Grundlagen der Aufwandsschätzung

Wie wir im Folgenden sehen werden, lassen sich die bisher genannten Herausforderungen bei der Aufwandsabschätzung weitestgehend durch gesunden Menschenverstand und konsequente Anwendung weniger Regeln und einiger Formeln (die sich mit Hilfe von Schulmathematik verstehen lassen) erkennen und aus dem Wege räumen. Auch das generische Rezept zur Durchführung von Aufwandsschätzungen ist alles andere als kompliziert, lautet es doch schlicht: Zählen und Rechnen ist besser als „Raten" (bzw. haltloses Schätzen). Das gilt sowohl für die manuellen Schätztechniken, als auch für die modellbasierten Schätzverfahren, die wir später noch kennenlernen werden. Letztere heißen in der Softwaretechnik üblicherweise **algorithmische Kostenmodelle** bzw. werden manchmal auch **parametrische Modelle** genannt. Kern eines solchen Modells ist eine mathematische Formel, die aus der gemessenen oder geschätzten Größe einer Software und weiteren Einflussfaktoren, wie z. B. der Art des zu entwickelnden Systems oder der Volatilität der Anforderungen, einen ungefähren Entwicklungsaufwand (meist in Personenmonaten) ableitet. Die meisten der heute verwendeten Kostenmodelle sind empirischer Natur, d. h. ihre Schätzfunktionen beruhen auf Regressionsanalysen, die aus den Aufwandsdaten einer Anzahl von untersuchten Projekten gewonnen worden sind. Mit der Familie der COCOMO-Ansätze [Boehm 81 & 00] fällt beispielsweise die bekannteste Gruppe von Kostenmodellen in diese Unterkategorie. Weniger gängige Ansätze, die die zu erwartenden Aufwände aus theoretischen Überlegungen abzuleiten versuchen, bezeichnet man als analytische Modelle.

Letztlich teilen also alle Ansätze zur Aufwandsschätzung zwei grundsätzliche Herausforderungen: zum einen die Überlegung, wie sich die zu erwartende Größe einer Software zuverlässig abschätzen bzw. besser frühestmöglich konkret abmessen lässt, und zum anderen die Frage, wie gut eine Vorhersagefunktion die gemessene Größe und weitere Einflussfaktoren eines Projekts in Aufwand umzurechnen vermag. Diesen Fragestellungen wollen wir im Folgenden zunächst dadurch auf den Grund gehen, dass wir einige in agilen Vorgehensmodellen angewendete Techniken genauer betrachten, die, kombiniert mit einer Portion gesundem Menschenverstand und etwas Disziplin bei der Anwendung, bereits erstaunlich akkurate

Vorhersageergebnisse erzielen können. So ist beispielsweise Scrum mit Hilfe teamspezifischer (Regressions-)Daten in der Lage, bereits nach kurzer Zeit recht genaue Schätzungen für den Gesamtprojektaufwand zu liefern und soll uns aus diesem Grund als ein erstes Anschauungsobjekt dienen, mit dessen Hilfe sich grundlegende Prinzipien der Aufwandsschätzung illustrieren und erlernen lassen.

… am Beispiel von Scrum

Agile Vorgehensmodelle wie **Extreme Programming** und **Scrum** versuchen Software nach dem Vorbild der Lean Production (also der schlanken Produktion) in klassischen Industrien zu entwickeln und konnten damit in den vergangenen Jahren eine immer größere Anhängerschaft für sich gewinnen. Obwohl böse Zungen nach wie vor behaupten, „agil" sei nur ein Euphemismus für ein ungeplantes Adhoc-Vorgehen, enthält ein agiles Management-Framework wie Scrum [Schwaber & Beedle 08] eine Menge brauchbarer Ideen, die, konsequent umgesetzt, auch in nicht agilen Projekten nutzbringend angewendet werden können. Wie alle agilen Vorgehensmodelle verfolgt auch Scrum, auf den sich die folgenden Erklärungen konzentrieren, eine iterative und inkrementelle Vorgehensweise. Das bedeutet, dass Projekte in eine Reihe von besser beherrschbaren Miniprojekten bzw. Iterationen, in Scrum Sprints genannt, zerlegt werden. Ein **Sprint** umfasst eine vorher verabredete Zeitspanne von einer bis maximal vier Wochen, in der eine zuvor festgelegte Menge von Anforderungen verfeinert und in ein Systeminkrement (also ein neues Teil des Gesamtsystems) umgesetzt wird. Scrum verabschiedet sich ganz bewusst von dem Gedanken, Projekte mit allen Eventualitäten mehrere Monate oder gar Jahre im Voraus planen zu können und setzt stattdessen auf eine sogenannte **empirische Prozessüberwachung**, um sozusagen auf Sicht durch ein Projekt zu navigieren. Hinter diesem etwas sperrig anmutenden Begriff verbirgt sich nichts weiter als die simple Idee, die Planung für den weiteren Projektverlauf regelmäßig mit Hilfe vorliegender Projektdaten zu überprüfen und gegebenenfalls anzupassen, sobald sich Abweichungen aus dem bisherigen Projektverlauf extrapolieren lassen.

Die Zeitplanung für das Gesamtprojekt erfolgt in Scrum über das sogenannte Product Backlog, in dem alle umzusetzenden Anforderun-

gen und weitere Aufgaben (wie z. B. das Aufsetzen der Entwicklungsumgebung oder eine notwendige Prototyperstellung) eingetragen werden. Jeder Backlog-Eintrag bekommt nach und nach zwei wichtige Kennzahlen zugewiesen, nämlich zum einen seine Priorität und zum anderen seine „Größe", gemessen in **Story Points** (dazu gleich mehr). Zu Beginn eines Sprints legt sich ein Scrum-Entwicklungsteam auf eine realistische Anzahl von Einträgen (und damit Story Points) aus dem Product Backlog fest, woraufhin diese in das sogenannte **Sprint Backlog** übernommen werden. Wichtig dabei ist, dass ein Scrum-Team nicht vom Kunden oder Management unter Druck gesetzt werden darf, mehr Story Points umzusetzen, als es selbst für machbar hält. Scrum sieht im Übrigen die optimale Teamgröße bei etwa 5 bis 7 Entwicklern, wobei es sich idealerweise um sogenannte generalisierte Spezialisten, also Mitarbeiter mit guten Kenntnissen in allen Bereichen der Softwareentwicklung, handeln sollte.

