

## Software-Entwicklung in Zeiten von SOA Zuverlässige Aufwandschätzung in agilen Teams

von Michael Aubermann

Das Rückgrat der Zeitplanung in Projekten ist die Aufwandschätzung. Dafür verwendete Verfahren, wie z.B. Function Point Analysis oder Predictive Object Points greifen allerdings nur, wenn weit reichende Spezifikationen des angestrebten Projektergebnisses zur Verfügung stehen. Diese zu erzeugen, ist Ziel in konventionellen Vorgehensmodellen. Projekte in der Software-Entwicklung laufen jedoch typischerweise in einem Zustand weitgehender Unwissenheit bzw. Unsicherheit ab, mit einem permanenten Lernprozess im Projektverlauf.



**Michael Aubermann**

Diplom-Volkswirt, Mitglied des Vorstands der Ropardo AG, über 20 Jahre unternehmerischer Erfahrung im EDV-Sektor

Kontakt:

[michael.aubermann@ropardo.de](mailto:michael.aubermann@ropardo.de)

Mehr Informationen unter:

[www.projektmagazin.de/autoren/](http://www.projektmagazin.de/autoren/)

Benutzt man die genannten Schätzmodelle unter solchen unsicheren Bedingungen und in Zeiten beschleunigten Wandels, wie das z.B. in der serviceorientierten Architektur (SOA) und Web 2.0 der Fall ist, wird der Aufwand regelmäßig unterschätzt. Das führt zu Qualitätsverlust und Fehlentwicklungen, die am Bedarf des Kunden vorbeigehen und deshalb nicht akzeptiert werden – und letztendlich zum Scheitern des Projekts.

Alternativ lassen sich für eine relativ große Klasse von Software-Projekten agile Vorgehensmodelle verwenden. Diese werden jedoch noch kaum akzeptiert, da sie oft anstelle klarer Regeln und Prozesse nur den Begriff der "kollaborativen Werte" setzen (The Agile Manifesto, 2001).

Dieser Beitrag soll Ihnen aufzeigen, wie man mit einer agilen Vorgehensweise – auch bei räumlich verteilten Teams unter Outsourcing-Bedingungen – zuverlässige Aufwandschätzungen und Projektpläne erhalten kann. Einzige Voraussetzung ist eine verbindliche und verantwortungsvolle Projektkultur.

### Entwickeln in Zeiten von SOA und Web 2.0

Software-Entwicklungsprojekte beinhalten – unabhängig von den konkreten Zielen – immer den Übergang von einer vagen Idee zu einem konkreten Produkt. Dabei ist eine zutreffende und realistische Aufwandschätzung notwendig, um die Projektarbeiten zielorientiert und risikobewusst steuern zu können. Da der Markt von den Unternehmen immer mehr Flexibilität erwartet, müssen auch die IT-Lösungen, insbesondere Softwarelösungen diesen Anforderungen entsprechen. Besonders gut lässt sich diese Entwicklung in dem derzeit vorherrschenden Architekturparadigma der serviceorientierten Architektur (SOA) nachvollziehen. Die IT-Lösungen von heute sind eine permanente Baustelle, in der Altes und Neues entwickelt, verbunden, auseinander gerissen und wieder zusammengesetzt wird, ganz nach den Geschäftsanforderungen.

Außerdem sind die Grenzen der Organisationen ständig in Bewegung. Kooperative Strukturen bilden sich heraus, Partnerschaften werden eingegangen und wieder gelöst, Unternehmen werden gegründet, fusionieren oder lösen sich auf.

Dieses pulsierende Geschehen hat auch zu dem Phänomen "Web 2.0" geführt. Dabei handelt es sich um eine Sichtweise, die immer stärker die Art und Weise diktiert, wie im Internet Anwendungen bereitgestellt werden. Aussagen wie "gemeinschaftliche Nutzung kollektiver Intelligenz" oder "Schaffung von Mehrwert für alle Teilnehmer" (Högg, Meckel, 2006) klingen zunächst vielleicht übertrieben, dennoch gibt es immer mehr erfolgreiche Geschäftsmodelle mit diesem Ansatz. Web 2.0 hat den Gedanken der Heterogenität von Systemen salonfähig gemacht. Die Plattformen für Web 2.0 erlauben es, eine Vielzahl unterschiedlicher Endgeräte und Programme zu kombinieren, indem sie diese als Dienste integrieren. Dazu passen keine langen Release-Zyklen und lokal installierte Applikationen mehr. Alles wird flüchtiger und ist ständig in Bewegung.

Viele Software-Projekte sind direkt oder indirekt von diesen Entwicklungen betroffen. Zum einen sind immer häufiger serviceorientierte Architekturen zu bedienen, zum anderen wirken die Ideen und der Takt von Web 2.0 als Paradigmenwechsel auch in den Köpfen der beteiligten Menschen – und zwar bei Anwendern

und Entwicklern gleichermaßen. Verkürzte Anwendungs-Lebenszyklen, die von mehreren Jahren auf wenige Monate zurückgegangen sind, erhöhen die Unsicherheit in Bezug auf den Umfang und die Qualität der benötigten Funktionalität.

Bisher erfolgreiche Prozessmodelle mit starker Betonung der Phasenübergänge können diesen Herausforderungen nur schwer begegnen. So geht beispielsweise das Wasserfallmodell davon aus, dass sich Anforderungen bereits in der Planungsphase präzise beschreiben lassen. Dabei wird außer Acht gelassen, dass sich die Anforderung während des Projektverlaufs ändern können und sich die Domänenkenntnisse mit zunehmendem Detaillierungsgrad erhöhen. Das Wasserfallmodell wurde bereits in den 70er Jahren kritisiert, spielt jedoch vor allem in der Lehre, aber auch in der allgemeinen Vorstellung von Auftraggebern immer noch eine dominante Rolle.

