekte
- wenn Features rasch benötigt werden
- wenn davon ausgegangen werden kann, dass sich Requirements ständig ändern
- wenn die Unternehmenskultur agile Entwicklung zulässt
- wenn die IT-Governance agile Entwicklung unterstützt (Zusammenhang mit ITIL, CMMI, CobiT und ähnlichen Frameworks)
- Bei geringem Umfang und geringer bis mittlerer Komplexität und bei geringer Kritikalität

7.4 Scrum

Als zweiten Vertreter der agilen Bewegung möchte ich Scrum vorstellen. Scrum ist mittlerweile wie Extreme Programming ähnlich weit verbreitet im Einsatz und wurde maßgeblich von Ken Schwaber geprägt. Einleitend erklärt Schwaber, dass es sich bei Scrum um einen verwirrenden und verblüffenden Prozess handelt, um komplexe Projek-

te abzuwickeln. Der Prozess selbst, die wenigen verwendeten Praktiken, Artefakte und Regeln sind überschaubar und einfach zu erlernen. Diese Leichtigkeit ist trügerisch, da nicht jede Eventualität in diesem Prozess abgedeckt ist. Scrum gibt lediglich einen organisatorischen Rahmen vor, wie trotz aller Komplexität Softwareentwicklungsprojekte Schritt für Schritt in Richtung Zielerreichung gelangen [Schwaber, K. (2004),XVII].

Der Scrum-Prozess ist geprägt von drei Worten: Apply, Inspect, Adapt [Bunse, C. et al. (2008), 126]. Damit ist eigentlich ein kontinuierlicher Verbesserungsprozess gemeint. Anforderungen werden rasch angegangen und umgesetzt, die Ergebnisse und der angewendete Prozess überprüft und wenn nötig angepasst. Schwaber spricht in diesem Zusammenhang von Inspect, Adapt und Visibility. Letzteres bezieht sich darauf, dass alle Aspekte, die Einfluss auf die Zielerreichung haben, für die handelnden Personen sichtbar und bewertbar sein müssen. Dies ist eine Voraussetzung um den Prozess steuern zu können [Schwaber, K. (2004),3].

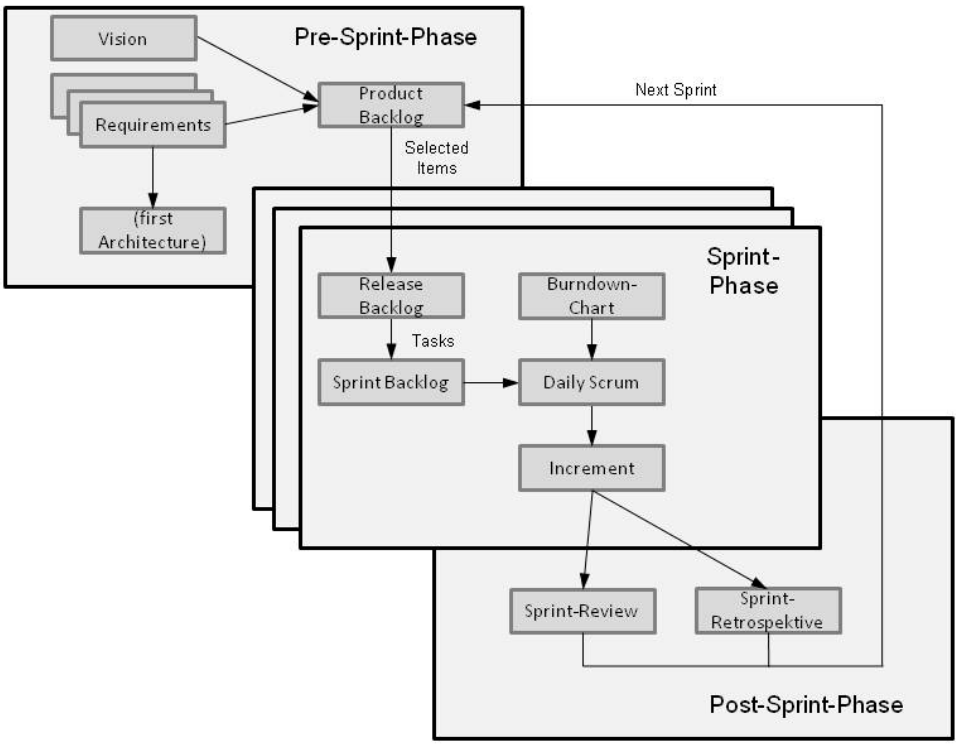


Abbildung 25: SCRUM - Vorgehensmodell

7.4.1 Eigenschaften

Der vielleicht am Stärksten ins Auge fallende Unterschied zu allen bisher betrachteten Modellen ist die Tatsache, dass sich in Scrum keinerlei Aussagen über Software Engineering Aktivitäten finden. Vielmehr versucht Scrum jene Mechanismen neu zu ordnen, die aus Sicht von Ken Schwaber nicht länger ausreichend funktionieren. Das sind zentralisierte Steuerung und Arbeitsverteilung.

Eine umfassende Planung des Gesamtprojekts wird aufgrund der sich rasch ändernden Rahmenbedingungen negiert. Den agilen Werten folgend sollen jegliche überflüssige Arbeitsschritte weggelassen werden, dies beinhaltet auch umfangreiche Dokumentation in Form von Spezifikationen. Stattdessen werden einige wenige Artefakte zur Steuerung vorgeschlagen, die unter 7.4.3 Scrum Vorgehensmodell beschrieben werden. Im Rahmen diese Arbeit wurde bisher bewusst auf die Beschreibung von Rollen innerhalb der Modelle verzichtet. Scrum stellt die Menschen in den Mittelpunkt und definiert neue Rollenbilder, die tief in die Organisationskultur eingreifen. Deshalb erachte ich es als zweckmäßig auf diese Rollen einzugehen.

7.4.2 Rollen in Scrum

Beschreibung in Anlehnung an [Schwaber, K. (2004),5] und [Ludewig, J. et al. (2010), 153]

Product Owner

Der Product Owner vertritt sämtliche Stakeholder im Projekt, insbesondere den Projektauftraggeber (im Sinne einer Projektmanagement-Rolle). Er ist verantwortlich für die Pflege und Priorisierung der Anforderungen in Form des Product Backlogs und damit für die Auswahl der wichtigsten Anforderungen für den kommenden Sprint. Der Product Owner ist idealerweise permanent für das Projekt-Team vor Ort verfügbar, um Anforderungen zu klären. Er beurteilt am Ende eines Sprints, ob die implementierten Funktionalitäten den Erwartungen entsprechen. Der Product Owner muss also von der Organisation ermächtigt sein, eigenständig Entscheidungen zu treffen.

The Team

Das Team ist verantwortlich, die ausgewählten Features eines Sprints zu implementieren. Das Team besteht aus sämtlichen Rollen (im Sinne der Software-Entwicklung, z.B.

Business Analyst, Developer,…), um die Features entwickeln zu können. Das Team ist in dem Sinne autark, als dass es sich selbst organisiert und alle Entscheidungen im Rahmen der Software-Entwicklung selbst trifft.

Scrum Master

Der Scrum-Master ist eine komplett neu definierte Rolle und dient dem Zweck den Scrum-Prozess in der Organisation zu etablieren. Er ist der Lehrer des Teams und stellt sicher, dass Scrum Regeln und Praktiken richtig angewendet werden. Er ermutigt das Team kreativ und sich selbst organisierend zu handeln. Er ist das Bindeglied zwischen Product Owner und dem Team. Er fungiert auch als Prellbock für das Team, um es von äußeren Einflüssen während eines Sprints abzuhalten. Darüber hinaus ist er verantwortlich dafür, den Fortschritt des Teams für alle Stakeholder transparent darzustellen [Schwaber, K. (2004),36].