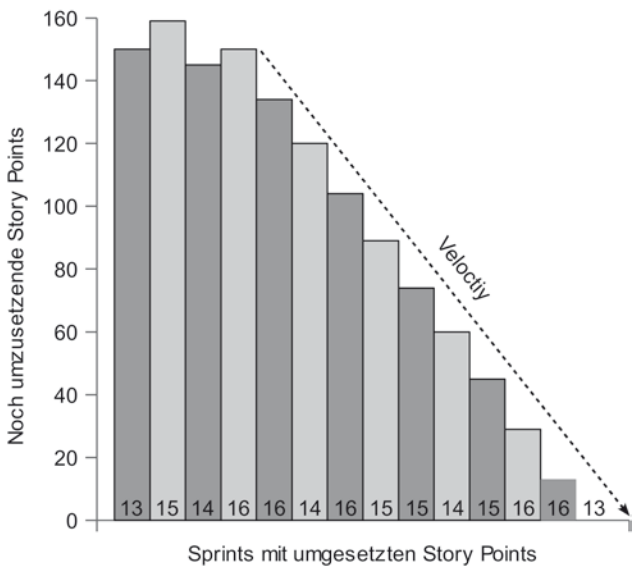
Die Anzahl von in einem Sprint umsetzbaren Backlog-Einträgen abzuschätzen, ist zu Beginn eines Projekts zwar alles andere als einfach (Details zum Vorgehen folgen gleich), da aber die tatsächlich umgesetzten Story Points am Ende jedes Sprints gezählt werden, lässt sich für ein Scrum-Team bereits nach wenigen Iterationen eine durchschnittliche **Entwicklungsgeschwindigkeit** (d. h. die Produktivität des Teams) prognostizieren, mit deren Hilfe auch ein ungefährer Zeitpunkt für die Fertigstellung des Projekts veranschlagt werden kann. Dies erfolgt mit der einfachen, aus dem Physik-Unterricht in der Schule bekannten Formel zur Berechnung der Geschwindigkeit:

$$v = \frac{s}{t} \text{ bzw. genauer: } v = \frac{\Delta s}{\Delta t}$$

Dabei steht v in Scrum für die Entwicklungsgeschwindigkeit (engl. **velocity**), s beschreibt die umgesetzte Anzahl der Story Points und t die dafür verbrauchte Zeit, also z. B. die Anzahl der Iterationen. Grafisch lässt sich das sehr schön in einem sogenannten Release **Burndown Chart** veranschaulichen, in dem die Summe der noch verbleibenden Story Points nach jedem Sprint in einem Balkendiagramm dargestellt wird. Grundsätzlich werden in Scrum übrigens nur Story Points als erledigt gezählt, deren zugehörige Anforderung auch vom Kunden als umgesetzt akzeptiert wurde. Dabei gilt das „Alles-oder-nichts-Prinzip": Ist eine Anforderung vollständig umgesetzt, werden ihre Story

Points zu 100 % für den Sprint gezählt, wird sie nicht abgenommen, werden überhaupt keine Story Points gezählt, und sie muss in einem der folgenden Sprints vollständig neu eingeplant werden.

Ist auf diese Weise die Entwicklungsgeschwindigkeit aus den ersten Sprints ermittelt, kann mit ihrer Hilfe anhand der noch nicht abgearbeiteten Backlog-Einträge das voraussichtliche Projektende vorhergesagt werden. In der folgenden Abbildung wurde beispielsweise mit Hilfe der durchschnittlichen Entwicklungsgeschwindigkeit aus den ersten drei Sprints (13+15+14 ergibt 14 Story Points pro Sprint) das voraussichtliche Projektende in einem Burndown Chart projiziert. Die in der Abbildung dargestellten Balken stehen jeweils für die Menge der am Ende eines Sprints noch zu implementierenden Story Points. Die Zahlen am unteren Rand der Balken geben an, wie viele Story Points im jeweiligen Sprint erfolgreich abgearbeitet werden konnten.



Da agile Projekte zu Beginn die meisten Anforderungen nur oberflächlich erfassen und erst im Projektverlauf weiter detaillieren, ist es, wie auch in der Abbildung bei Sprint 2 und Sprint 4 zu sehen, durchaus

nicht unüblich, dass die Gesamtzahl der zu bearbeitenden Story Points in den ersten Iterationen eines Projekts noch weiter anwachsen und die Prognose des Liefertermins sich verschlechtern kann. Mit Hilfe dieser einfachen Technik zeigt sich in einem Scrum-Projekt also schon nach sehr kurzer Zeit (einige Wochen bis wenige Monate), ob vorliegende Erwartungen bzw. Zielvorgaben tatsächlich realistisch sind und eingehalten werden können. Wird in einer Organisation iterative und agile Softwareentwicklung jedoch nicht wirklich gelebt, ist es mitunter schwierig, mit einer solchen Prognose nach kurzer Zeit bei den richtigen Stellen Gehör zu finden. Dennoch werden die Zahlen in den allermeisten Fällen recht behalten.

Planspiele

Die Entwickler von agilen Vorgehensmodellen haben aus verschiedenen Fehlern der (Wasserfall-)Vergangenheit gelernt und versuchen Anforderungen für ein Projekt auf Basis der gewünschten Funktionalität so weit zu zerlegen, dass sie zuverlässig abschätzbar werden. Die dafür verwendete Maßeinheit **Story Points** ist allerdings kein absolutes Maß (wie z. B. Lines of Code oder Function Points, die wir später noch kennenlernen werden), sondern nur ein relatives, d. h. teamspezifisches Maß, das dazu dient, Anforderungen untereinander vergleichen zu können.

Um alle Teammitglieder gleichberechtigt in die Aufwandsabschätzungen einzubeziehen, verwenden agile Methoden häufig ein sogenanntes Planspiel, in Scrum gewöhnlich **Planning Poker** genannt. Der Spielablauf ist wie folgt: Eine typischerweise als **User Story** [Schwaber & Beedle 08] festgehaltene Anforderung wird dem Team vom Product Owner (einer Mischung aus Produkt- und Projektmanager) vorgestellt. Jedes Teammitglied verfügt über einen Satz von Pokerkarten, die üblicherweise etwa die ersten sechs bis acht Zahlen der Fibonacci-Folge und die Null für die zu schätzende Anzahl an Story Points enthalten. Weitere mögliche Varianten sehen zusätzlich eine Karte mit einer Kaffeetasse („Ich brauche eine Pause“), mit einem Fragezeichen („Anforderung unklar“) und einem „too large“ vor. Wie in der folgenden Tabelle gezeigt, berechnet sich eine neue Zahl der Fibonacci-Folge jeweils durch Addition ihrer beiden Vorgänger, so dass die Folge umso schneller zu wächst, je weiter sie voranschreitet.

Anzahl Story Points	Beschreibung
0	praktisch überhaupt kein Aufwand
1	der kleinste mögliche Aufwand
2	ein sehr kleiner Aufwand, etwa zwei kleinste Aufwände zusammen
3	ein kleiner Aufwand, etwa ein kleinster und ein sehr kleiner zusammen
5	ein mittlerer Aufwand
8	ein großer Aufwand
13	ein sehr großer Aufwand
21	ein riesiger Aufwand
too large	ein viel zu großer, nicht mehr zu überblickender Aufwand

Dieses zunehmende Wachstum ist gewollt, bringt es doch die üblicherweise stark anwachsende Schätzungenauigkeit bei der Abschätzung von (zu) großen Anforderungen zum Ausdruck. Die Grundidee des Planning Poker ist nun, die Schätzungen zu Beginn so zu kalibrieren, dass eine oder mehrere sehr kleine, gut verstandene und überschaubare Anforderungen auf einen oder zwei Story Points festgesetzt werden.