Verfeinerte phasenbetonte Ansätze finden sich vor allem in Vorgehensmodellen, die sich schwerpunktmäßig auf die Bereitstellung eines Ordnungs- und Dokumentationsrahmens für die Projektdurchführung konzentrieren. In der Regel stellen diese Modelle umfangreiche Methodensammlungen zur Verfügung, die durch Maßschneidern (Tailoring) auf den benötigten Umfang zurechtgestutzt werden sollen. Beispiele dafür sind das V-Modell mit seinen Modifikationen oder sog. schlanke Vorgehensmodelle wie der Rational Unified Process. Diese theoretisch sehr ansprechenden Ansätze begegnen der wachsenden Dynamik der anfordernden Märkte durch eine massive Verstärkung der Kontroll- und Steuerungsmechanismen bei der Projektdurchführung. Vor dem Hintergrund der aktuellen Entwicklungen werden viele Projekte zu klein sein, um den damit gestiegenen Verwaltungsaufwand zu erwirtschaften.

## Der Markt für Softwareentwicklung

Unternehmen, die auf professioneller Basis Software entwickeln möchten, stehen unter enormem Konkurrenzdruck, der zunehmend über den Preis ausgeübt wird – angeheizt durch das Lohngefälle in Offshore- und Nearshore-Ländern. Sie sind deshalb gezwungen, nach anderen Merkmalen zu streben, die neben dem Preis als Auswahlkriterium bei der Vergabe von Entwicklungsaufträgen dienen können. Als Möglichkeiten bieten sich hier Qualität und Pünktlichkeit an.

Die Qualität eines Software-Produkts lässt sich jedoch nicht im voraus beurteilen, da dieses ja erst hergestellt werden muss. Zudem ist das Spezifizieren von Qualitätseigenschaften für Auftraggeber unattraktiv. Die Anforderungen unterliegen kurzfristigen Schwankungen und Veränderung und die Auftraggeber möchten das Risiko einer "falschen Anforderung" keinesfalls alleine tragen.

Dazu kommt, dass vielen Unternehmen die Fähigkeit abhanden gekommen ist, ihre Lieferanten hinsichtlich der Kompetenz in der Software-Entwicklung zu beurteilen. Durch Verlagerung der Entwicklungsaktivitäten außerhalb der Unternehmensgrenzen (Outsourcing-Maßnahmen) versuchen die Unternehmen häufig, der Unsicherheit in Software-Projekten beizukommen. Je nach Schnittführung wird dabei die IT-Abteilung von sämtlichen Programmierfähigkeiten befreit oder die Arbeit wird so uninteressant, dass viele fähige Entwickler freiwillig das Unternehmen verlassen. Diese Experten konnten früher zwischen den Fachabteilungen und den Anbietern von Entwicklungs-Dienstleistungen vermitteln und die Auswahl der Lieferanten durch gezieltes Hinterfragen von Termin- und Aufwandsaussagen verbessern.

Stattdessen versuchen die Unternehmen, das Entwicklungsrisiko mit engen Terminen und niedrigen Budgets zu minimieren. Demzufolge wird die Pünktlichkeit oder Termintreue zum weiteren Erfolgsfaktor. Die Auswahl der Lieferanten folgt dabei einem einfachen Prinzip: Erweist sich der Lieferant als untauglich, wird das nächste Projekt an ein anderes Unternehmen vergeben.

Erfolgreiche Projekte halten also Termine und Budgets ein und liefern die vereinbarte Funktionalität in einwandfreier Qualität. Aufwandschätzung wird somit zu einem der Schlüsselfaktoren für den wirtschaftlichen Erfolg in Software-Unternehmen.

## Ein typisches Software-Projekt

Ein Unternehmen beauftragt einen Software-Dienstleister oder die interne IT-Abteilung mit der Entwicklung einer Anwendung. Der Auftragnehmer setzt dieses Vorhaben in einem Projekt um, das oft wie im Folgenden beschrieben abläuft.

## Spezifikation

Der Projektleiter erstellt zusammen mit dem Kunden eine Spezifikation. Er führt die Kommunikation mit dem Kunden und hat in der Regel einen betriebswirtschaftlichen Hintergrund sowie allgemeine Erfahrung im Projektmanagement. Oft hat er nur geringe Erfahrung im Programmieren, manchmal auch keine Erfahrung im Projektmanagement.

## Architektur und Entwurf

Der Projektleiter erstellt nun eine detaillierte Spezifikation und die Architektur sowie den Entwurf des zukünftigen Systems. Häufig ist daran auch ein Architekt oder Experte für den Systementwurf beteiligt. Viele dieser Software-Architekten haben entweder schon lange nicht mehr programmiert oder keine Programmiererfahrung, da sie sich auf den Entwurf von Software spezialisiert haben. Das so entstandene "Design" muss dann unter Zuhilfenahme von geeigneten "Tools" von den Programmierern nur noch "herunterprogrammiert" werden.

Die dafür verwendeten, oft sehr teuren Modellierungs-Tools erzeugen auf Knopfdruck die Entwurfs- und Systemdokumentation und generieren den Code, auf dem die Programmierer aufsetzen können. Damit kann der Software-Dienstleister günstige Programmierer beschäftigen, die als Seiteneinsteiger ihr Wissen oft in mehr oder weniger guten Umschulungsmaßnahmen erworben haben und nur über wenig Erfahrung verfügen.

## Aufwandschätzung

Nachdem die Spezifikation und der Entwurf eingefroren wurden (sie bilden ja die Grundlage des vertraglichen Leistungsumfangs), muss ein Zeitplan erstellt werden. Dazu benötigt man die Aufwandsabschätzung. Zu diesem Zeitpunkt stehen der Zeitrahmen für die Realisierung und das verfügbare Budget meist schon als unveränderbare Randbedingungen fest.

Die Projektleitung muss also bei der Zeitplanung auch festlegen, welche Anforderungen in dem gesetzten Zeit- und Budgetrahmen realisiert werden müssen und welche realisiert werden können. Da ihr für eine realistische Schätzung, wie gesagt, die technische Erfahrung fehlt, wird diese durch mehr oder weniger systematisches Raten ersetzt: Es wird ein Aufwand "geschätzt", der sich am verfügbaren Budget, den Kosten des Entwicklungsteams, einer Gewinnmarge und dem gewünschten Zeitraum orientiert. Der zu liefernde Funktionsumfang ist eine weitere Randbedingung für die Schätzung. Bild 1 stellt die Anatomie eines solchen Projekts dar.