Der Scrum Master stellt keinen Projektmanager dar, darauf wird in Scrum grundsätzlich verzichtet. Das heißt, er hat auch keine Autorität dem Team gegenüber, wohl aber über den Prozess [Cohn, M. (2010), 145]. Für größere, komplexere Projekte und speziell für verteilte Teams ist ein Projektmanager aber durchaus vertretbar. Dabei kümmert sich der Projektleiter um organisatorische Aspekte wie Projektmarketing im Unternehmen und ist Ansprechstation für persönliche Belange des Teams. Darüber hinaus ist er verantwortlich für budgetäre und Personalfragen [Eckstein, J. (2009), 43].

7.4.3 Scrum Vorgehensmodell

Kern des Modells sind mehrere hinter einander folgende Iterationen, sogenannte Sprints. Diese Sprints sind in ihrer Länge genormt, Schwaber definiert dreißig Tage pro Sprint. Am Ende eines Sprints wird immer funktionsfähige Software ausgeliefert. Innerhalb eines Sprints gibt es täglich kurze Abstimmungsmeetings, sogenannte Daily Scrums. Am Ende aller Iterationen steht die fertige Applikation oder Ausbaustufe dem Kunden zur Verfügung [Schwaber, K. (2004),5].

In Anlehnung an [Bunse, C. et al. (2008), 126] wir das Vorgehensmodell in drei Phasen unterteilt, einer Pre-Sprint-, Sprint- und Post-Sprint-Phase.

Pre-Sprint Phase

In der Pre-Sprint Phase wird zunächst eine Vision formuliert, die das zu erstellende System aus Sicht der Anforderer erst einmal vage beschreibt und im Laufe der Zeit genauere Formen annimmt. In einem weiteren Schritt wird aus den vorliegenden Requirements das initiale Product-Backlog formuliert. Diese Requirements liegen in unterschiedlichem Detaillierungsgrad vor. Die wichtigsten und somit für den ersten Sprint relevanten Anforderungen sollten jedenfalls bereits ein Stadium erlangt haben, das es ermöglicht, diese einerseits zu priorisieren und andererseits eine Aufwandsschätzung durch das Team erlaubt. Cohn spricht in diesem Zusammenhang vom Product-Backlog als Eisberg, da vage formulierte Anforderungen als „Epen“ und genauer spezifizierte z.B. als User-Stories vorliegen [Cohn, M. (2010), 274].

Am Ende der Pre-Sprint Phase sollte das Product Backlog eine Reihung nach Priorität des Product Owners aufweisen. Auch wenn in Scrum grundsätzlich vorab keine architekturelle Betrachtung vorgesehen ist und Design-Entscheidungen prinzipiell in den Sprints vom Team “on demand“ getroffen werden, bietet sich die Möglichkeit in der Pre-Sprint-Phase eine initiale Architekturbetrachtung aufgrund des derzeit vorliegenden Product-Backlogs durchzuführen. Dies hilft bei komplexeren und verteilten Vorhaben die grundlegende Stoßrichtung für den ersten Sprint festzuhalten und eine gemeinsame Sicht über einen möglichen Aufbau zu erhalten. Die Tätigkeiten werden idealerweise in einem Anforderungs-Workshop definiert, in dem alle Rollen teilnehmen.

Sprint Phase

Jeder Sprint beginnt mit einem Sprint Planning Meeting. Dieses Meeting ist genormt und dauert acht Stunden. Im ersten Teil präsentiert der Product Owner dem Team jene Anforderungen, welche die höchste Priorität aufweisen. Das Team hinterfragt die Anforderungen solange bis ein gegenseitiges Verständnis herrscht. Das Team schätzt den Aufwand für die initialen Features und wählt so viele davon aus, als es glaubt im anstehenden Sprint vollständig implementieren zu können. Im Sprachgebrauch von Scrum spricht man von „done-done“, was im Prinzip „potentiell auslieferbar“ bedeutet. Daraus resultiert quasi ein Release-Backlog für den Sprint. In der zweiten Hälfte des Meetings plant das Team die Umsetzung der ausgewählten Features und bricht dabei die Requirements in Tasks aus technischer Sicht herunter. Jeder Task steht dabei für einen in sich geschlossenen Arbeitsschritt, der nicht länger als sechzehn Stunden, also zwei Arbeitstage, dauern darf. Damit beginnt der Sprint zu laufen. Das Team organisiert sich

selbst, in dem die Tasks unter den Teammitgliedern eigenverantwortlich verteilt werden.

Täglich trifft sich das Team zum „Daily-Scrum“ Meeting, einem 15-minütigen Informations-Termin. Jedes Mitglied berichtet, welche Tätigkeiten seit dem letzten Meeting durchgeführt wurden, welche Tätigkeiten noch folgen und welche Hindernisse zu überwinden sind. Daraus resultierende offene Fragen werden in einem anderen Rahmen diskutiert. Resultate werden dem Team entweder sofort nach Lösungsfindung oder im folgenden Daily-Scrum mitgeteilt. Das Meeting dient der „Visibility“ und ist primär ein Instrument zum Projektmanagement [Bunse, C. et al. (2008), 128].

Der Scrum Master führt während des Sprints einen Sprint-Burndown-Chart, das die Geschwindigkeit der Abarbeitung der Tasks dokumentiert (Velocity). Zur Übersicht fügt sich das Sprint-Burndown-Chart in den Release-Burndown-Chart ein, neben dem Product-Backlog das Steuerungsinstrument für das Gesamtprojekt.

Das Product Backlog wird laufend vom Product-Owner aktualisiert. Neue Anforderungen werden aufgenommen, bestehende repriorisiert oder gänzlich herausgenommen. Dies hat allerdings keinen Einfluss auf den laufenden Sprint, der Umfang bleibt grundsätzlich unberührt.

Wenn sich herausstellen sollte, dass der definierte Inhalt eines Sprints nicht in der Art umgesetzt werden kann, darf der Scrum-Master den Sprint abbrechen, um neuerlich ein Sprint-Planning Meeting abzuhalten. Auf der anderen Seite können zusätzliche Anforderungen aus dem Product-Backlog in einen laufenden Sprint aufgenommen werden, wenn die Teamgeschwindigkeit höher ist als zuerst angenommen. Die Entscheidung darüber obliegt dem Product Owner [Schwaber, K. (2004),136].

Am Ende eines Sprints sind idealerweise alle im Sprint-Backlog angeführten Tasks abgearbeitet und funktionsfähige Software als Inkrement für den Kunden auslieferbar vorhanden.

Post-Sprint Phase

Die Post-Sprint Phase ist geprägt von zwei Meetings, dem Sprint Review Meeting und der Sprint Retrospektive.

Beim auf vier Stunden angesetzten Sprint Review Meeting präsentiert das Team allen Stakeholdern die erreichten Ergebnisse des Sprints. Änderungswünsche zu bereits imp-

lementierten Features werden notiert und gegebenenfalls in den Product Backlog überführt. Gleiches gilt für neue Funktionalitäten, die im Rahmen der Diskussion zutage treten.

Die Sprint Retrospektive ist mit drei Stunden angesetzt und gilt nur für das Team, Scrum Master und Product Owner. Ziel des Termins sind potentielle Verbesserungspotentiale aufzudecken, um den Scrum-Prozess besser umsetzen zu können.

7.4.4 Diskussion

Aussage zu Software Engineering Disziplinen

Scrum lässt hinsichtlich Software Engineering viele Fragen offen. Methoden und Techniken werden im Team definiert, um die jeweilige Aufgabe bestmöglich umzusetzen. In der Realität sind diese Freiheiten oft beschnitten, wenn zwingende Vorgaben im Unternehmen existieren oder freiwillig das Regelwerk aufgrund von Erfahrung erweitert wird. Ich denke, dass im agilen Umfeld Standardisierung durchaus Sinn macht, um nicht jedes Mal das Rad neu erfinden zu müssen.

Oftmals wird Scrum als Projektmanagementmethode verwendet und mit Praktiken aus XP ergänzt. Im Gegensatz zu XP sind im Team grundsätzlich keine Generalisten enthalten, sondern alle für die Features benötigten Rollen, wie z.B. Business Analyst und Developer.