Danach kann das eigentliche Planspiel beginnen, da weitere Anforderungen relativ zu diesem künstlich festgelegten Eichpunkt abgeschätzt werden können. Der Product Owner wählt jeweils die nächste Anforderung aus, beschreibt sie dem Team und diskutiert ggf. offene Fragen mit den Teammitgliedern. Sobald alle Fragen geklärt sind, werden die Teammitglieder gebeten, diejenige ihrer Karten verdeckt vor sich auf den Tisch zu legen, die den Aufwand der vorliegenden Anforderung aus ihrer Sicht am besten bewertet. Haben alle „Spieler" eine Karte abgelegt, werden diese gleichzeitig aufgedeckt, wodurch gewährleistet wird, dass auch die Schätzungen von zurückhaltenderen Teammitgliedern bis dahin unbeeinflusst erfolgen konnten. Zeigen alle Karten die gleiche Anzahl Story Points, gilt die diskutierte User Story mit dem entsprechenden Wert als geschätzt. Sind Abweichungen aufgetreten, werden diese diskutiert und insbesondere die Spieler mit der höchsten und der niedrigsten Schätzung nach einer Begründung für ihre Annahmen gefragt. Nach einer Diskussion der neuen Erkenntnisse wird wie zuvor geschätzt und der Prozess ggf. so lange wiederholt, bis eine einheitliche Schätzung erreicht ist. Dies ist

glücklicherweise meist nach wenigen Wiederholungen der Fall. Sollten Zweifel an der Korrektheit von zuvor geschätzten Anforderungen aufkommen, sind diese ggf. noch einmal zu überprüfen, um sicherzustellen, dass die neuen Schätzergebnisse im Vergleich zu den bisherigen Schätzungen konsistent geblieben sind.

Zeitplanung

Sprints sind die zentralen Elemente, mit deren Hilfe in Scrum Produktinkremente (also lauffähige Teile der Software) erstellt werden. Was das Arbeitstempo betrifft, sollten wir uns aber von diesem Namen nicht in die Irre leiten lassen. Da ein Projekt üblicherweise eher mit einem Marathon- als einem Kurzstreckenlauf gleichzusetzen ist, baut Scrum auf ein nachhaltiges Arbeitstempo, das es allen Mitarbeitern erlaubt, ohne Überlastung dauerhaft an einem Projekt zu arbeiten. Weitere wichtige Regeln sind, dass Sprints auf eine vorab festgelegte Zeitspanne begrenzt bleiben müssen, und dass die jeweiligen Aufwände für Sprintplanung und -abschluss zusammen nicht mehr als 10 % der verfügbaren Zeit beanspruchen dürfen (dies entspricht z. B. bei vierwöchigen Iterationen zweimal einem Tag). Scrum geht davon aus, maximal einen Zeitraum von vier Wochen im Detail planen zu können und sieht für die Sprintplanung die im Folgenden diskutierten Planungsaktivitäten vor: Der Product Owner bereitet üblicherweise im Verlauf des jeweils aktuellen Sprints ein realistisches und prägnantes Ziel für den folgenden Sprint vor und wählt gemäß der festgelegten Prioritäten passende Anforderungen aus dem Product Backlog aus. Zu Beginn eines Projekts kann es dabei sinnvoll sein, hochpriore, aber sehr komplexe Anforderungen zurückzustellen, um dem Team die Möglichkeit zu geben, sich mit einfacheren Aufgaben aufeinander einzuspielen. Ggf. kann es für den Product Owner notwendig werden, sich zur Vorbereitung der eigentlichen Planungssitzung mit einigen Teammitgliedern zu treffen, um die gewählten Anforderungen so aufzubereiten, dass sie später vom gesamten Team zügig verstanden und geplant werden können. Sollte es nicht möglich sein, die Anforderungen entsprechend aufzubereiten, kann das darauf hindeuten, dass z. B. die Erstellung eines Prototypen als Aufgabe für den nächsten Sprint angesetzt werden sollte.

In der Sprint-Planungssitzung, die die genaue Zeitplanung für den bevorstehenden Sprint zum Ziel hat, behält immer das Team das letz-

te Wort über die Anzahl der umzusetzenden Anforderungen. Um in diesem Meeting das **Sprint Backlog** erstellen zu können, muss jedes Teammitglied zuvor seine verfügbare Kapazität, d. h. seine zu erwartende **Bruttoarbeitszeit** ermitteln. Diese umfasst die regelmäßige Anwesenheit abzüglich Fehltage wie Urlaub, Feiertage oder Weiterbildungen. Auch die Aufwände für Sprintplanung und -abschluss (wie gesagt, maximal 10 % der für den Sprint zur Verfügung stehenden Zeit) sind abzuziehen. Scrum geht davon aus, dass die Bruttoarbeitszeit ungleich der Zeit ist, die tatsächlich für das Projekt zur Verfügung steht (also der **Nettoarbeitszeit**), da sonstige Tätigkeiten wie Telefonate, Meetings und ähnliche Aktivitäten üblicherweise etwa ein Viertel bis ein Drittel eines regulären Arbeitstages in Anspruch nehmen. In der Praxis sind durchaus auch noch extremere Werte bekannt. Es empfiehlt sich daher, die Nettoarbeitszeit für ein neues Projekt mindestens um 25 % niedriger anzusetzen und diese Zahl im Projektverlauf zu kalibrieren, also an die tatsächlichen Gegebenheiten anzupassen. Auf diese Weise lässt sich die für einen Sprint tatsächlich zur Verfügung stehende Arbeitszeit des Teams (in Personenstunden), die zur Abarbeitung des Sprint Backlog genutzt werden kann, mit einfachen Mitteln abschätzen.

Anforderungen oder andere Aufgaben liegen zu diesem Zeitpunkt üblicherweise nur in grober Form (also z. B. als User Stories) und auch größenmäßig nur relativ in Story Points abgeschätzt vor, haben also noch keinen direkten Bezug zum benötigten Umsetzungsaufwand. Um böse Überraschungen im Sprint zu vermeiden, ist es in der Planungssitzung Aufgabe des Teams, alle Aktivitäten (Design, Implementierung, Testen etc.) zu finden, die zur Umsetzung eines Product-Backlog-Eintrags erforderlich sind, und diese in das Sprint Backlog zu übernehmen. Alle Sprint-Backlog-Einträge müssen ebenfalls mit einer Aufwandsabschätzung versehen werden und zwar dieses Mal mit einer konkreten Anzahl von benötigten Personenstunden. Die Literatur zu Scrum empfiehlt eine maximale Genauigkeit von einem Viertel-Personentag und je nach Autor pro Aktivität eine obere Grenze von ein bis zwei Personentagen. Aufgaben, für deren Umsetzung mehr Zeit veranschlagt wird, gelten als zu groß und entsprechend mit zu großen Schätzungenauigkeiten behaftet und sollten weiter zerlegt werden, bis sie im genannten Zeitrahmen erledigt werden können. Sind alle Aktivitäten entsprechend abgeschätzt, lässt sich durch Aufsummieren der Zeitaufwände und einen Vergleich mit der summierten Nettoarbeitszeit des Teams sehr leicht erkennen, ob

die geplanten Backlog-Einträge im kommenden Sprint umgesetzt werden können. Ggf. ist zu diesem Zeitpunkt natürlich ein Nachlegen in bzw. Entfernen von Anforderungen aus dem Sprint Backlog möglich. Dabei behält aber stets das Team das letzte Wort über die Machbarkeit. Getreu dem Motto „Puffer statt **Überstunden**" [Mangold 09] weist übrigens auch die Scrum-Literatur immer wieder darauf hin, dass die vorhandene Nettoarbeitszeit des Teams nur zu maximal 85 % verplant werden sollte, da unvorhergesehene Zwischenfälle sonst automatisch zu Überstunden, Stress und letztendlich zu fallender Entwicklungsproduktivität führen werden.