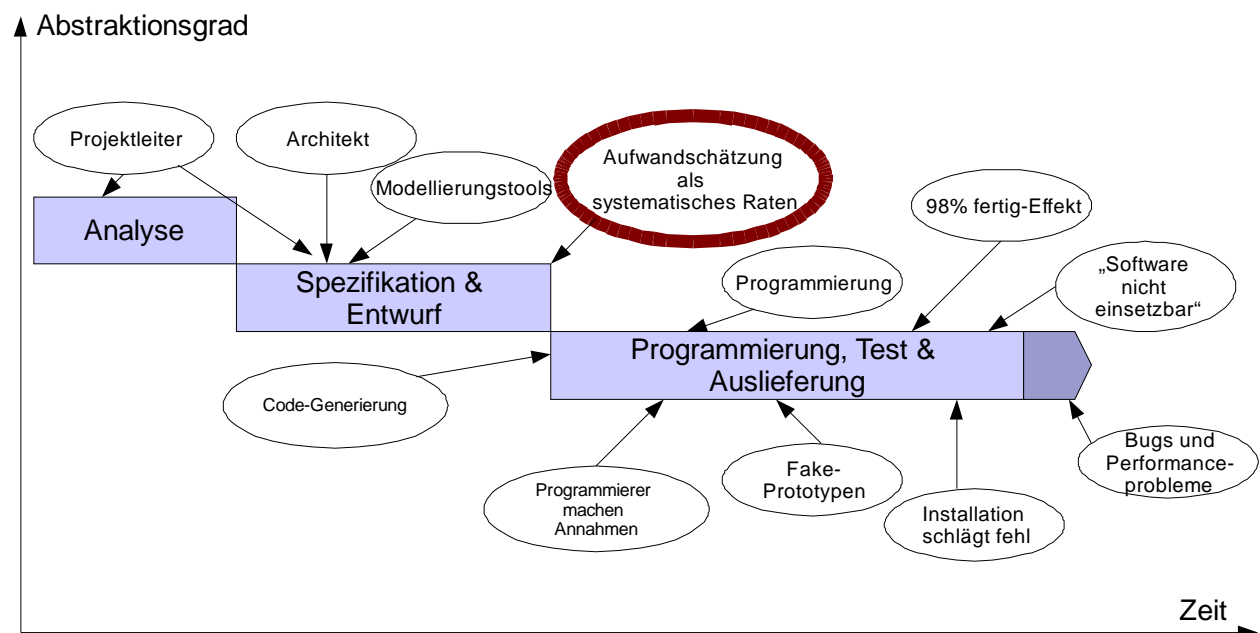


Bild 1: Anatomie eines Softwareprojekts (ähnlich: Richter 2003, S. 28).

## Programmierung

Der Projektleiter arbeitet das Ergebnis der Aufwandschätzung in einen Zeitplan ein und präsentiert diesen dem Auftraggeber. Danach treten die Programmierer zum ersten Mal in Aktion. Sie erhalten dazu die Spezifikation, den Entwurf und den vielleicht von einem Tool generierten Code. Schon bald stellen sie fest, dass die Spezifikation nicht komplett und das Design nicht ganz korrekt ist. Da sie keinen direkten Kontakt zum Kunden haben, treffen sie Annahmen, die die Spezifikationslücken füllen sollen. Diese stimmen meistens nicht mit den Anforderungen des Auftraggebers überein. Auch der Auftraggeber kann inzwischen weitere Anforderungen entdeckt haben, die aber nicht Teil der ursprünglichen Spezifikation sind. Oft unterstellt er sogar, dass diese Anforderungen implizit und automatisch berücksichtigt sind, da sie wesentlicher Bestandteil seiner Fachwelt sind. Darum weist auch niemand die Projektleitung auf Spezifikationsmängel hin.

### Änderungen während der Entwicklung

Der Projektleiter hat drei Möglichkeiten, auf neue Anforderungen – nach Einfrieren der Spezifikation – zu reagieren:

- Kostenlos in den Plan aufnehmen
- Zusatzbudget fordern
- Ignorieren

Im Verlauf des Projekts wird er je nach Zeitpunkt eine der drei Möglichkeiten wählen. Anfangs werden zusätzliche Anforderungen kostenlos aufgenommen. Dann wird zusätzliches Budget angefordert, da ja schon genügend Zugeständnisse gemacht wurden. Lässt sich kein Budget mehr einfordern, werden Anforderungen schlicht ignoriert.

Den Programmierern werden neue Anforderungen auf zwei Arten kommuniziert:

- in Form einer geänderten oder ergänzten Spezifikation
- als Mitteilung in Form einer E-Mail, die die Beschreibung der Anforderung enthält

Im ersten Fall müssen die Programmierer die gesamte Spezifikation erneut lesen, um herauszufinden, was zu tun ist. Im zweiten besteht die Gefahr, dass die E-Mail mit den Anforderungen aus den Augen verloren wird. In keinem der Fälle wird jedoch systematisch die Zeitplanung aktualisiert. Eine realistische Schätzung des Zusatzaufwands findet nicht statt und die zeitlichen Randbedingungen der ursprünglichen Planung dürfen nicht verändert werden. Die Programmierer können nun wählen:

- in der gleichen Zeit mehr arbeiten oder
- den Abgabetermin ignorieren

Da sich Kopfarbeit nicht beliebig komprimieren lässt, werden sich über kurz oder lang alle für das Ignorieren entscheiden, dies jedoch dem Projektleiter nicht mitteilen, da sie von diesem ohnehin kein Verständnis dafür erwarten.

### Erfolgreiches Scheitern

Der Rest ist schnell erzählt. Der Projektleiter kontrolliert den Fortschritt, indem er die Programmierer nach dem Prozentsatz der Fertigstellung fragt. Diese Werte gibt er dann in ein Projektmanagement-System oder in Excel ein. Er bekommt ab einem gewissen Zeitpunkt fast immer die Standardantwort "Wir sind zu 98% fertig, aber man kann noch nichts sehen". Um den Auftraggeber in Sicherheit zu wiegen, werden sogenannte Fake-Prototypen produziert. Der Projektleiter übt vor der Präsentation, welche Knöpfe er drücken darf. Der Auftraggeber darf deshalb nicht selbst probieren.