Kommunikation, Koordination, Steuerung

Das Modell ist in allen Belangen strikt getaktet, sei es eine Iteration oder auf der Mikroebene ein Daily-Scrum Meeting. Dadurch orientiert sich das gesamte Projekt-Team an einem Rahmen und lernt mit der Zeit, wie viele Features pro Sprint umgesetzt werden können. Erreicht wird dies durch die Anwendung normierter Schätzverfahren, die mit der Teamgeschwindigkeit (Velocity) in Verbindung gesetzt werden.

Die Koordination von größeren Projekt-Teams als die optimale Anzahl von sieben Teammitgliedern wird durch die Schaffung mehrerer relativ unabhängiger Teams erreicht. Diese werden Scrum of Scrums oder Meta-Scrums genannt. Theoretisch kann beliebig skaliert werden, allerdings steigt mit jeder eingezogenen Ebene die Wahrscheinlichkeit von Informationsverlust, was die Steuerung negativ beeinflusst. Bei Meta-Scrums wird die übergreifende Koordination durch jeweils einen Vertreter eines

Scrum-Teams erreicht. Bei steigender Anzahl an Scrum-Teams, würde ein Meta-Scrum zu viele Teilnehmer aufweisen. Es bietet sich an entweder Meta-Scrums je Standort oder je Domäne, wie auch immer diese aussehen, zu etablieren. Wie beim Daily-Scrum berichten einzelne Teammitglieder über die Inhalte des jeweiligen Feature-Teams. Die XP-Praktik Collective Ownership kann so nicht direkt erreicht werden, da jedes Team für einen Teil der Software verantwortlich ist. Um Wissen breiter zu streuen bietet sich eine Rochade von Teammitgliedern nach einem abgeschlossenen Sprint an. Dies ist besonders im Offshoring mit verteilten Teams problematisch, da die Face-to-Face Komponente fehlt. Restriktives Element für Scrum-Projekte höheren Umfangs ist die Integrationsumgebung (vgl. XP-Praktik Continuous Integration). Da nicht gleichzeitig integriert werden darf, kann es hier zu einem Engpass kommen.

Mit der Sprint-Retrospektive wird ein kontinuierlicher Verbesserungsprozess institutionalisiert. Es geht dabei nicht darum Fehler zu dokumentieren, sondern für kommende Sprints zu lernen. Bei verteilter Entwicklung können aufgrund der räumlichen Distanz Vertreter der Scrum-Teams in übergeordneten Retrospektive-Sitzungen zusammen kommen. Praktikabel erscheint eine Taktung von 2-3 Iterationen.

Der mit Scrum neu definierte Scrum-Master übernimmt als „Trainer“ und „Facilitator“ eine Rolle, um eine optimale Kommunikation, Koordination und Steuerung im Scrum-Projekt zu ermöglichen. Mit wachsamem Auge schnappt er Probleme auf und ermöglicht es dem Team, diese zu lösen. Durch seine Schnittstellenfunktion zum Kunden stellt er ein Bindeglied zum Team dar.

Die Anlaufphase im Offshoring-Modus ist bei kleinen Projekten mit nur einem Team wenn möglich eine gemischte Teambesetzung anzustreben. Bei umfangreicheren Projekten bietet sich eine Vorgehensweise an, die mit nur einem oder zwei Teams in der ersten Iteration beginnt. In der zweiten Iteration wird dieses Team aufgeteilt und um neue Mitarbeiter ergänzt. Ist die maximale Anzahl an Teams erreicht, kommt die Rotation von Teammitgliedern wieder zum Tragen.

Für Change Requests gilt gleiches wie im Extreme Programming. Eine Aufnahme in das Product Backlog ist jederzeit möglich, der laufende Sprint bleibt von neuen Anforderungen allerdings unberührt.

Anwendungsfälle

Unter folgenden Bedingungen kann Scrum eingesetzt werden (eigen und in Anlehnung an [Sommerville, I. (2011), 63], [Thompson, K. (2012)], [Hastie, J. (2008)]):

– Es gilt selbiges wie für Extreme Programming, siehe Kapitel 7.3.4Diskussion
– Mit Scrum sind zusätzlich Projekte mittleren Umfangs und Komplexität denkbar (Scrum of Scrums). Die Kritikalität sollte auf niedrigem Niveau bleiben.

8 Modellbildung

In diesem Kapitel wird ein Leitfaden basierend auf den in den vorigen Kapiteln herausgearbeiteten Erkenntnissen erstellt. Die erste Stufe bildet die Ableitung eines passenden Delivery-Modells. Die zweite Stufe gibt Anhaltspunkte für die Auswahl des Kollaborationsmodells. Die dritte Stufe stellt die Relation zu einem Software-Entwicklungsmodell her.

Es wird bewusst darauf verzichtet, den einzelnen Kriterien Werte zuzuordnen, um in weiterer Folge basierend auf einem Kalkulationsschema zu einer Entscheidung zu gelangen. Im Laufe der Recherche zu dieser Arbeit hat sich herausgestellt, dass es keine allgemeingültige Bewertungsmethode für die betrachteten Sachverhalte gibt. Objektiv bewertbare Aspekte scheinen meiner Wahrnehmung nach von Autoren und Praktikern unterschiedlich bemessen. Es kann lediglich von einer Tendenz eher für oder eher gegen eine Methode, Praktik, Modell gesprochen werden. Ein Aspekt erscheint eher hoch oder eher niedrig. Kriterien haben teils Bezug zum Unternehmenskontext, teils dreht sich die Bewertung um Projekte. Meiner Ansicht nach ist die subjektive Einschätzung der Verantwortlichen in den Unternehmen wichtiger einzustufen als die Eingliederung in rigide, intervallbasierte Skalen und Tabellen.

Beispielhaft einige Gedanken dazu:

- Einsparungspotential: In erster Linie gilt es eine strategische Entscheidung zu treffen, wie niedrig die Kostenlatte angesetzt wird. Der angestrebte Zielwert wird von einem Unternehmen eher hoch bewertet, für ein anderes wiederum als niedrig klassifiziert. Es kommt auf den Unternehmenskontext an.
- Projektumfang: Projekte können verschiedenen Kategorien zugeordnet werden. Eine Beurteilung der „Größe“ ist wiederum eine subjektive Einschätzung und im Unternehmenskontext zu bewerten.
- Kritikalität: Imageverlust kann nicht für sich alleine in Zahlen ausgedrückt werden.
- Kritikalität: Umsatzrückgang kann antizipiert und monetär bewertet werden. Die Auswirkung dessen auf das Unternehmen ist wiederum im Kontext dazu zu sehen.

Um einen Leitfaden dieser Art erstellen zu können, müssen die Erkenntnisse verdichtet und vergleichbar gemacht werden. Wenn daher bei der Anwendung der folgenden Ent-

scheidungshilfen tendenziell eine Ausprägung hervorgestrichen wird, sei trotzdem auf die detaillierten Ausführungen in den vorigen Kapiteln verwiesen.

8.1 Ableitung des Delivery-Modells

Unternehmenskontext

Führendes Motiv für Software Development Offshoring ist es eine Kostenreduktion zu erreichen. Hierbei handelt es sich um eine strategische Vorgabe des Unternehmens. Prinzipiell lässt sich schlussfolgern, dass je größer die räumliche Distanz zum Lieferanten ist, desto niedriger sind die Entwicklungskosten (vgl. Abbildung 3).

Für die Auswahl eines geeigneten Delivery-Modells ist „Erfahrung“ ein nicht zu unterschätzender Aspekt.

Erfahrung bezieht sich in diesem Kontext auf

- Kenntnisse im Lieferantenmanagement
- Erfahrungen bezüglich Entwicklungs-Sourcing bzw. Offshoring
- Unternehmenskultur (prozessorientiert vs. chaotisch)
- Unternehmensgröße

Falls keine Fertigkeiten bestehen Lieferanten auszusteuern, sollte zuerst der Weg mit inländischen Vendoren bestritten werden. Die Risiken im Offshoring-Modus wären meiner Ansicht nach zu hoch, um einen Projekterfolg zu gewährleisten. Sind bereits Projekte mit Lieferanten aus dem Inland abgewickelt worden, begünstigt das die Erfolgsaussichten. Der nächste Schritt wäre auf Lieferanten aus dem nahe gelegenen Ausland mit Onsite- oder Onshore- Komponente zurück zu greifen. Erst wenn Projekte mit diesem Setup erfolgreich abgeschlossen wurden, würden sich günstigere Varianten mit hohem Offshore-Anteil anbieten.