Insgesamt ist Scrum ein relativ überschaubares Vorgehensmodell, enthält aber dennoch weit mehr interessante Managementtechniken, als wir in dieser kompakten Darstellung mit Fokus auf Aufwandsabschätzungen besprechen können. Geeignete weiterführende Literatur, diese genauer zu erkunden, ist beispielsweise das bereits erwähnte Buch des Scrum-Mitentwicklers Ken Schwaber [Schwaber & Beedle 08] oder das allgemeiner auf agile Methoden eingehende Werk von Hruschka et al. [Hruschka 03]. Ähnlichkeiten von in Scrum verwendeten Techniken mit den im Folgenden diskutieren Vorgehensweisen (wie z. B. der Delphi-Methodik oder Expertenschätzungen) und anderen heute verwendeten Vorgehensmodellen sind übrigens sicherlich nicht zufällig: Wie so oft in der Software-Industrie standen auch hier verschiedene ältere Ansätze Pate und lieferten hilfreiche Best Practices, die gemeinsam den Kern der Lean-Development-Philosophie von Scrum ausmachen.

Einfache Schätztechniken

Nachdem uns Scrum einen ersten Einblick in Schätzmodelle zur Softwareentwicklung geliefert hat, betrachten wir in diesem Unterkapitel zunächst einige grundlegende Techniken, die sich sehr gut einsetzen lassen, um mit wenig Aufwand, frühzeitig ein erstes Gefühl für den Aufwand eines Projekts zu bekommen. Mit einem gesteigerten Schätzaufwand ermöglichen sie später im Projektverlauf sogar eine recht präzise Abschätzung des zu erwartenden Entwicklungsaufwands. Im Wesentlichen ist dazu die Übersicht über die im Projekt erforderlichen Tätigkeiten und etwas Erfahrung zur Abschätzung des zur Umsetzung erforderlichen Aufwands völlig ausreichend. Bevor wir uns aber konkret

mit den Schätzverfahren auseinandersetzen können, sollten wir uns zunächst noch einmal ins Gedächtnis rufen, welche Tätigkeiten bei der Planung und Durchführung großer Softwareprojekte eine Rolle spielen.

Teile und herrsche

Heutige Softwaresysteme sind meist viel zu komplex, um von einer einzelnen Person als monolithische Einheit entwickelt werden zu können. Die Koordination von zahlreichen Projektbeteiligten findet sich also als eine zentrale Herausforderung im Projektmanagement wieder, von wo wir uns aus diesem Grund zunächst die sogenannte Netzplantechnik [Mangold 09] in Erinnerung bringen: Prinzipiell geht es darum, ein Projekt in voneinander abhängige Teilaktivitäten zu zerlegen, um auf deren Basis eine Zeitplanung erstellen zu können. Das dahinter stehende „Divide-and-Conquer-Prinzip" ist nicht nur für die Planung nützlich, sondern erleichtert, wie bei Scrum bereits gesehen, auch die Aufwandsschätzungen, da es offensichtlich einfacher und genauer ist, eine überschaubare Teilaktivität und nicht gleich ein komplettes Projekt schätzen zu müssen.

Diese Vorgehensweise ist zwar prinzipbedingt mit mehr Aufwand verbunden, kann sich aber z. B. auch in einem Verkaufsgespräch mit einem Kunden durchaus lohnen. So wirkt (und ist) eine Aufschlüsselung eines Projekts in vielleicht 30 Unteraktivitäten bzw. Komponenten, die sich schlüssig zur gesamten Projektdauer addieren, mit Sicherheit professioneller als ein einzelner, grob über den Daumen gepeilter und schlecht begründbarer Wert. Auch mathematisch betrachtet ist diese Zerlegung vor dem Hintergrund des Gesetzes der großen Zahlen vorteilhaft, zumindest wenn wir davon ausgehen, dass sich das Über- und Unterschätzen bei den einzelnen Teilaktivitäten in etwa die Waage halten. In einem solchen Fall mitteln sich Schätzfehler heraus, und es ist wahrscheinlich, dass eine Addition von Einzelschätzungen letztlich näher am tatsächlichen Ausgang des Projekts liegen wird, als eine einzelne Schätzung für das gesamte Projekt.

Aber zurück zur Netzplantechnik: Unter der Voraussetzung, dass Abhängigkeiten zwischen Teilaktivitäten bekannt sind (also z. B. müssen Anforderungen an eine Komponente erfasst sein, bevor ein Design für sie erstellt werden kann), lassen sich diese hinter- bzw. ggf. auch

nebeneinander anordnen, so dass die minimale Projektdauer, also der sogenannte kritische Pfad, bestimmt werden kann. **Puffer** für andere Pfade durch das Projekt ergeben sich automatisch, da diese im Normalfall kürzer sind. Das Wasserfallmodell [Bunse & v. Knethen 08] stellt zwar aus Sicht des Projektmanagements eine einfache und effektive Unterteilung eines Entwicklungsprojekts in einzelne Entwicklungsphasen (Anforderungsanalyse, Softwareentwurf, Implementierung, Testen) dar, als Grundlage für Aufwandsschätzungen und einen Netzplan ist es allerdings zu grob, so dass sich in der Literatur wie beispielsweise bei Jones [Jones 07] eine feinere Aufschlüsselung in 25 notwendige Entwicklungsaktivitäten findet. Verfeinert man diese nochmals in Unteraktivitäten, gelangt man zu einer Menge von etwa 50 bis 100 Aufgaben, die im Englischen die **Work Breakdown Structure** (oder kurz WBS) eines Softwareprojekts genannt wird. Neben einer am Wasserfallmodell angelehnten generischen WBS findet sich beispielsweise in Barry Boehms Buch [Boehm 00] auch eine an den moderneren Rational **Unified Process** (RUP, s. [Larman 05]) angepasste Version. Jones argumentiert, dass zur Erstellung von Schätzungen üblicherweise eine Verfeinerung auf die Ebene der 25 genannten Entwicklungsaktivitäten (pro Komponente) ausreichend sei. Das Ableiten einer Aufwandsschätzung aus der Summe der ungefähren Aufwände von Teilaktivitäten ist somit durch zwei Eigenschaften charakterisierbar: zum einen erfolgt es von unten nach oben (also bottom-up) und zum zweiten handelt es sich um ein lineares Verfahren, das mögliche negative Skaleneffekte in sehr großen Projekten (mehr dazu auf Seite 33) nicht berücksichtigt.