Schließlich kommt es zur Auslieferung. Im Pilotbetrieb wird massiv daran gearbeitet, Fehler zu beheben, die Performance zu tunen usw. Diese Aktivitäten waren zwar alle vorhersehbar, aber keinesfalls geplant. Die so entstehenden Kosten werden, je nach Gesinnung des Projektleiters, aus zusätzlich erkämpften Budgets gedeckt oder in anderen Projekten versteckt. Der Pilotbetrieb dauert sehr lange und schließlich geht das System in Produktion.

Ein solches Projekt ist zwar abgeschlossen, aber ist es auch erfolgreich? Haben die beteiligten Parteien verantwortlich gehandelt? Im Folgenden werden alternative Ansätze für erfolgreiche Software-Projekte entwickelt.

## Was ist eigentlich schief gelaufen?

Neben vielen kleineren Schwächen, die sehr zutreffend in den Anti-Patterns von William Brown et al. nachzulesen sind (Brown 1998), hat das beschriebene Projekt zwei große Probleme:

1. Die Verantwortlichkeiten sind so verteilt, dass wichtige Entscheidungen von Personen getroffen werden, die nicht über hinreichende Erfahrung verfügen.
2. Es ist keine vernünftige Strategie für den Umgang mit Änderungen ersichtlich.

Der deutsche General und Militärtheoretiker Carl von Clausewitz schrieb bereits 1832: "Es ist alles im Kriege sehr einfach, aber das Einfachste ist schwierig. Diese Schwierigkeiten häufen sich und bringen eine Friktion hervor, die sich niemand richtig vorstellt, der den Krieg nicht gesehen hat."

Auch wenn Kriegsführung absolut nichts mit Software-Entwicklung zu tun hat, lässt sich die Idee der Friktion leicht in diesen Kontext übertragen. Im Laufe eines Projekts summieren sich unzählige kleine Hindernisse zu einer bemerkenswerten Friktion. Dieser Effekt ist in der Aufwandschätzung nur enthalten, wenn die schätzenden Personen über eigene Erfahrungen in vergleichbaren Situationen verfügen.

Carl von Clausewitz drückte das folgendermaßen aus: "Man denke sich einen Reisenden, der zwei Stationen am Ende seiner Tagereise noch gegen Abend zurückzulegen denkt, vier bis fünf Stunden mit Postpferden auf der Chaussee; es ist nichts. Nun kommt er an der vorletzten Station an, findet keine oder schlechte Pferde, dann eine bergige Gegend, verdorbene Wege, es wird finstere Nacht, und er ist froh, die nächste Station nach vielen Mühseligkeiten erreicht zu haben und eine dürftige Unterkunft zu finden. So stimmt sich im Kriege durch den Einfluss unzähliger kleiner Umstände, die auf dem Papier nie gehörig in Betrachtung kommen können, alles herab, und man bleibt weit hinter dem Ziel."

Die zu Beginn genannte Forderung nach Flexibilität macht es notwendig, dass während der Entwicklungszeit neue oder geänderte Anforderungen berücksichtigt werden. Anforderungen werden damit zu beweglichen Zielen. Der Planungsprozess muss diesen Umstand berücksichtigen.

## Agilität

Agilität ist die Fähigkeit, mit Änderungen umzugehen. Agile Vorgehensmodelle, wie sie seit Ende der 90er Jahre immer häufiger vorgestellt werden, betonen die Notwendigkeit von Änderungen. Die wohl bekannteste agile Methode ist das Extreme Programming (Beck 1999).

Das Agile Manifesto fordert (The Agile Manifesto 2001):

- Individuen und Interaktionen wiegen mehr als Prozesse und Tools.
- Funktionierende Programme wiegen mehr als ausführliche Dokumentation.
- Die stetige Zusammenarbeit mit dem Kunden hat Vorrang gegenüber stetiger Vertragsverhandlungen.
- Reagieren auf Änderungen statt Einhalten eines festgelegten Plans.

Die in der Aufzählung gesetzten Schwerpunkte mögen die Alltagserfahrung bei der Software-Entwicklung widerspiegeln. Es ist jedoch nicht hilfreich, auf das Planen zu verzichten, nur weil es schwierig ist und Disziplin erfordert. Und Prozesse sind dann sinnvoll, wenn sie bei der Durchführung der Arbeit unterstützen, zum Beispiel, indem einfache Pläne ständig aktuell gehalten werden. Eine ausgewogene Beachtung beider Seiten ist daher sinnvoller.

Auftraggeber verlangen verständlicherweise verbindliche Aussagen zu Kosten und Zeitbedarf. Oft sind die Entwicklungsprojekte mit anderen Vorhaben verknüpft. Die Zeitfenster sind durch diese Kopplung sehr klein, was Termintreue zu einer wichtigen Qualität werden lässt. Wie also kann man den Aufwand für das System bestimmen?