Prozessorientiert operierende Unternehmen haben höhere Erfolgschancen Offshore-Projekte angemessen abzuschließen (vgl. Kapitel 4.2.7 Standardisierte und dokumentierte Prozesse). Dies steht im Zusammenhang mit der Unternehmensgröße. Tendenziell sind kleinere Unternehmen eher chaotisch und nicht allzu prozessorientiert ausgerichtet.

Projektkontext

In Kapitel 3 Delivery-Modelle wurden Konstellationen aus der Praxis basierend auf den ausgewählten Erfolgsfaktoren diskutiert. Daraus lassen sich Ausprägungen ableiten, die eine Auswahl vergleichbar machen.

Die kritischen Erfolgsfaktoren beziehen sich hauptsächlich auf Themen der Kategorien Kommunikation, Koordination und Steuerung. Diese drei Ausprägungen gehören zu den Hauptinhalten des Projektmanagements.

Ein Projekt kann kategorisiert werden in

- Umfang
- Kritikalität
- Komplexität

Der Projektumfang kann klassifiziert werden durch

- Geplante Projektdauer
- Anzahl Mitarbeiter (in internen Personentage)
- Investitionsvolumen

Die Kritikalität kann bestimmt werden durch

- Umsatzeinbruch
- Gewinnminderung
- Imageverlust
- Marktdruck
- Regulatorische Vorgaben
- Unumgänglicher Fertigstellungstermin
- Beeinträchtigung wesentlicher Geschäftsprozesse
- Spezielle Sicherheitsbestimmungen durch vertrauliche Daten
- Verlässlichkeit und Verfügbarkeit des Systems

Die Komplexität kann eruiert werden durch

- Anzahl Systeme
- Anzahl Schnittstellen
- Anzahl verwendeter Technologien
- Anzahl beeinflusster Geschäftsprozesse
- Abhängigkeiten zu anderen Projekten
- Anzahl mitwirkender Organisationseinheiten
- Anzahl Lieferanten

Je höher Umfang, Kritikalität und Komplexität eines Softwareentwicklungsprojekts sind, desto mehr muss das Augenmerk auf funktionierende Kommunikationsmechanismen und –medien, Projektkoordination und Steuerungselemente gelegt werden.

Kommunikation, Koordination und Steuerung funktionieren umso besser,

- je näher ein Lieferant lokalisiert ist
- wenn die Zeitzonenunterschiede gering sind
- je weniger (Human-) Schnittstellen überwunden werden müssen.

Je höher die Kritikalität und Komplexität eines Projekts ist, desto wichtiger wird persönliche Kommunikation, daher gilt wenn möglich die Auswahl in diesen Fällen auf Delivery-Modelle zu beschränken, die gänzlich Onsite oder Onshore abgewickelt werden. Ansonsten bieten sich Modelle mit einem hohen Anteil Onsite oder Onshore an. Für höchst kritische Projekte sind Global Delivery-Modelle aufgrund ihrer hohen Anzahl an Humanschnittstellen nicht geeignet.

Netzdiagramme

Durch die vielfältigen Bewertungskriterien im Projektkontext scheint es hilfreich, diese gesondert zu beurteilen. In Abbildung 27 findet sich beispielhaft die Bewertung eines Vorhabens in Form eines Netzdiagramms, wobei innen niedrigere und außen höhere Werte zu finden sind.

In Abbildung 28 wird das Ergebnis zu den Kategorien Umfang, Kritikalität und Komplexität verdichtet und mit dem Unternehmenskontext in Verbindung gebracht. Daraus resultiert eine Tendenz, welches Delivery-Modell grundsätzlich angewendet werden sollte.

