

Software Aufwandschätzung

Inhaltsverzeichnis

1 Einleitung.....	3
1.1 Software.....	3
1.2 Probleme bei der Softwareentwicklung.....	3
2 Software Aufwandschätzung.....	4
3 Methoden der Software Aufwandschätzung.....	5
3.1 Empirische Schätzverfahren.....	5
3.1.1. Expertenschätzung.....	5
3.1.2. Delphi Methode.....	5
3.2 Algorithmische Schätzverfahren.....	6
3.2.1. Function Point Verfahren.....	6
3.2.2. COCOMO.....	9
3.2.3. COCOMO II.....	11
3.3 Andere Schätzverfahren.....	13
4 Résumé.....	14
Quellenangaben.....	15

Ausarbeitung im Rahmen des Proseminars
Software Engineering SS2004

Seminarleiter : Patrick Reuther
Andreas Weber

Autor : Marco Schuh
schuh@castor.uni-trier.de

1 Einleitung

Dieser Text soll einen Einblick in die Software Aufwandschätzung geben, einige wichtige Schätzverfahren vorstellen und die Fragen klären : Was ist Software Aufwandschätzung ? Wie wird sie durchgeführt ? Warum wird sie durchgeführt ?

Bevor ich auf das eigentliche Thema der Software Aufwandschätzung eingehe, möchte ich kurz einen Begriff einführen, vor allem für diejenigen, die diesen Text ohne das Thema betreffende Vorwissen lesen.

Wenn man über Software Aufwandschätzung redet, stellt sich einem erst einmal die Frage : Was ist überhaupt Software ?

1.1 Software

Unter Software versteht man die Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Computersystems zu tun haben.

(frei übersetzt nach IEEE 610.12)

Aus dieser Definition wird leicht ersichtlich, dass es sich bei Software um weit mehr handelt, als das Programm selbst.

Folglich besteht auch die Entwicklung von Software aus weit mehr als dem reinen Produzieren und Kompilieren von Code. Es muss dokumentiert werden, Anpassungen für den Kunden vorgenommen werden oder Fehler verbessert werden.

1.2 Probleme bei der Softwareentwicklung

Probleme gibt es viele bei der Softwareentwicklung, weil diese erstens ein weit gefächertes Feld beschreibt und zweitens sehr komplex sein kann.

Natürlich ist und bleibt der Mensch eine der größten Fehlerquellen bei der Softwareentwicklung, egal ob durch Tippfehler, Fehleinschätzungen oder falsche Programmkonstruktion.

Das Tempo der Evolution von Software sowohl während der Entwicklung als auch während der Benutzung ist ein weiteres nicht zu unterschätzendes Problem, denn größere Softwareprojekte dauern meist mehrere Jahre an und in dieser Zeit verändern sich oft schon die Anforderungen an die Software selbst, sowie technische Gegebenheiten und Möglichkeiten.

Obwohl es noch viele weitere Probleme gibt, nenne ich hier zuletzt den Fakt, dass Software ein immaterielles Gut ist - man kann sie nicht anfassen. „Software“ ist ein relativ abstrakter Begriff, dies führt zu Schwierigkeiten, wenn es darum geht mit dem Begriff Software umzugehen.

All diese Probleme führten dazu ein technisches und planerisches Vorgehen bei der Softwareentwicklung zu verwenden, das Software Engineering.

Ein Teilgebiet dieses Software Engineering genannten Vorgehens ist, die Software Aufwandschätzung, auf die ich jetzt näher eingehen möchte.

2 Software Aufwandschätzung

Software Aufwandschätzung ist der Versuch durch die Analyse von influenten Faktoren den Aufwand und die Kosten für die Entwicklung, Instandhaltung und Pflege einer Software zu schätzen.