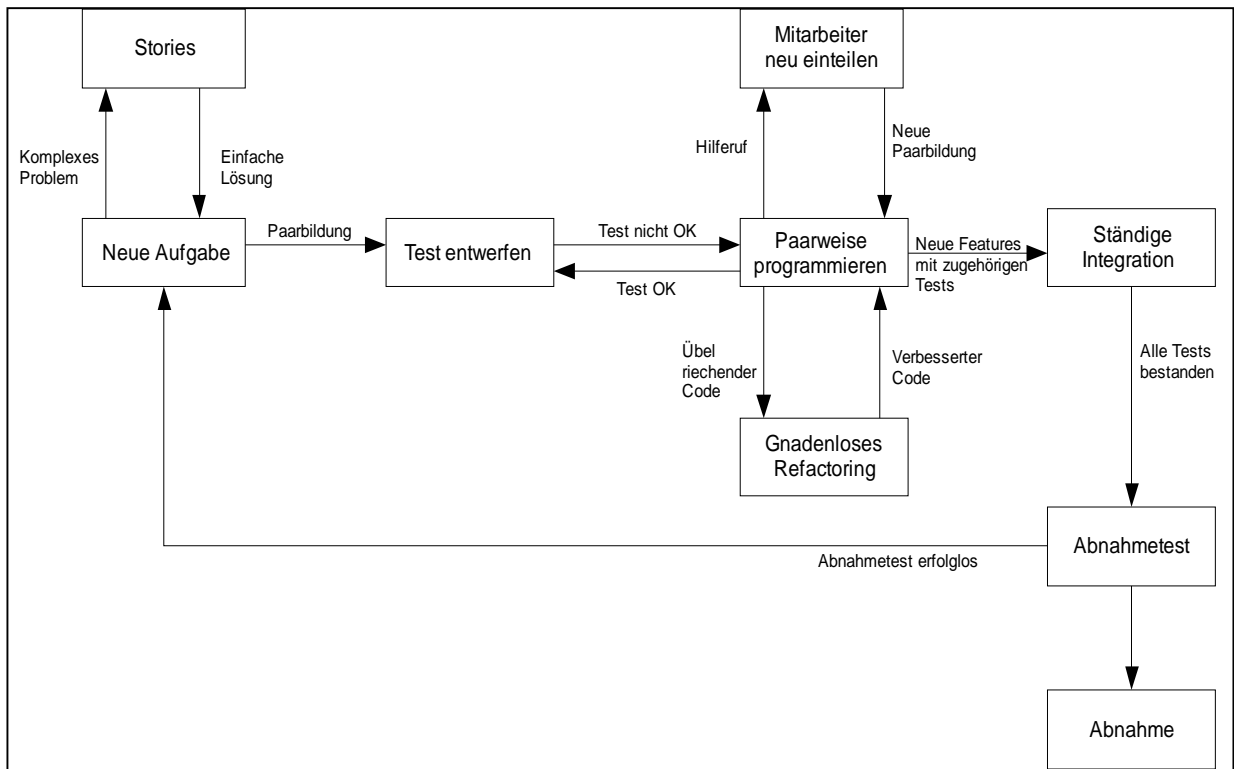


Bild 2: Das Prozessmodell des Extreme Programming.

Hat die Firma sehr viel Erfahrung im Bereich klassischer Schätzverfahren wie Function Point Analysis oder Predictive Object Points und ist das Einsatzgebiet der Software klar umrissen, so können diese Praktiken, auf den Lebenszyklus der Software bezogen, durchaus erfolgreich sein. Bei unsicheren Bedingungen, z.B. bei vagen Vorstellungen des Auftraggebers oder bei neuen Technologien, liefern diese Techniken jedoch meist zu geringe Schätzergebnisse.

Die Verwendung von agilen Prozessmodellen bietet hier eine pragmatische Alternative für eine relativ große Klasse von Software-Projekten. Sie entbindet eine Organisation jedoch nicht von der Pflicht, verbindliche, nachvollziehbare Angaben über zu erwartende Zeit- und Ressourcenanforderungen zu machen. Hier lässt sich oft ein Interessenskonflikt zwischen Management und Programmierern mit einem zum Teil rigorosen aber wirkungslosen Prozess- und Richtliniengerangel beobachten. Dieser Konflikt ist einer der Gründe, warum es so schwer ist, agile Vorgehensweisen zu offiziellen Verfahren in Unternehmen zu erheben. So wird auch verhindert, dass sich diese Verfahren etablieren.

Um die Vorteile der agilen Vorgehensweise in einer verbindlichen Projektkultur nutzen zu können, benötigt man einen intelligenten Plan mit der richtigen Detaillierungsebene. Ein solcher Plan entsteht auf der Basis einer Anforderungsbeschreibung, die in der Fachsprache des Anwenders die benötigten Funktionen beschreibt. Ihn zu erstellen erfordert Erfahrung und Verantwortung auf den richtigen Ebenen. Management und die Software-Entwicklung müssen dabei gemeinsam die Verantwortung für die Aufwandsschätzung und Projektsteuerung übernehmen.

Die Bestandteile des Plans sind

- Entwurfsdokument
- Feature-Liste
- Release-Plan

Die Feature-Liste wird gemeinsam mit dem Auftraggeber entwickelt, der Release-Plan ist eine einfache Verteilung der zu entwickelnden Features auf gleich große Intervalle, an deren Ende immer eine Auslieferung steht.

## Erfahrung statt Raten

Das nun vorgestellte Verfahren haben wir erfolgreich in Projekten zur Entwicklung individueller Software-Lösungen eingesetzt. Die Projektdauer lag dabei zwischen sechs Wochen und zwölf Monaten, die Teamgröße variierte zwischen drei und sechs Personen. Das Verfahren setzt vor allem auf offene Kommunikation mit dem Auftraggeber und auf die Erfahrung der Entwickler im Umgang mit Unsicherheit bei den Anforderungen. Typischerweise kommt es in Festpreis-Projekten mit festem Liefertermin zum Einsatz.

Um es gleich vorweg zu nehmen: Das erfolgreichste Mittel gegen Unsicherheit ist Erfahrung. Die Verantwortung für das Management eines Software-Projekts sollte daher ein erfahrener Programmierer tragen. Erfahrung bedeutet in diesem Zusammenhang, dass dieser Programmierer mindestens fünf Jahre intensiv in mindestens zwei Programmiersprachen aktiv entwickelt hat und immer noch auf Ballhöhe mitentwickelt.

In einem Software-Projekt erstellt diese Person zusammen mit dem Auftraggeber die Spezifikation. Diese besteht aus der Feature-Liste, optional einem Prototypen der Benutzeroberfläche (nur bei Projekten sinnvoll, die eine grafische Benutzeroberfläche benötigen) und einem Entwurfsdokument.

## Das Entwurfsdokument

Das Entwurfsdokument muss die, für die zu entwickelnde Software relevanten Geschäftsprozesse, die technische Architektur und alle nichtfunktionalen Anforderungen beschreiben. Weitergehende Entwurfsdetails wie Datenmodell oder Klassenmodell werden nicht erstellt, da sie einerseits zu flüchtig und andererseits zu detailliert für die Diskussion mit dem Auftraggeber sind.