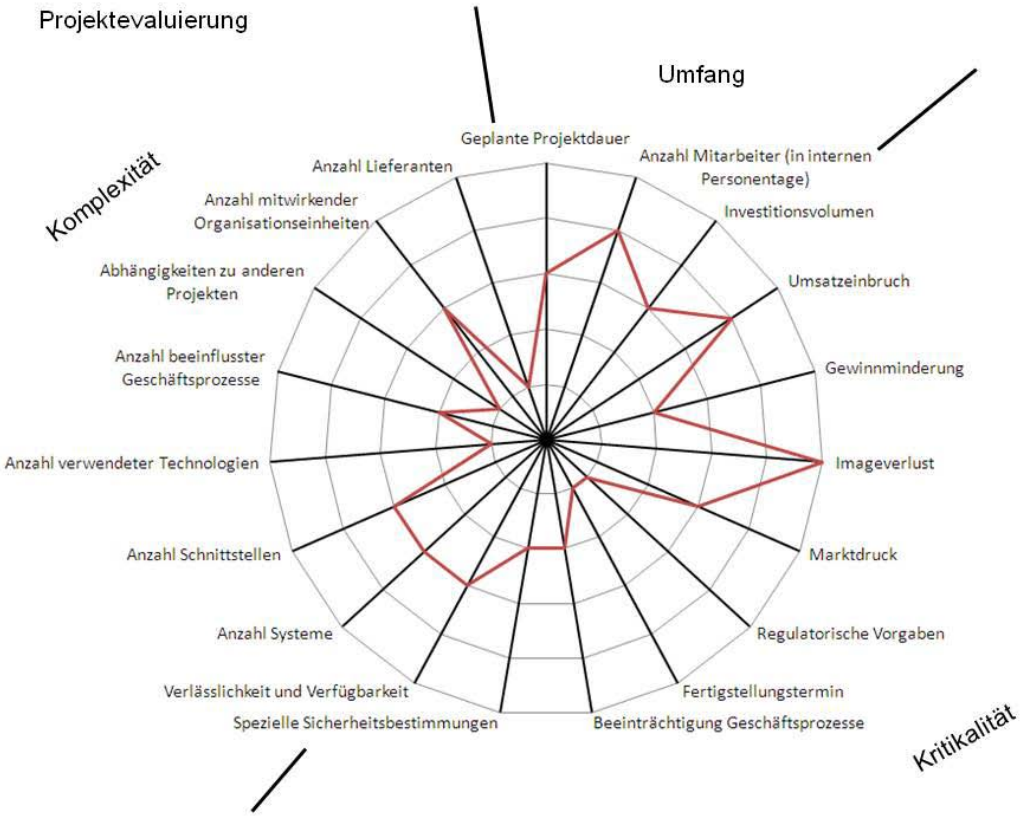


Abbildung 26: Projektevaluierug

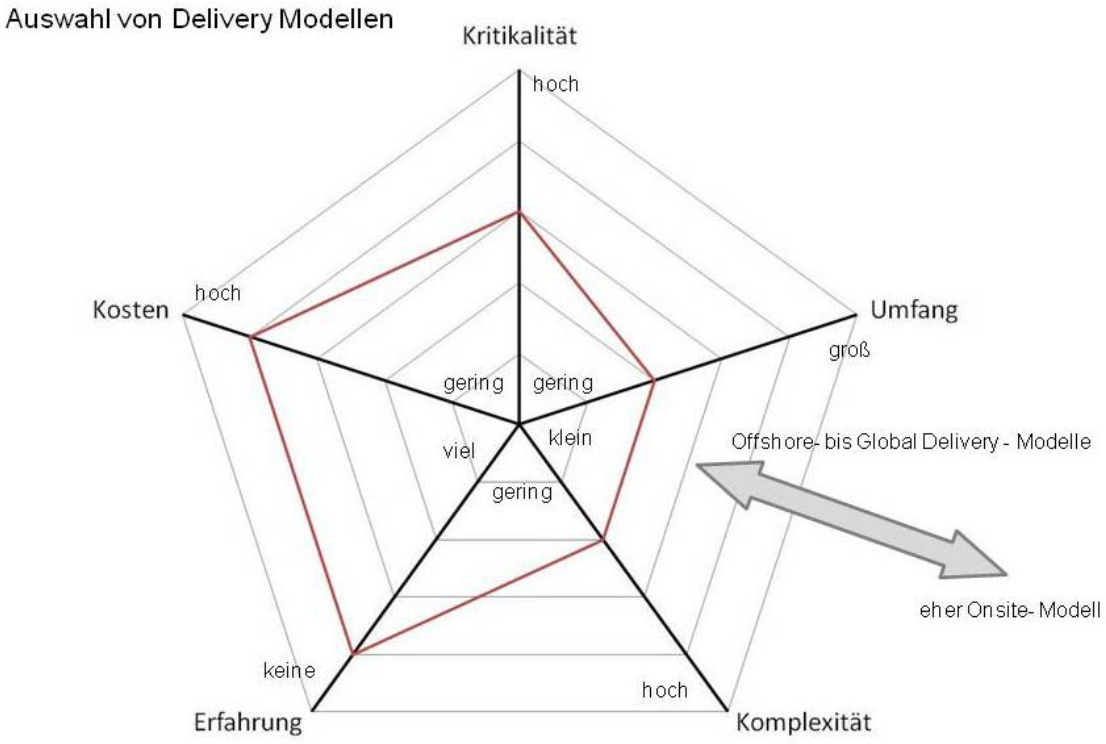


Abbildung 27: Auswahl Delivery Modell

8.2 Ableitung des Kollaborationsmodells

Im Gegensatz zur Bestimmung von Delivery-Modellen gilt für Kollaborationsmodelle ein wesentlich stärkerer Bezug zum Unternehmenskontext. In Kapitel 4 Kollaborationsmodelle wurden fünf Konstellationen eingehend basierend auf kritischen Erfolgsfaktoren diskutiert und bezüglich Projektkategorien und möglicher Software-Entwicklungsmodelle beleuchtet. Der Fokus liegt auf der Offshore-Readiness des Auftraggebers und auf der Fähigkeit zu Durchführung von Offshore-Softwareentwicklungsprojekten.

Hauptaugenmerk liegt meiner Ansicht nach darauf, welche Software Engineering Rollen vom Kunden überhaupt besetzt werden können. In weiterer Folge muss die Organisation projekthaftes Vorgehen gewährleisten. In gewisser Weise geht dies einher mit der strategischen Frage, welche Tätigkeiten im Unternehmen durchgeführt werden sollen und welche am Markt zugekauft werden. Für die detaillierte Diskussion sei an dieser Stelle auf Kapitel 4.3 Ausgewählte Kollaborationsmodelle unter Berücksichtigung der Erfolgsfaktoren verwiesen.

Abbildung 29 zeigt eine Gegenüberstellung der Kollaborationsmodelle mit den direkt vergleichbaren Erkenntnissen in Bezug auf

Unternehmenskontext

Unter diesem Punkt wird hauptsächlich die Fähigkeit eines Unternehmens betrachtet, Software Engineering Aktivitäten durchzuführen. Dabei steht „+“ für hohe Fertigkeiten bzw. sämtliche Disziplinen (nahezu) duchführbar und „-“ für geringe bis keine Fertigkeiten bzw. nur im Bereich Requirements Engineering.

Projektkontext

Der Projektumfang ist in diesem Zusammenhang eher untergeordnet zu betrachten. Vielmehr bedingen die unterschiedlichen Kollaborationsmodelle teilweise Einschränkungen in den Ausprägungen Komplexität und Kritikalität.

Software-Entwicklungsmodell

Unter diesem Punkt erfolgt eine Einordnung, welches der in Kapitel 7 Ausgewählte Software-Entwicklungsmodelle diskutierten Vorgehensmodelle grundsätzlich eigesetzt werden kann. Es erfolgt an dieser Stelle keine Wertung welches Modell vorzuziehen ist.

Nach derzeitigem Erkenntnisstand (vgl. Kapitel 4.3 Ausgewählte Kollaborationsmodelle unter Berücksichtigung der Erfolgsfaktoren) existieren jedoch Einschränkungen, die in Abbildung 29 dargestellt sind.

Kollaborationsmodelle im Vergleich	*Unternehmen*	*Projekt*		*SW-Entwicklungsmodell*		
	Skills	**Komplexität**	**Kritikalität**	**sequentiell**	**iterativ/ inkrementell**	**agil**
Extended Workbench	+	gering - hoch	gering - hoch	+	-	-
Over the Fence	-	gering	gering	+	+	o
Erweiterte Extended Workbench	+	gering - hoch	gering - hoch	+	+	o
Over the Fence light	-	gering	gering	+	+	+
Setup im agilen Umfeld	+	gering - mittel	gering	-	-	+

Abbildung 28: Kollaborationsmodelle im Vergleich

8.3 Ableitung des Software-Entwicklungsmodells

In den folgenden Ausführungen werden Erkenntnisse aus der Diskussion der jeweiligen Software-Entwicklungsmodelle in Relation gebracht. Dafür werden Tabellen erstellt, mit deren Hilfe eine Gegenüberstellung einfacher zu bewerkstelligen ist. Die Ausprägungen werden kurz erläutert. Ich möchte nicht darüber hinwegtäuschen, dass es sich hierbei um eine stark verkürzte Sicht handelt. Jedes Modell hat seine Stärken und Schwächen und korrekt angewendet seine Berechtigung. Für Details sei auf die vorigen Kapitel zum Thema Software-Entwicklungsmodelle, jeweils Unterkapitel „Diskussion“ verwiesen.

Software Engineering

Der Wasserfall hat Schwächen im Bereich Requirements Engineering, da sämtliche Anforderungen vorab definiert werden müssen. Jeder Anforderung wird grundsätzlich die gleiche Priorität beigemessen. Dies kann zu mangelhaften Architekturen führen. Eine Entdeckung von Designfehlern erfolgt frühestens mit der Implementierung oder mit der Integration der Teile. Auch das beeinflusst die Architektur negativ, da kurz vor Projektende die gesamte Systemarchitektur nicht mehr aufgebrochen, sondern nur gefixt wird.

Im Unified Process steht die architektonische Betrachtung im Vordergrund. Anforderungen, die als Architekturtreiber angesehen werden, müssen zuerst ausformuliert werden. Ziel ist eine stabile Architektur zu erreichen. Die Erhebung erfolgt strukturiert in Form von Use-Cases. Überhaupt verwendet der Unified Process anschauliche Modelle

zur Beschreibung funktionaler und technischer Sachverhalte. Modelle bergen allerdings die Gefahr, dass wesentliche Informationen für neue Mitarbeiter und andere Stakeholder verloren gehen. Die Feedbackschleifen, ob einzelne Sachverhalte richtig verstanden wurden, sind ausreichend gegeben. Erreicht wird dies, durch sukzessive Verfeinerung der Artefakte und z.B Prototyping zum besseren Verständnis.

Bei agilen Methoden beziehe ich mich im Software Engineering Kontext auf Extreme Programming, da Scrum wenig über Software Engineering aussagt. Die Anwendbarkeit für ein bestimmtes Projekt vorausgesetzt, bietet XP durch den ständigen Informationsaustausch eine hervorragende Ausgangsbasis, sodass die Anforderungen korrekt implementiert werden können. Korrekt bezieht sich allerdings auf die Funktion und nicht zwingend auf die Architektur. Refactoring als probates Mittel verspricht architekturelle Defizite wieder auszugleichen. Fraglich ist, ob diese Praxis wegen des häufigen Termindrucks auch angemessen durchgeführt werden kann. Als negativ ist aus meiner Sicht die unzureichende Dokumentation zu sehen. Das ist der Grund für eine neutrale Bewertung.

In Abbildung 30 sind die Erläuterungen zusammengefasst dargestellt

SW-Entwicklungsmodelle im Vergleich	*Software Engineering*	
	Software Specification	Software Design
Wasserfall (sequentiell)	-	-
Unified Process (interativ/ inkrementell)	++	++
XP / Scrum (agil)	o	o

Abbildung 29: SW-Modelle vs. Software Engineering

Kommunikation, Koordination, Steuerung

Der Wasserfall ist strikt dokumentenbasiert. Eine neutrale Bewertung erfolgt weil Projekte denkbar sind, in denen die Kommunikation in Schriftform ausreichend ist. Das ist insbesondere bei wiederkehrenden, ident aufgebauten Anforderungen der Fall. Es mangelt per Definition an persönlicher Kommunikation. Dies hat Auswirkung auf den Wissenstransfer. Change Requests sind kaum möglich.