Wie wir noch sehen werden, ist auch der umgekehrte Weg (also ein Top-down-Vorgehen) möglich. Mit Hilfe algorithmischer Schätzmodelle (s. ab Seite 68) wird anhand verschiedener Projektparameter der Gesamtaufwand für ein Projekt abgeschätzt und anschließend auf die einzelnen Entwicklungsaktivitäten heruntergebrochen. Die dafür notwendige ungefähre prozentuale Zuordnung der Aufwände auf die jeweiligen Phasen findet sich ebenfalls bei Boehm; wir werden diese auf Seite 88 noch genauer betrachten. In der Praxis werden übrigens meist beide Verfahren miteinander kombiniert: Die initiale Aufwandsabschätzung wird mit Hilfe eines algorithmischen Modells erstellt und anteilig auf einzelne Anforderungen heruntergebrochen. Auf Basis der Entwicklungsaktivitäten erfolgt anschließend eine Verteilung der zur Verfügung stehenden Ressourcen auf diese und letztlich die Erstellung eines detaillierten Projekt- bzw. Iterationsplans.

Analogieschlüsse

Sicherlich eines der am einfachsten durchführbaren Schätzverfahren sind Analogieschlüsse, vorausgesetzt, es existieren brauchbare Vergleichsdaten von mindestens einem ähnlichen Vorgängerprojekt. Das prinzipielle Vorgehen ist bereits im Mathematikunterricht der Schule gebräuchlich, nutzt es doch einen simplen **Dreisatz**, um aus Größe und bekanntem Entwicklungsaufwand eines Altsystems und der geschätzten Größe eines Neusystems den Entwicklungsaufwand für das neue System abzuleiten. Hilfreich ist es dabei, das Altsystem in wenigstens fünf Teile oder Aktivitäten zerlegen zu können, wozu sich natürlich die Verwendung einer Work Breakdown Structure als Grundlage anbietet. Generell gilt, je mehr Einzelschritte bzw. -teile verglichen werden können, desto genauer wird das Ergebnis ausfallen, allerdings um den Preis eines entsprechend höheren Aufwands.

Idealerweise sind für das Altsystem entweder die Lines of Code (LOC, weitere Details ab Seite 35) oder direkt der investierte Aufwand in Personenmonaten bekannt, und ein anderes Größenmaß (wie z. B. die Anzahl der Datenbanktabellen) ist einfach abzählbar, wie beispielhaft in der folgenden Tabelle für eine recht überschaubare Webapplikation gezeigt:

Systemteil	Anzahl	Größe
Benutzerschnittstelle	15 Webseiten	2.500 LOC
Reports	10 Reports	3.000 LOC
Geschäftsprozesse	8 Prozesse	1.600 LOC
Entitäten	24 DB-Tabellen	2.400 LOC
Utilities	12 Klassen	2.000 LOC
Summe	-	11.500 LOC

Wichtig bei diesem Vorgehen ist natürlich, die tatsächlichen Werte des Altsystems zum Vergleich heranzuziehen und nicht an Stelle des tatsächlichen Aufwands nur die ursprüngliche Schätzung oder Planung als Vergleichsgrundlage zu verwenden. Die folgende Tabelle zeigt zunächst die so ermittelten Multiplikationsfaktoren für das Neusystem.

Systemteil	Anzahl alt	Schätzung neu	Faktor
Benutzerschnittstelle	15 Webseiten	18 Webseiten	1,2
Reports	10 Reports	15 Reports	1,5
Geschäftsprozesse	8 Prozesse	6 Prozesse	0,75
Entitäten	24 DB-Tabellen	20 DB-Tabellen	0,83
Utilities	12 Klassen	15 Klassen	1,25

Mit deren Hilfe kann nun die ungefähre Größe des Neusystems auf Basis seiner Einzelteile abgeschätzt werden, wie in der folgenden Tabelle dargestellt.

Systemteil	Größe alt	Faktor	Größe neu
Benutzerschnittstelle	2.500 LOC	1,2	3.000 LOC
Reports	3.000 LOC	1,5	4.500 LOC
Geschäftsprozesse	1.600 LOC	0,75	1.200 LOC
Entitäten	2.400 LOC	0,83	2.000 LOC
Utilities	2.000 LOC	1,25	2.500 LOC
Summe	11.500 LOC	-	13.200 LOC

Liegt, wie in der Tabelle gezeigt, nur die Größe des Altsystems vor, kann auch nur die ungefähre Gesamtgröße des Neusystems geschätzt werden, die dann z. B. als Eingangsparameter für ein algorithmisches Schätzmodell (ab Seite 68) verwendet werden sollte. Liegt zumindest pauschal der für die Erstellung des Altsystems benötigte Aufwand vor, kann auch dieser für das Neusystem hochgerechnet werden. Nehmen wir an, die Entwicklung des Altsystems habe insgesamt etwa 25 Personenmonate Aufwand benötigt, so müssen wir diese mit dem Größenverhältnis der beiden Systeme (13.200 geteilt durch 11.500) multiplizieren und erhalten einen Aufwand von ungefähr 29 Personenmonaten. Noch besser wäre es natürlich, wenn die Aufwände sogar heruntergebrochen auf die einzelnen Systemteile vorhanden wären, denn dann erhalten wir ohne den Umweg über die LOC die Prognose für die Aufwände der einzelnen Systemteile.

Wie zuvor bereits angedeutet, ist eine möglichst große Ähnlichkeit zwischen beiden Systemen wichtig, das schließt sowohl die Zielplattform als auch andere Einflüsse wie das Projektteam oder schlicht das Systemumfeld (ein nur „inhouse“ verwendetes Intranetsystem ist beispielsweise mit dem zentralen Online-Shop eines Versandhändlers

nur schwer vergleichbar) mit ein. So macht es beispielsweise wenig Sinn, ein in Cobol erstelltes Altsystem mit einem Neusystem in Java vergleichen zu wollen. Mit Vorsicht zu genießen sind ferner Prognosen, bei denen Einzelfaktoren sehr groß (etwa über 2,5) oder sehr klein (etwa unter 0,5) ausfallen, da in einem solchen Fall Skaleneffekte das Ergebnis verfälschen können.