## Der Prototyp

Für den Auftraggeber ist es hilfreich, wenn er sich ein Bild von der künftigen Software machen kann. Dazu kann ein einfacher Prototyp der grafischen Benutzeroberfläche erstellt werden, in dem Menüs und elementare Bedienkonzepte so wie typische Layouts der Dialoge sichtbar werden.

## Die Feature-Liste

Die Feature-Liste ist das Ergebnis der Anforderungsanalyse. Ein Feature beschreibt einen funktionalen Sachverhalt mit den Worten und aus Sicht des Anwenders. Es wird also nur das "fachliche Was" ohne die technischen Details beschrieben. Lösungsmöglichkeiten, die der Autor der Feature-Liste eventuell bereits im Hinterkopf hat, gehören nicht auf die Feature-Liste, sondern zur Implementierung.

Features sollten so beschaffen sein, dass der Anwender beurteilen kann, ob das Feature fertig ist. Bei mehrschichtigen Anwendungen ist ein Feature immer ein Durchstich und wird nicht auf eine Schicht begrenzt. Auch sollten sich Features nach Möglichkeit nicht überlappen und zudem möglichst wenige Abhängigkeiten von anderen Features haben.

Ein Feature hat eine Kurzbezeichnung, die möglichst aus einem Nomen und einem Verb bestehen sollte, um den funktionalen Charakter zu betonen. Ferner gibt es eine möglichst vollständige und sorgfältig formulierte Beschreibung. Jedes Feature wird genau einem Programmierer zugewiesen, der die Verantwortung für Pünktlichkeit und Fehlerfreiheit übernimmt. Die Angaben zum geschätzten Aufwand und nach Fertigstellung zum tatsächlich benötigten Aufwand sind ebenfalls fester Bestandteil eines Features.

## Aufwandschätzung

Generell sollte ein Feature so geschnitten sein, dass der zu erwartende Aufwand für die Entwicklung fünf Personentage (PT) nicht übersteigt. Größere Features müssen nach Möglichkeit weiter zerlegt werden. Mit zunehmender Erfahrung wird der Ersteller der Feature-Liste diese Randbedingung immer besser treffen.

Grundsätzlich lassen sich folgende Fälle beim Erstellen von Feature-Listen unterscheiden:

- Bekannte Problemklasse: Erfahrungen aus anderen Projekten liegen vor.
- Unbekannte Problemklasse: Erfahrungen aus anderen Projekten liegen nicht vor.

Für die Aufwandschätzung ist eine bekannte Problemklasse natürlich ideal, da geplanter und realisierter Aufwand aus vorherigen Projekten als Vergleichswerte herangezogen werden können. Da jedes System seine Besonderheiten hat, sollten die Werte nicht unreflektiert übernommen werden.

Handelt es sich um eine unbekannte Problemklasse, besteht der erste Schritt gewöhnlich darin, diese zu einer bekannten Problemklasse zu machen. Dazu sollte sich der Autor des Features im Idealfall soweit in das Problem einarbeiten, dass er grob in der Lage wäre, die Arbeit der zukünftigen Anwender zu verrichten. Mit dieser Erfahrung ist es möglich, gute Feature-Listen zu schreiben und den Aufwand zuverlässig zu schätzen.

In beiden Fällen helfen die folgenden Techniken bei der Aufwandschätzung: das Komplexitäts-Größe-Portfolio und das Schätzspiel, auch Planning Poker genannt. Beide ermitteln den Aufwand indirekt und arbeiten mit relativen statt mit absoluten Maßstäben.

## Komplexitäts-Größe-Portfolio

Aufwand entsteht, entsprechend des Änderungsbedarfs, abhängig von der Größe des Problems und seiner Schwierigkeit oder Komplexität. Jedes Feature kann bezüglich dieser beiden Kategorien beurteilt und klassifiziert werden.

### Beispiele für Größe

- Klein: Eine Sache muss an genau einer Stelle getan werden.
- Mittel: Einige Dinge müssen an einigen Stellen getan werden.
- Groß: Viele Dinge müssen an vielen Stellen getan werden.

### Beispiele für Komplexität

- Niedrig: Die Implementierung ist klar und etwas Ähnliches wurde bereits einmal erledigt.
- Mittel: Konkrete Erfahrungen mit ähnlichen Dingen liegen nicht vor, eine theoretische Lösung ist jedoch bekannt.
- Hoch: Es ist noch keine Idee vorhanden, wie das Feature umgesetzt werden könnte.

Jedes Feature wird in das Portfolio eingeordnet (Bild 3), wobei der Quadrant bestimmt, in welche Schwierigkeitsklasse das Feature fällt. Dabei bedeutet Hellgrau einfache, Mittelgrau mittlere und Dunkelgrau hohe Schwierigkeit.

Mit jeder Schwierigkeitsklasse wird ein Aufwand assoziiert, der von Projekt zu Projekt variieren kann und mit zunehmender Erfahrung immer besser angepasst wird. Durch Einordnung in eine der Klassen kann somit der Aufwand für ein Feature bestimmt werden. Die Technik lässt sich weiter verfeinern, das Grundprinzip ist aber immer das gleiche.

Als Ausgangspunkt, um den Aufwand für ein Projekt zu schätzen, dienen die Feature-Liste und das Entwurfsdokument. Letzteres ist für erfahrene Entwickler bereits ein Maß für die zu erwartende Komplexität, da es den Systemaufbau und den Einsatzbereich beschreibt.

Nun verteilt der Projektleiter die Features der Feature-Liste anhand der vorangehenden Fragen in das Portfolio. Gibt es bereits ein Projekt ähnlicher Größe und ähnlicher Architektur, kann der Aufwand für die einzelnen Problemklassen übernommen werden. Stehen diese Daten nicht zur Verfügung, verwendet man ein Projekt mit ähnlicher Architektur als Ausgangsgröße.