Im Unified Process werden zwar auch Artefakte in Schriftform verfasst, allerdings werden diese häufig aus CASE Tools extrahiert und sind somit bezüglich Aktualität und Strukturiertheit weitaus höher einzustufen als die Güte der Dokumente im Wasserfall.

Direkter Kontakt mit dem Kunden erfolgt häufig, ein Wissenstransfer wird teils durch Artefakte, teils persönlich durchgeführt. Dadurch wird eine positive Bewertung abgegeben. Change Requests können bis zum Ende der Elaborationsphase eingebracht werden. Später im Prozess nur dann, wenn die Architektur nicht beeinträchtigt wird.

Bei agilen Methoden steht die persönliche Kommunikation im Vordergrund, Dokumente werden selten erstellt. Dadurch erfolgt der Wissensaustausch hochgradig mittels direkten Kommunikationswegen. Da der Terminus Change Request im Vokabular von Agilisten nicht vorkommt, passen diese Modelle für Projekte mit noch hochgradig unbekannten und sich ändernden Anforderungen.

In Abbildung 31 sind die Erläuterungen zusammengefasst dargestellt

SW-Entwicklungsmodelle im Vergleich	*Kommunikation, Koordination, Steuerung*			
	dokumenten-basiert	direkte, persönliche Kommuni-kation	Wissens-transfer	Change Request Handling
Wasserfall (sequentiell)	o	--	dokumenten-basiert	--
Unified Process (interativ/ inkrementell)	+	+	dokumenten-basiert / persönlich	+
XP / Scrum (agil)	-	++	persönlich	++

Abbildung 30: SW-Modelle vs. Kommunikation

Komplexität, Kritikalität und Projektumfang

Die Gegenüberstellung in Abbildung 32 stellt eine Zusammenfassung der Erkenntnisse unter der Rubrik Anwendungsfälle im Unterkapitel „Diskussion“ der Software-Entwicklungsmodelle dar. Dazu sind folgende Erläuterungen nötig.

Für bestimmte Projekte kann es vorkommen, dass ein sequentielles Vorgehen wie beim Wasserfall zwingend nötig wird. Das kann bei regulatorischen Vorgaben oder sicherheitskritischen Projekten der Fall sein. Hier würde eine mittlere bis hohe Kritikalität zutreffen.

– Komplexität gering bis mittel
– Kritikalität mittel bis hoch
– Umfang mittel

Ein Wasserfall kann auch für kleine Anforderungen (minor enhancements) ausreichend sein.

– Komplexität gering
– Kritikalität gering
– Umfang gering

Der Unified Process ist bestens geeignet für hochkritische Projekte größeren Umfangs mit hoher Komplexität.

– Komplexität hoch
– Kritikalität hoch
– Umfang hoch

Können agile Methoden nicht mehr angewendet werden, sei es weil der Umfang eine ordentliche Steuerung nicht mehr zulässt, ist eine Vorgehensweise angeraten, die sich mehr an strikten Vorgaben orientiert. Hierfür kommen die iterativ/ inkrementellen Modelle wie der UP in Betracht.

– Komplexität mittel
– Kritikalität mittel
– Umfang mittel

Mit Extreme Programming alleine können aufgrund der überschaubaren Teamgröße lediglich Projekte mit geringerem Umfang abgewickelt werden.

– Komplexität gering bis mittel
– Kritikalität gering
– Umfang gering

Mit Scrum sind zusätzlich Projekte mittleren Umfangs und Komplexität aufgrund der Skalierbarkeit denkbar (Scrum of Scrums). Die Kritikalität sollte auf niedrigem Niveau bleiben.

– Komplexität gering bis mittel
– Kritikalität gering
– Umfang gering bis mittel

In Abbildung 32 sind die Erläuterungen zusammengefasst dargestellt.

SW-Entwicklungsmodelle im Vergleich	*Projekt*		
	Komplexität	**Kritikalität**	**Umfang**
Wasserfall (sequentiell)	gering-mittel	gering-hoch	gering-mittel
Unified Process (interativ/ inkrementell)	mittel-hoch	mittel-hoch	mittel-hoch
XP / Scrum (agil)	gering-mittel	gering	gering-mittel

Abbildung 31: SW-Modelle vs. Projektkategorie

8.4 Arbeiten mit dem Modell

Abbildung 33 zeigt die Wechselwirkungen zwischen

– Delivery-Modell
– Kollaborationsmodell
– Software-Entwicklungsmodell

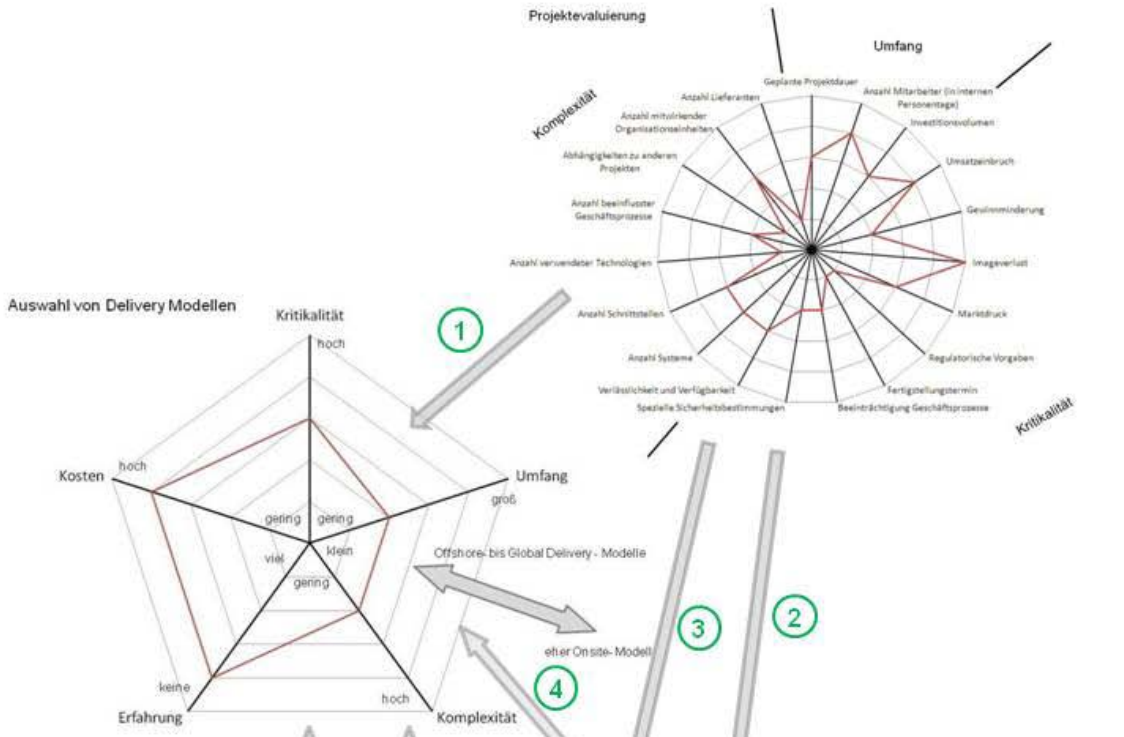
und

– Projektkontext
– Unternehmenskontext

Die Projektevaluierung hat direkten Einfluss auf die Auswahl des Delivery-Modells (1) und liefert Informationen, ob ein Kollaborationsmodell grundsätzlich gewählt werden kann (2). Mit (3) kann bestimmt werden, ob ein Projekt grundsätzlich zu einem Software-Entwicklungsmodell passt. Die Wechselwirkung zwischen Delivery-Modell und Kollaborationsmodell ist in (4) ersichtlich. (5) zeigt die Wechselwirkung zwischen Kollaborationsmodell und Software-Entwicklungsmodell. Ob ein Softwareentwicklungsprojekt zu einem Delivery-Modell passt, ist in (6) und indirekt über (7) ersichtlich. Die Stärken und Schwächen eines Software-Entwicklungsmodells in Bezug auf Software Engineering kann mit den vorhandenen Skills im Unternehmen in Relation gesetzt werden (8).