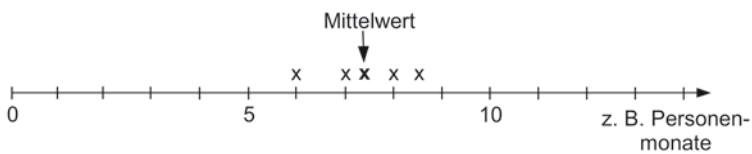
Expertenschätzungen

Schätzungen durch Experten sind ebenfalls eine einfache und daher in der Praxis weit verbreitete Methode zur Ableitung von Aufwandsschätzungen auf Anforderungsebene. Vorausgesetzt, es sind erfahrene Experten verfügbar, erreicht dieses Vorgehen in der Praxis durchaus brauchbare Ergebnisse. Dies gilt insbesondere dann, wenn nicht nur ein Experte zu seiner Meinung befragt wird, sondern etwa drei bis fünf Spezialisten mit möglichst unterschiedlichen Hintergründen ihre Schätzungen abgeben und diskutieren können. Studien belegen für Einzelschätzungen eine Fehlerrate von etwa 50 bis 60 %, während von Expertengruppen abgegebene Schätzungen sich etwa bei der Hälfte (also 25 bis 30 % Fehlerrate) einpendeln. Idealerweise werden Expertenschätzungen übrigens von den Mitarbeitern erstellt, die später auch die Umsetzung der jeweiligen Aufgabe übernehmen sollen. Sie können sowohl die erwartete Größe der Software (beispielsweise in Lines of Code) als Eingabe für ein algorithmisches Kostenmodell, als auch direkt den zu erwartenden Entwicklungsaufwand abschätzen. Da Ersteres sicher nicht einfacher oder genauer zu schätzen ist, ist es sinnvoller, direkt auf die anschaulicheren Aufwände zu fokussieren.

Ferner hat es sich in der Praxis als hilfreich erwiesen, auch Schätzklausuren von Expertengruppen zu strukturieren und zu moderieren, in der Literatur finden sich dafür zwei entsprechende Techniken, die beide nicht zufällig an das Planning Poker von Scrum erinnern. In Anspielung auf das Orakel von Delphi wurde in den späten 1940er Jahren in den USA die sogenannte **Delphi-Methode** zur Erstellung von Aufwandsschätzungen (damals natürlich außerhalb der Softwareindustrie) entwickelt. Diese sieht vor, dass verschiedene Experten unabhängig voneinander ihre Schätzungen festlegen, diese bei einem Treffen diskutieren und sich abschließend gemeinsam auf einen Wert

einigen. Dieser sollte, wie beim Planning Poker von Scrum bereits beschrieben, nicht einfach der Mittelwert aller Prognosen sein, sondern ein begründeter Wert, der für alle Teilnehmer plausibel und nachvollziehbar erscheint.

Verschiedene Studien legen allerdings nahe, dass diese einfache Methodik keine deutliche Verbesserung gegenüber unstrukturierteren Schätzworkshops erbringt, so dass Barry Boehm daraus die sogenannte **Breitband-Delphi-Methode** (Wideband Delphi) entwickelt hat [Boehm 81]. In dieser finden sich noch weitere bereits aus dem Planning Poker von Scrum bekannte Elemente (die natürlich Scrum von dort übernommen hat). So wird innerhalb einer Schätzklausur nach Breitband-Delphi zunächst das Projekt von einem Moderator vorgestellt, und die einzelnen Schätzaufgaben werden den Experten mitgeteilt. Anschließend werden die Aufgaben gemeinsam abgearbeitet, wobei jeder Experte seine Schätzungen zunächst unabhängig erstellt und sein vorläufiges Ergebnis anonym dem Moderator zukommen lässt (optional können diese Schritte auch bereits vor dem eigentlichen Meeting durchgeführt werden). Der Moderator stellt die Schätzergebnisse zusammen und trägt sie in der folgenden Grafik ein, die beispielsweise auf einem Whiteboard, einem Flipchart oder auch auf einem Beamer präsentiert werden kann.



Wichtig ist dabei, die Bandbreite der Skala groß genug zu wählen, um den Teilnehmern nicht von vorneherein eine Beschränkung zu suggerieren. Der Moderator trägt in diese Grafik auch den Mittelwert der Schätzungen ein, woraufhin die Teilnehmer die vorliegenden Schätzungen diskutieren, um herauszufinden, welchen Wert alle als eine realistische Prognose betrachten. Liegen alle Schätzungen sehr nahe beieinander, kann es hilfreich sein, einem der Teilnehmer die Rolle des Advocatus Diaboli zuzuweisen, der bewusst versucht, mögliche Gründe für größere Abweichungen vorzubringen. Sobald die Teilnehmer eine Schätzung als ausreichend diskutiert erachten, wird anonym darüber abgestimmt, ob der so ermittelte Erwartungswert für alle

akzeptabel ist. Sollte ein Teilnehmer diesem Wert nicht zustimmen, muss die aktuelle Schätzrunde wieder von vorne begonnen werden. Werden solche Iterationen notwendig, hat es sich als sinnvoll erwiesen, auch die Ergebnisse der vorherigen Runden im Blick zu behalten, so dass die Teilnehmer erkennen können, wie die Schätzungen über die Zeit (hoffentlich) konvergieren. Steve McConnell beschreibt in seinem Buch [McConnell 06] eigene Studien anhand derer er die Ergebnisse von Wideband Delphi um ca. 50 % besser einschätzt, als die von unstrukturierteren Vorgehensweisen, natürlich um den Preis eines wesentlich höheren Zeitaufwands. Interessanterweise gibt es auch Studien, die solche Vorteile nicht nachweisen konnten, so dass es plausibel erscheint, Wideband Delphi vor allem dann einzusetzen, wenn bisher unbekannte Aufgaben abgeschätzt werden sollen. Soll also beispielsweise eine zuvor noch niemals eingesetzte Technologie verwendet werden, können die in der Schätzklausur vorgebrachten Argumente der Experten auch direkt in die Risikoanalyse [Ebert 06] des Projekts einfließen und somit diesen gesonderten Arbeitsaufwand reduzieren helfen.

Zwei- und Drei-Punkt-Schätzungen

Unter dem in der Praxis herrschenden Zeitdruck ist es nicht immer möglich, Schätzaufgaben so weit in Teilaktivitäten zu zerlegen, dass sie mit einem guten Bauchgefühl und entsprechender Genauigkeit abschätzbar wären. Schätzexperten tun sich in einem solchen Falle häufig entsprechend schwer, sich auf einen einzelnen Schätzwert festzulegen. Dieses Problem lässt sich aber mit einem kleinen Kniff, der sogenannten **Zwei-Punkt-Schätzung**, sogar dafür benutzen, die Genauigkeit von Schätzungen besser beurteilen zu können: Dafür erstellen wir für jede Teilaktivität jeweils eine optimistische (**Best Case** b_i) und eine pessimistische (**Worst Case** w_i) Schätzung aus denen sich mit Hilfe der Formel

$$e_i = \frac{(b_i + w_i)}{2}$$

ein mittlerer Erwartungswert e_i herleiten lässt, der für die weitere Projektplanung verwendet werden kann. Die Nutzung dieses Verfahrens

ist besonders für frühe Projektphasen zu empfehlen, wenn noch sehr wenige Einzelheiten über das zu erstellende System bekannt sind.