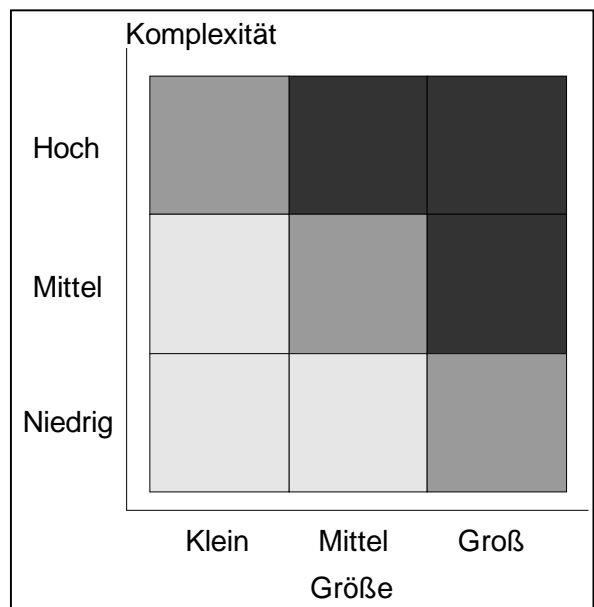


Bild 3: Komplexitäts-Größen-Matrix.

Als nächstes prüft der Projektleiter mit dem Auftraggeber, ob der Aufwand pro Klasse plausibel ist. Dabei helfen ihm die Erfahrungen, die er während der Erstellung der Feature-Liste gesammelt hat. Gegebenenfalls sind hier Anpassungen in beide Richtungen nötig. Als Faustregel gilt: Kürzere Projekte haben niedrigere Werte in den Klassen; kleinere Teams haben bei gleicher Laufzeit ebenfalls niedrigere Werte.

Wenn keine Erfahrungen aus vergleichbaren Projekten vorliegen, ist dieser Ansatz problematisch. Dann hilft das folgende Schätzspiel dabei, den Aufwand für jedes Feature zu bestimmen.

## Schätzspiel (Planning Poker)

Diese Technik basiert auf einem Ansatz, der als "Wideband Delphi" von Barry Boehm entwickelt und 1981 publiziert wurde. Die Spielregeln des Schätzspiels lauten:

1. Jedes Teammitglied erhält ein Kartenset, bestehend aus Karten mit gültigen Schätzwerten.
2. Der Projektleiter stellt ein Feature vor.
3. Jedes Teammitglied wählt den geschätzten Aufwand in Form einer Karte, zeigt sie aber noch nicht.
4. Nun werden alle Karten aufgedeckt, so dass jeder die Schätzwerte auf den Karten sehen kann.
5. Unterschiede in den Schätzungen, insbesondere Ausreißer, werden besprochen.
6. Die Schritte 3 bis 5 werden solange wiederholt, bis die Schätzungen übereinstimmen.

Das Schätzspiel nutzt den Gruppenprozess zur Meinungsbildung. Es funktioniert deshalb so gut, weil die Personen zu Wort kommen, die auch die Arbeit erledigen werden. Alle Meinungen werden gehört und Aufwandschätzungen werden begründet. Somit lässt sich die Zuverlässigkeit deutlich steigern.

Jedes Feature erhält zwei Aufwandswerte: Minimum und Maximum, wobei letzterer den Minimalwert um nicht mehr als 30% übersteigen sollte. Als Minimalwert gilt der auf jeden Fall benötigte Wert, der Maximalwert stellt die obere Grenze dar. Somit beträgt der Puffer maximal 30% des Aufwands. Der Auftraggeber sollte immer den Maximalwert budgetieren.

## Einen verbindlichen Release-Plan erstellen

Der Projektleiter ergänzt im nächsten Schritt die Feature-Liste mit dem ermittelten Schätzaufwand pro Feature. Somit steht dem Auftraggeber eine "Einkaufsliste" zur Verfügung, aus der er die Features auswählt, die realisiert werden sollen.

Um den verbindlichen Release-Plan zu erstellen, wird die zeitliche Lage des Projekts (der Ende-Termin und der nächstmögliche Anfangstermin) vereinbart. Nun wird diese Zeitspanne, unabhängig von den zu realisierenden Features, in mehrere gleich lange Releases eingeteilt. Releases werden durchnummeriert und erhalten feste Anfangs- und Ende-Termine. Werden beispielsweise zwei-Wochen-Releases geplant, so stehen jedem Entwickler 10 Arbeitstage zur Verfügung. Der Projektleiter wird daher Features mit einem Aufwand von Minimum acht und Maximum 11 Tagen einplanen.

Jeder Release-Termin beinhaltet eine Auslieferung an den Auftraggeber. Dieser muss die Software testen und innerhalb einer vorgegebenen Zeit festgestellte Fehler melden. Die Fehler werden in den Release-Plan eingearbeitet und im nächsten Release behoben. Nach jeder Auslieferung bestimmt der Projektleiter zusammen mit dem Team den Stand des Projekts und trägt den tatsächlichen Aufwand bei den realisierten Features nach. Dieser Feedbackmechanismus liefert die Antwort auf die Frage, ob die mit den Aufwandsklassen assoziierten Aufwandswerte plausibel sind. Sollte sich das Niveau des Projekts deutlich von den bisherigen Erfahrungen unterscheiden, entstehen bereits in den ersten Releases deutliche Hinweise, die ein rechtzeitiges Eingreifen ermöglichen.

Dieser Plan ist einfach und flexibel. Änderungen lassen sich leicht erfassen, da die Termine für die Releases feststehen. Features werden anhand der Prioritäten abgearbeitet, die sich immer ändern können, solange das Feature noch nicht umgesetzt ist. Neue Features lassen sich hinzufügen und bestehende Features können gestrichen werden. Ein robuster Änderungsprozess steht somit zur Verfügung, bei dem alle Verantwortlichen die Konsequenzen ihrer Entscheidungen sehen und tragen können.