Das Modell kann vielschichtig genutzt werden. Verschiedene Parameter wie ein Kostenlimit oder ein zwingend vorgeschriebenes Entwicklungsmodell können fixiert werden. Die Auswirkung auf andere Parameter wird offensichtlich. Ein bestimmtes Kolla-

borationsmodell könnte aus strategischen Gesichtspunkten auszuwählen sein. Beispielsweise wäre dies eine erweiterte Extended Workbench, wenn architekturelle Entscheidungen im Unternehmen bleiben sollen.



Kollaborationsmodelle im Vergleich

	Unternehmen	*Projekt*		*SW-Entwicklungsmodell*		
	Skills	**Komplexität**	**Kritikalität**	**sequentiell**	**iterativ/ inkrementell**	**agil**
Extended Workbench	+	gering - hoch	gering - hoch	+		-
Over the Fence	-	gering	gering	+		o
Erweiterte Extended Workbench	+	gering - hoch	gering - hoch	+	+	o
Over the Fence light	-	gering	gering	+	+	+
Setup im agilen Umfeld	+	gering - mittel	gering	-		+

SW-Entwicklungsmodelle im Vergleich

	Projekt		
	Komplexität	**Kritikalität**	**Umfang**
Wasserfall (sequentiell)	gering-mittel	gering-hoch	gering-mittel
Unified Process (interativ/ inkrementell)	mittel-hoch	mittel-hoch	mittel-hoch
XP / Scrum (agil)	gering-mittel	gering	gering-mittel

SW-Entwicklungsmodelle im Vergleich

	Software Engineering	
	Software Specification	**Software Design**
Wasserfall (sequentiell)	-	-
Unified Process (interativ/ inkrementell)	++	++
XP / Scrum (agil)	o	o

SW-Entwicklungsmodelle im Vergleich

	Kommunikation, Koordination, Steuerung			
	dokumenten-basiert	**direkte, persönliche Kommuni-kation**	**Wissens-transfer**	**Change Request Handling**
Wasserfall (sequentiell)	o	--	dokumenten-basiert	--
Unified Process (interativ/ inkrementell)	+	+	dokumenten-basiert / persönlich	+
XP / Scrum (agil)	-	++	persönlich	++

Abbildung 32: Modell Gesamtsicht

9 Schlussbetrachtung

„Das ist nicht das Ende, es ist auch nicht der Anfang vom Ende, aber vielleicht ist es das Ende des Anfangs.“, Sir Winston Churchill

Als ich zu Beginn meiner Überlegungen an einer Mind-Map arbeitete, wusste ich bereits genau worauf ich hinaus wollte, oder auch wieder nicht. Im Laufe der Zeit nahm die erwähnte Mind-Map unermessliche Ausmaße an, deren Inhalt aus meiner Sicht unbedingt umgesetzt werden musste. Die Abgrenzung des Themas schien schwierig und war bis knapp vor Fertigstellung dieser Abhandlung Thema. Mein Ziel habe ich trotz aller Euphorie erreicht, obgleich nicht alle ursprünglich geplanten Aspekte ihre Berücksichtigung finden konnten. Ein Lerneffekt hat sich eingestellt, die Abgrenzung frühzeitig und realistisch festzulegen.

Das ist es auch auf was ich in meinem kurzen Conclusio eingehen möchte und damit vielleicht dem/ der Einen oder Anderen einen Denkanstoß für eine weiterführende Behandlung des Themas geben kann.

Beweggrund Nummer 1 für IT-Offshoring ist das Kosteneinsparungsmotiv. Eine ausführliche Diskussion der Einsparungspotentiale und versteckten Kostenfallen im IT-Offshoring kam in dieser Abhandlung zu kurz. Beispielsweise fallen in der Anlaufphase eines Projekts oder strategischen Partnerschaft Kostenblöcke an. Während der Zusammenarbeit ist mit höheren Koordinationskosten zu rechnen.

Zusammenarbeitsmodelle im Zusammenhang mit Software-Entwicklungsmodellen bedingen gegebenenfalls besonderer Vertragsformen. Beispielsweise können Fixpreiskontrakte in Verbindung mit agilen Methoden schwer vereinbart werden.

Kulturelle Aspekte in der Zusammenarbeit mit Unternehmen weit entfernter Länder sind ebenso spannend wie zwischen verschiedener Unternehmenskulturen selbst. Eine Betrachtung dieses Aspekts kam in dieser Arbeit zu kurz.

Wie kann eine Einbindung von IT-Offshore Development in die IT-Governance erfolgen? Welche Änderungen in der Organisation müssen durchgeführt werden, um optimal für IT-Sourcing gewappnet zu sein?

Vielleicht kann eine qualitative oder quantitative Studie zu diesem Thema weitere Erkenntnisse bringen.

Das Ende ist der Anfang.

Literaturverzeichnis

Amberg, M. et al. (2005): „Kritische Erfolgsfaktoren für Offshore Softwareentwicklungsprojekte“. In:
http://www.competencesite.de/downloads/29/5f/i_file_4102/studie_offshoresoftwareentwicklungsprojekte_unierlangen.pdf 28.8.2010

Amberg, M. et al. (2006): IT-Offshoring; Physica-Verlag, Heidelberg.

Beck, K. (1999): „Extreme Programming Explained“; Addison-Wesley, Boston.

Boczan, O. (2011): „Vorlesung IT-Projektmanagement“ In:
http://www.cs.hm.edu/die_fakultaet/ansprechpartner/lehrbeauftragte/boczan/vorlesung/projektmanagement/vorlesungsskripte.de.html 12.09.2012

Boehm, B. et al. (2003): „ Observations on Balancing Discipline and Agility”. In: Agile Development Conference, 2003. ADC 2003. Proceedings of the 25.-28.6.2003
http://ieeexplore.ieee.org/ 25.09.2011

Böhm, C. (2004): „Managing Cost Factors for Offshore IT Projects”. In:
http://www.competence-site.de/downloads/d6/62/i_file_4031/TransCrit-Offshore-Cost-2.0.pdf 29.8.2011

Bundesministerium für Verteidigung (1997): „Entwicklungsstandard für IT-Systeme des Bundes. Vorgehensmodell; Teil 3: Handbuchsammlung“. Allgemeiner Umdruck Nr. 250/3
http://www.v-modell.iabg.de/ 25.09.2011

Bullinger, H.-J. (1999): „Effizientes Informationsmanagement in dezentralen Organisationsstrukturen“; Springer-Verlag, Berlin.

Bunse, C. et al. (2008): “Vorgehensmodelle kompakt”; Spektrum Akademischer Verlag, Heidelberg.

CMMI Product Team (2010): “CMMI for Development, Version 1.3.”; Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Cohn, M. (2010): “Agile Softwareentwicklung. Mit Scrum zum Erfolg”; Addison-Wesley, München.

Cunningham, W. et al. (2001): “Agile Manifesto. Manifesto for Agile Software Development”. http://www.agilemanifesto.org/ 10.06.2012

Da Rold, C. et al. (2009): “Gartner Keynote: The European Outsourcing and IT Services Scenario: Insight for a Turbulent Market”. Outsourcing & IT Services Summit 15.-16.6.2009

Deutsche Bank Research – Offshoring Report 2005, Ready for take off, Nr.52 In: BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.;
http://www.bitkom.org/files/documents/OffshoringReport2005_BITKOM_DBR_ECO52.pdf 26.08.2010

Ebert, C. (2006): „Outsourcing kompakt: Entscheidungskriterien und Praxistipps für Outsourcing und Offshoring von Software-Entwicklung“; Spektrum, Akademischer Verlag; München.

Eckstein, J. (2009): „Agile Softwareentwicklung mit verteilten Teams“; dpunkt.Verlag, Heidelberg.