Im Rahmen der sogenannten Program Evaluation and Review Technique (PERT) wurde in den späten 1950er Jahren von Lockheed und der US Navy die sogenannte **Drei-Punkt-Schätzung** (die auf der Dreiecksverteilung basiert) entwickelt. Diese sieht zusätzlich zum Best und Worst Case noch die Ermittlung eines wahrscheinlichsten (most likely) Ergebnisses m_i vor, das mit Hilfe der folgenden Formel gewichtet in die Ermittlung des Erwartungswertes einfließen kann:

$$e_i = \frac{b_i + 4 \cdot m_i + w_i}{6}$$

Die Standardabweichung s als ein Maß für die Genauigkeit des Erwartungswertes lässt sich für eine entsprechende Dreiecksverteilung als ein Sechstel der Differenz zwischen Worst und Best Case abschätzen:

$$s_i = \frac{w_i - b_i}{6}$$

Angenommen, wir schätzten die optimale Dauer einer Aktivität auf 9 Monate, den wahrscheinlichsten Verlauf auf 11 und die Dauer bei schlechtem Verlauf auf 15 Monate, so läge unser Erwartungswert bei rund 11,3 Monaten und die zugehörige Standardabweichung bei einem Monat. Eine entsprechende Reihe von einzelnen Erwartungswerten (für die Dauern verschiedener Aktivitäten) kann, wie in der folgenden Formel gezeigt, zur gesamten Projektdauer aufsummiert werden:

$$e = \sum_{i=1}^{n} e_i$$

Aus der Statistik wissen wir, dass zur Berechnung der Standardabweichung von aufsummierten Erwartungswerten die Wurzel über die Summe der Quadrate der einzelnen Standardabweichungen gebildet werden muss:

$$s = \sqrt{\sum_{i=1}^{n} s_i^2}$$

Ferner wissen wir aus der Statistik, dass sich die Summe mehrerer Erwartungswerte (bzw. ihrer Verteilungen) recht schnell einer Normalverteilung annähert, so dass wir mit dem Erwartungswert *e* und der Standardabweichung *s* die Eintrittswahrscheinlichkeiten konkreter Projektdauern abschätzen können. Weitere Hintergründe und ein Beispiel dazu folgen ab Seite 99.

Fehlerrechnung

Wenn wir langfristig unsere Fähigkeiten in der Aufwandsschätzung verbessern möchten, drängt es sich nach Abschluss eines Projekts natürlich auf, die gemachten Schätzungen mit den tatsächlichen Ergebnissen zu vergleichen. Voraussetzung dafür ist zum einen, die Schätzungen aufzubewahren und zum anderen die tatsächlichen Aufwände der durchgeführten Projekte zu erfassen. Da beides für ein effektives Controlling ohnehin notwendig ist, kommt nur noch die Fehlerrechnung selbst als minimaler Zusatzaufwand hinzu. Zu diesem Zweck bietet es sich an, den Betrag der relativen Fehlergröße (engl. magnitude of relative error oder MRE) mit Hilfe der folgenden Formel zu bestimmen:

$$MRE = \left| \frac{Aufwand - Schätzung}{Aufwand} \right|$$

Bei einer entsprechend genauen Aufwandserfassung im Projekt lässt sich der Gesamt-MRE als ein Maß für die Schätzqualität durch einfaches Aufsummieren der MREs aller Teilaktivitäten ermitteln. Alternativ kann natürlich auch der Gesamtaufwand der entsprechenden Schätzung gegenübergestellt werden. Aber Achtung, eine brauchbare Fehlerrechnung setzt natürlich voraus, dass nicht versucht wird, Äpfel mit Birnen zu vergleichen. Häufig schleichen sich nämlich unbemerkt Fehlerquellen ein, die die Aussagekraft einer Fehlerrechnung sehr schnell zunichte machen können: Entspricht der Funktionsumfang am Ende des Projekts noch dem ursprünglich geplanten oder ist durch Change Requests oder Scope Creep (s. Seite 118) neue Funktionalität hinzugekommen? Möglicherweise ist auch auf Grund von Zeitdruck im Projekt Funktionalität gestrichen worden? Wurde das bei der Bemessung des tatsächlichen Aufwands ebenso berücksichtigt wie

die Frage, ob alle geplanten Aktivitäten (insbesondere das Testen) mit der ursprünglich geplanten Sorgfalt durchgeführt werden konnten? Ferner ist es natürlich wichtig, die eigentliche Aufwandsschätzung mit dem tatsächlich geleisteten Aufwand zu vergleichen und nicht etwa einen dem Kunden angebotenen Aufwand (mit einkalkuliertem Puffer) mit dem letztendlich in Rechnung gestellten. Zur Erinnerung, sind die genannten Punkte beachtet worden, gilt in der Praxis bereits ein Schätzfehler von 10 % als ein sehr guter, nur schwer erreichbarer Wert. Trotzdem sollten sich mit zunehmender Erfahrung in den bisher besprochenen und noch folgenden Techniken die MREs künftiger Projekte diesem Wert zumindest annähern.

Kampfpreise, Parkinson und ein Ding der Unmöglichkeit

Da Softwareprojekte üblicherweise nicht unbeeinflusst von äußeren Umständen stattfinden, finden sich in der einschlägigen Literatur (wie z. B. [Boehm 81]) häufig zwei weitere Schätzstrategien, die eigentlich keine sind. Der Vollständigkeit (und der Unterhaltung) halber sollen sie an dieser Stelle aber dennoch genannt werden. Eine davon findet oft Anwendung, wenn eine scharfe Wettbewerbssituation ein Unternehmen dazu zwingt, ein Angebot mit einem Kampfpreis abzugeben, der unter Umständen nicht einmal mehr kostendeckend ist. In der englischsprachigen Fachliteratur wird das üblicherweise als **Pricing to Win** bezeichnet. Dieses Vorgehen mag aus wirtschaftlichen oder strategischen Erwägungen heraus sinnvoll sein, um beispielsweise in einem wichtigen Marktsegment Fuß zu fassen oder in schweren Zeiten überhaupt einen Auftrag zu bekommen, den potentiellen Projektbeteiligten bleibt aber letztlich nur die Hoffnung, dass ein Mitbewerber diesen Kampfpreis noch einmal unterbieten und sich das zu erwartende **Todesmarsch-Projekt** [Yourdon 04] einhandeln wird. Denn üblicherweise hat das Management spätestens bei Vertragsabschluss das strategische Ziel längst wieder aus den Augen verloren und verlangt danach doch einen möglichst kostendeckenden oder gar gewinnbringenden Projektverlauf. Und das bedeutet für Softwareprojekte zumeist, möglichst viel Arbeit von möglichst wenigen Mitarbeitern (mit möglichst vielen unbezahlten Überstunden) erledigen zu

lassen. Von einer sinnvollen Schätzstrategie kann in diesem Zusammenhang natürlich keine Rede mehr sein. In einem solchen Fall sollte man keinesfalls der Versuchung erliegen, aus den festgelegten Kosten in einer Art „umgekehrten Schätzung" den erlaubten Aufwand für das Projekt zu errechnen. Soll dennoch ein kostendeckender Projektverlauf erreicht werden, ist der einzig gangbare Ausweg das rigorose Beschneiden der Funktionalität des zu erstellenden Systems.