## Verantwortlichkeit und Erfahrung

Im Rahmen der Personalentwicklung sollten die Stufen der Verantwortlichkeit mit den Erfahrungsstufen abgestimmt werden. Als Anhaltspunkt kann man folgende Erfahrungsstufen formulieren:

- Programmiererfahrung
- Projekt- und Teamerfahrung
- Entwurfserfahrung
- Managementenerfahrung

Analog dazu lässt sich die Verantwortlichkeit in folgende Abstufung bringen:

- Programmierverantwortung: Verantwortlich für ein Feature
- Modulverantwortung: Verantwortlich für ein Feature-Set
- Projektverantwortung: Verantwortlich für eine Anwendung
- Account-Verantwortung: Strategisch verantwortlich für einen Kunden

Diese Stufen sind keine Titel im konventionellen Sinn, es sind dafür auch keine offiziellen Beförderungen notwendig. Zunehmende Verantwortung ist eine Entwicklung, bei der Menschen in Positionen hineinwachsen, die ihrer Erfahrung entsprechen.

## Agile Teams

In einem agilen Team kommt es darauf an, dass sich jeder auf jeden verlassen kann. Es wird weitestgehend auf Protokolle und andere Dokumentation verzichtet, um nicht unnötig Zeit zu verlieren. Allerdings sind ab einer gewissen Teamgröße negative Skaleneffekte durch den Kommunikationsaufwand zu beobachten. Daher sollten agile Teams nicht zu groß werden. Empfehlungen belaufen sich auf bis zu zehn, in Ausnahmefällen auch 20 Mitgliedern. Für den hier vorgestellten Ansatz ist eine Teamgröße von maximal sechs Personen vorgesehen. Dabei handelt es sich immer um Personen mit Programmiererfahrung, davon mindestens die Hälfte des Teams mit Projekt- und Teamerfahrung. Ein Teammitglied muss mindestens über Entwurfserfahrung verfügen. Diese Person übernimmt dann die Projektleitung.

Im Falle einer regional verteilten Entwicklung, wie sie für Nearshore- oder Offshore-Outsourcing typisch ist, müssen Sie darauf achten, dass für das entfernte Team dieselben Kriterien in Bezug auf Erfahrung und Verantwortung gelten. Allerdings kommt eine Modifizierung hinzu: Beide Vertragspartner stellen je ein zusätzliches Teammitglied für die Kommunikation und das Management der Probleme, die durch die räumliche und sprachliche Kluft entstehen.

Der Leiter der Produktentwicklung und der Teamleiter Produkt sind für die reibungslose Abwicklung des Projekts zuständig (Bild 4). Der Leiter der Produktentwicklung legt dabei die Plattform, Infrastruktur und das Vorgehensmodell fest. Er schreibt die Feature-Liste und diskutiert sie mit dem Teamleiter Produkt. Zusammen wird die Aufwandschätzung durchgeführt. Erst danach erfolgt die offizielle Freigabe der Feature-Liste durch den Leiter der Produktentwicklung. Der Teamleiter übernimmt nun die Durchführung der Entwicklung und regelt die Zuteilung der Features an die Entwickler. Er liefert die Releases an den Leiter der Produktentwicklung, der diese testet und das Ergebnis der Tests dem Teamleiter übermittelt.

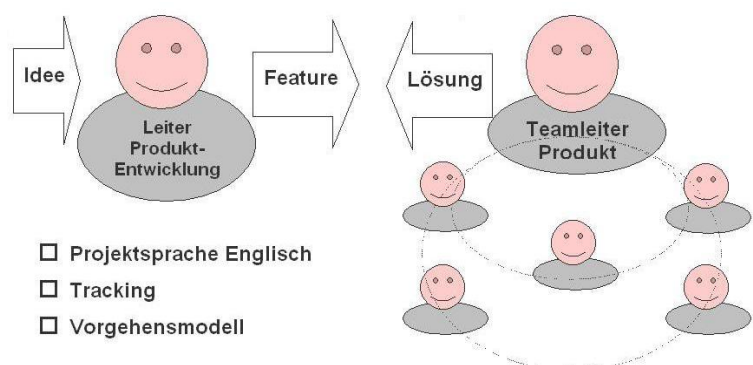


Bild 4: Agiles Team unter Outsourcing-Bedingungen.

## Fazit

Die beschriebene Vorgehensweise wurde von uns aus eigenen Ansätzen und dem "Feature-based Programming" (Richter 2003) entwickelt. Die bisherigen Ergebnisse zeigen, dass agile Teams bei ausreichender Erfahrung und gut balancierter Verantwortlichkeit erfolgreich bei der Durchführung von Software-Projekten unter unsicheren Bedingungen sind. Sie verfügen über Expertenwissen im Bereich der Programmierung und entwickeln schnell ein sehr gutes Verständnis der zu lösenden Probleme.

Wie man solche Teams bildet und einsatzfähig hält, ist jedoch nicht ganz einfach zu beantworten und erfordert weitergehende Untersuchungen. Hinweise zum Thema Teambildung und Teamwork finden sich in (DeMarco, Lister 1999). Die Agilität lebt von der Fähigkeit, Neues schnell erfassen und umsetzen zu können. Dazu müssen alle Teammitglieder mit den geforderten modernen Technologien vertraut sein und sich permanent weiterbilden. Hier müssen vom Unternehmen Bedingungen zum Lernen geschaffen und von den Mitarbeitern Kreativität eingebracht werden. Letzten Endes ist für den dauerhaften wirtschaftlichen Erfolg des Unternehmens die Fähigkeit notwendig, begabte und fähige Mitarbeiter anzuziehen und zu begeistern.

## Literatur

- The Agile Manifesto. URL <http://agilemanifesto.org/>
- Beck, Kent: Extreme Programming Explained, Addison-Wesley, 1999
- Brown, William J. et al.: AntiPatterns, Wiley & Sons, 1998
- von Clausewitz, Carl: Vom Kriege, Erstdruck Berlin 1832/34, Insel Verlag, 2005
- DeMarco, Tom; Lister, Timothy: Wien wartet auf Dich! Hanser Verlag, 1999
- Garmus, David; Herron, David: Function Point Analysis, Addison-Wesley, 2001
- Högg, Roman; Meckel, Miriam; Stanoevska-Slabeva, Katarina; Martignoni, Robert: Overview of business models for Web 2.0 communities. In: Proceedings of GeNeMe 2006, 2006.- GeNeMe 2006.- Dresden, S. 23-37.- URL <http://www.alexandria.unisg.ch/Publikationen/31411> (2007-06-30)
- Richter, Stefan: Feature-based Programming, Addison-Wesley, 2003