Eveleens, J.L. et al. (2008): “The rise and fall of the Chaos report figures” In:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.7918&rep=rep1&type=pdf 25.09.2011

Faiz, M. et al. (2007): „Offshore Software Development Models“; Information and Emerging Technologies, 2007. ICIET 2007. International Conference on 6.-7.7.2007 http://ieeexplore.ieee.org/ 1.10.2010

Gadatsch A. (2006): „IT-Offshore realisieren“; Vieweg Verlag; Wiesbaden.

Hastie, J. (2008): “Project Classification” In: http://www.softed.com/resources/Docs/ProjectClassification.pdf 08.07.2012

Hruschka ,P. et al. (2009): „Agility kompakt. Tips für erfolgreiche Systementwicklung.“; Spektrum Akademischer Verlag, Heidelberg.

IEEE Std 610.12 (1990): “IEEE Standard Glossary of Software Engineering Terminology”, IEEE Standard Association.

Jacobson, I. et al. (1999): “The Unified Software Development Process”; Addison-Wesley, Boston.

Jensen, M. et al. (2007): „Managing Offshore Outsourcing of Knowledge-intensive Projects - A People Centric Approach”. In: Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on 27.-30.8.2007 http://ieeexplore.ieee.org/ 29.08.2010

Kannaley, M. (2010): „SDLC 3.0; Beyond a tacit understanding of agile“; Fourth Medium Press, San Jose.

Ludewig, J. et al. (2010): „Software Engineering“. Dpunkt.Verlalg, Heidelberg.

Mertens, P. (2005): „Die (Aus-)Wanderung der Softwareproduktion – Eine Zwischenbilanz“, Arbeitsberichte des Instituts für Informatik, Band 38, Nummer 3, Friedrich-Alexander- Universität; Erlangen-Nürnberg

Moczadlo, R. (2005): „Softwaredevelopment deutscher Unternehmen in Indien und Nearshore im Vergleich“. In: http://www.competence-site.de/downloads/67/a9/i_file_4086/Softwaredevelopment_Moczadlo.pdf 29.8.2011

Munassar, N. et al. (2010): “A Comparison Between Five Models Of Software Engineering”. In: IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010
http://www.ijcsi.org/papers/7-5-94-101.pdf 10.06.2012

Nicklisch, G. et al. (2009): „Offshoring; Globale Projekte: Kulturdifferenzen minimieren“. In: iX KOMPAKT 03/2009 IT-Projekte; Heise Zeitschriften Verlag; Hannover.

Paguette, L. (2004): „The Sourcing Solution: A Step-By-Step Guide to Creating a Successful Purchasing Program; Amacom, New York.

Royce, W. (1970): “Managing the Development of large Software Systems”; IEEE WESCON, proceedings on
http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf 01.10.2011

Royce, W. (1998): “Software Project Management, A Unified Framework”; Addison-Wesley, Boston.

Schwaber, K. (2004): “Agile Project Management with Scrum”; Microsoft Press, Redmond.

Schwarze L. et al. (2005): „IT-Outsourcing- Erfahrungen, Status und zukünftige Herausforderungen“ in: Strahringer S. (Hrsg.) et al.: Outsourcing; dpunkt.verlag, Heidelberg.

Sommerville, I. (2011): “Software Engineering”; Pearson, Boston.

Sutherland, J. et al. (2007); “Scrum and CMMI Level 5: The Magic Potion for Code Warriors.” AGILE 2007. International Conference on 13.-17.8.2007.
http://ieeexplore.ieee.org/ 28.09.2011

The IT Governance Institute (2005): “CObIT 4.0 Control Objectives, Management Guidelines, Maturity Model”, Rolling Meadows.

Thompson, K. (2012): “When to Use Scrum for software projects?” In: http://www.cprime.com/community/articles/whentousescrum.html 10.08.2012

Weber, M. (2005): BITKOM Leitfaden Offshoring. In: BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.; http://www.bitkom.org/files/documents/BITKOM_Leitfaden_Offshoring_31.01.2005.pdf 25.08.2010

Weber, M. (2008): „Terminologie Outsourcing: Vorschlag zur Vereinheitlichung von Begriffsinhalten im Outsourcing Umfeld.“ In: BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.; http://www.bitkom.org/files/documents/Leitfaden_Outsourcing-Terminologie.pdf 25.08.2010

Weber, M. (2009): “Cloud Computing -Evolution in der Technik, Revolution im Business - BITKOM-Leitfaden.” In: BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.; http://www.bitkom.org/files/documents/BITKOM-Leitfaden-CloudComputing_Web.pdf 25.08.2010

Wells, Don (2009): „Extreme Programming: A gentle introduction”. http://www.extremeprogramming.org/ 10.06.2012

Anhang

Abstract

IT Software-Entwicklung ist erwachsen geworden und steht vor der Herausforderung der Industrialisierung der IT. Trotz aller Versuche durch Methoden und Tools die Qualität von Software-Entwicklung zu erhöhen, krankt es nach wie vor an der erfolgreichen Umsetzung von Projekten.

Unternehmen stehen verstärkt unter Druck die Kosten zu senken. Deshalb wird „IT-Sourcing“ mittlerweile ein hoher Stellenwert beigemessen. Dabei geht es grob gesagt darum, interne Ressourcen und externe Bereitstellung von Dienstleistungen optimal auszubalancieren. IT-Entscheider erhoffen sich dadurch einerseits auf günstigere Arbeitskräfte aus Billiglohnländern zuzugreifen, andererseits bei Bedarfsspitzen angemessen skalieren zu können.

Diese der Arbeit beschäftigt sich mit dem Gebiet von IT-Sourcing bzw. IT-Offshore Software Development. Nach einer begrifflichen Einordnung folgt eine Kategorisierung der verschiedenen Offshoring-Konzepte. In weiterer Folge werden diese Konzepte nach deren Relevanz beurteilt und eingegrenzt.

Folgend werden grundlegende Organisationsformen und Zusammenarbeitsmodelle im Offshore Development identifiziert und im operativen Kontext beleuchtet. Um eine objektive Darstellung zu gewährleisten, werden diese anhand von kritischen Erfolgsfaktoren in Relation gesetzt.

Der zweite Teil der Arbeit beschäftigt sich mit Software Engineering und Software-Entwicklungsmodellen. Es werden ausgewählte Software-Entwicklungsmodelle betrachtet, die in der Praxis gerne angewendet werden. Jedes Modell wird in Bezug auf den Offshoring-Kontext diskutiert und deren Anwendbarkeit beurteilt.

Abschließend wird ein Modell erarbeitet, um es einem Unternehmen zu ermöglichen, eine optimale Konstellation aus Delivery-Modell, Kollaborationsmodell und Software-Entwicklungsmodell zu bestimmen. Dabei wird besonderes Augenmerk auf den Projektkontext und Unternehmenskontext gelegt.

Keywords

Offshoring, Offshore, Nearshore, Sourcing, Outsourcing, Software Delivery, Kollaboration, Software Engineering, Software-Entwicklung, Software Development, Vorgehensmodell, Prozessmodell, Software Entwicklungsmodell, Wasserfall, Unified Process, Extreme Programming, Scrum, agile Methoden

Lebenslauf

Name	Gerald Köhl
Nationalität	Österreich
Geburt	12.6.1973; Ried im Innkreis; Oberösterreich

Akademische Laufbahn

2012	Magisterarbeit an der Fakultät für Informatik, Institut für Knowledge and Business Engineering, Workflow Systems and Technology.
2003, 2006	Magisterstudium Betriebswirtschaft am BWZ der Universität Wien, Vertiefungsfach Management.
2003	Verleihung des akad. Grads (Bakk. rer. soc. oec.) durch Bologna Reform
1995 - 2000	Diplomstudium der Betriebswirtschaft am BWZ der Universität Wien mit BBWL Wirtschaftsinformatik am Lehrstuhl für Knowledge Engineering BBWL Organisation am Lehrstuhl für Organisation und Planung

Berufserfahrung

2010 – heute	A1 Telekom Austria AG, Wien
2001 – 2010	mobilkom austria AG, Wien
1998 – 2001	Erste Bank Informationstechnologie, Wien