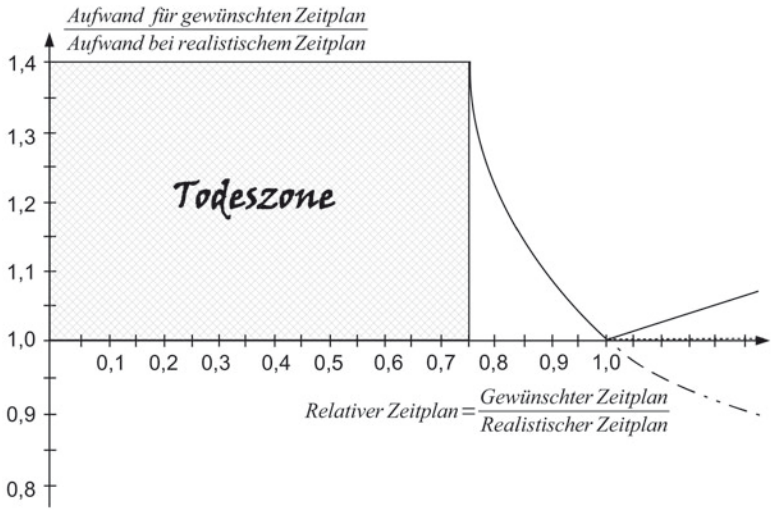
Gerüchteweise ist auch das genaue Gegenteil der gerade geschilderten Situation bereits in der Praxis beobachtet worden, nämlich der Fall, dass für ein Projekt mehr als ausreichend Zeit und Ressourcen zur Verfügung standen. Interessanterweise lässt sich dann vielfach beobachten, dass ein Projekt nicht nur den gegebenen Zeitrahmen vollständig ausfüllt, sondern auch alle angebotenen Ressourcen komplett verschlingt und nicht etwa, wie vielleicht zu erwarten wäre, früher und mit weniger Gesamtaufwand abgeschlossen werden kann. Dieser Effekt wird in der Literatur gerne als **Parkinson's Law** bezeichnet. Gründe für diese Problematik sind vor allem das sogenannte **Gold Plating**, also das Verzieren des Systems mit immer neuen Verschönerungen, die auf Grund des nicht vorhandenen Zeitdrucks noch eben eingebaut werden, sowie das uns sicher allen bekannte Aufschieben von Aufgaben mit in weiter Ferne liegenden Deadlines, das gemeinhin auch als **Studentensyndrom** bekannt ist.

Nichts ist unmöglich?

Vergegenwärtigen wir uns die beiden eben angesprochenen Phänomene noch einmal in grafischer Form: Gehen wir dazu davon aus, dass wir in der Lage wären, die tatsächlich benötigte Dauer eines Projekts korrekt abzuschätzen. Wir sehen auf der x-Achse der folgenden Abbildung den relativen Zeitplan für ein solches hypothetisches Projekt, wobei der Wert 1,0 genau die tatsächlich benötigte Zeit beschreibt. Die y-Achse beschreibt den Aufwand, der für einen gewünschten Zeitplan notwendig ist, jeweils im Verhältnis zu dem Aufwand, der bei idealem Zeitplan notwendig wäre. Anders ausgedrückt beschreibt der Schnittpunkt von 1,0 auf der x- und 1,0 auf der y-Achse genau den idealen Aufwand bei korrekt geschätzter Projektdauer.

Wird ein tatsächlicher Projektaufwand überschätzt, führt das nach Parkinson's Law für jeden Tag über der idealen Projektdauer zu einem

linearen Anstieg des Aufwands, in der Abbildung als ansteigende Gerade am rechten Rand zu sehen. Es gibt allerdings auch andere Schätzmodelle, die in einem solchen Fall einen gleichbleibenden Aufwand (gepunktete Linie auf der x-Achse) oder sogar einen abnehmenden Aufwand (gestrichelte Linie unterhalb der x-Achse) prognostizieren. Durch gutes Projektmanagement sollte sich nach herrschender Meinung zumindest ein gleichbleibender Aufwand erreichen lassen.



Bewegen wir uns nun aber ausgehend von der idealen Dauer auf der x-Achse nach links, wollen das Projekt also in kürzerer Zeit beenden, bedeutet das einen steigenden Aufwand, da die gleiche Arbeit mit mehr Mitarbeitern in weniger Zeit verrichtet werden muss. Bedingt durch negative Skaleneffekte steigt dieser Aufwand in etwa kubisch an, worin praktisch alle bekannten Schätzmodelle übereinstimmen (vgl. z. B. die entsprechend umzustellende Formel von COCOMO auf Seite 71). Diese Erkenntnis ist nicht neu: Es wird in der Literatur entsprechend immer wieder eindringlich darauf hingewiesen, dass eine Verkürzung der Projektdauer auf weniger als etwa 75 % der nominal benötigten Zeit unmöglich ist, da alle analysierten Aufwandskurven in einer solchen Situation beinahe senkrecht in die Höhe schießen.

Die dafür zusätzlich benötigten Mitarbeiter treiben den Kommunikationsaufwand so sehr in die Höhe, dass das Projekt unter diesen Umständen von vorne herein zum Scheitern verurteilt ist (wie bei dem auf Seite 10 beispielhaft genannten Projekt mit 600 Mitabeitern und einem Zeitrahmen von einer Woche). Von dem für qualitativ gute Softwareentwicklung so wichtigen Wissensaufbau im Projekt, der unter extremem Zeitdruck natürlich ebenfalls stark leidet, gar nicht zu sprechen.

Lawrence Putnam [Putnam & Myers 92] hat für den genannten Bereich linksseits der 75 % den schönen Namen „The Impossible Zone" (in der Grafik mit ***Todeszone*** übersetzt) gefunden, für die Steve McConnell [McConnell 06] noch einmal sehr eindringlich unterstreicht, dass eine Zeitplankompression von mehr als 25 % nicht möglich ist und weder durch härtere noch durch bessere Arbeit und schon gar nicht durch das Hinzufügen von noch mehr Mitarbeitern erreicht werden kann: Es ist schlichtweg ausgeschlossen! Jeder Versuch eines Eindringens in die Todeszone, wird für die Projektbeteiligten unweigerlich zu einem Todesmarsch [Yourdon 04] und durch nicht einhaltbare Zeitpläne, Kostensteigerungen, extremen Stress sowie letztlich durch ein gescheitertes Projekt bestraft. Zusammenfassend sei noch einmal Fred Brooks [Brooks 95] zitiert: „Wegen eines Mangels an Kalenderzeit sind (bisher) mehr Softwareprojekte gescheitert als aus allen anderen Gründen zusammen."

http://www.springer.com/978-3-8274-2751-9

Aufwandsschätzungen in der Software- und Systementwicklung kompakt

Hummel, O.

2011, VIII, 128 S. 16 Abb., Softcover
ISBN: 978-3-8274-2751-9