In den Anfängen der Softwareentwicklung wurde eigentlich keine Aufwandschätzung betrieben, aber je größer und komplexer die Projekte wurden, desto mehr kristallisierte sich etwa Anfang der achtziger Jahre heraus, dass so etwas wie Software Aufwandschätzung notwendig wird. Man braucht sich als aktuelles lokales Projekt nur die Entwicklung des Fahndungssystems Inpol für das BKA anzusehen, bei dem man schätzt, dass die ursprünglich eingeplanten Entwicklungskosten von ca. 20 Millionen Euro um weit mehr als 100 Millionen Euro überschritten wurden. Diese krasse Fehleinschätzung ist kein Einzelfall, etwas ähnliches ist schon unzählige Male vorgekommen und dies sogar noch in größeren Dimensionen. Hierbei ist es offensichtlich eine gute Idee vorher mit der entwickelnden Firma einen Vertrag über das Volumen des Projektes abzuschließen um wirtschaftliche Interessen wahren zu können und im Notfall Schadensersatz zu fordern. Dies geschieht mit Hilfe der Software Aufwandschätzung.

Der Sinn der Schätzung ist es, möglichst genaue Angaben zum benötigten Aufwand machen zu können, um zum Beispiel den Ablauf des Projekts planen zu können und die benötigte Arbeitszeit einzuschätzen oder um dem Kunden einen Preis nennen zu können.

Diese Schätzung erfolgt meist in einer bestimmten Maßeinheit, den so genannten Mitarbeitermonaten (auch kurz MM genannt oder PM für Personenmonate oder person months). Ein Mitarbeitermonat beschreibt die Leistung, die ein Mitarbeiter durchschnittlich innerhalb eines Monats vollbringt, dabei miteinbezogen ist schon Urlaub, Krankheit, Einarbeitung in das Projekt und andere Faktoren die die Produktivität des Mitarbeiters beeinflussen.

Mit Hilfe dieser Einheit der Mitarbeitermonate kann man mit den verschiedenen Methoden der Aufwandschätzung, relativ unabhängig von den Arbeitern der entwickelnden Softwarefirma Schätzungen abgeben, obwohl dabei bei extrem schlechten oder versierten Programmerteams natürlich Ausreißer in der Schätzung provoziert werden können. Außer in Mitarbeitermonaten kann man auch die Anzahl der Codezeilen schätzen, jedoch rechnet man die Anzahl der Codezeilen bei den meisten Schätzverfahren sowieso wieder auf Mitarbeitermonate um.

Falls die Schätzung jedoch direkt für den Endkunden erfolgt, so rechnet man die Mitarbeitermonate meist in die entstehenden Kosten in der jeweiligen Währung um, denn für den Kunden ist diese Information relevant, um zu entscheiden ob sich die Entwicklung eines bestimmten Produkts wirtschaftlich rechnet.

Die Schätzung wird durch eine oder mehrere schätzende Personen durchgeführt, welche mit einer bestimmten Schätzmethode systematisch oder einfach „pi mal Daumen“ versuchen, durch Analyse der diversen Einfluss nehmenden Faktoren den Aufwand zu schätzen.

Augenscheinlich ergeben sich hier schon einige Problemfelder der Software Aufwandschätzung, wie zum Beispiel den Einfluss der Kompetenz von Entwicklern und Schätzern. Ein weiteres Problem ist die unterschiedliche Größe von Software. Allzu oft werden Erfahrungen aus kleinen Projekten auf größere Projekte linear übertragen obwohl der Aufwand dort nicht unbedingt linear ansteigt.

3 Methoden der Software Aufwandschätzung

3.1 Empirische Schätzverfahren

3.1.1. Expertenschätzung

Bei dieser Methode beurteilen Experten aufgrund von bekannten Projekten und deren Aufwand den Aufwand eines zu analysierenden Projekts. Der Experte kann ein unabhängiger Schätzer sein, der mit der Materie vertraut ist, aber auch der Projektleiter oder bei einem kleinen Projekt der Programmierer selbst. Als Vorteil dieser Methode ist herauszustellen, dass sie einfach zu realisieren ist und ein sehr kostengünstiges Verfahren darstellt. Aber natürlich ist diese Methode oft ungenau, weil ein Projekt häufig nicht mit anderen hundertprozentig vergleichbar. Außerdem liegt es auf der Hand, dass die Qualität dieser Schätzung mit der Erfahrung und dem Können des Schätzers steht und fällt.

3.1.2. Delphi Methode

Bei der Delphi Methode wird versucht Expertenschätzungen zu objektivieren und zu relativieren, indem im ersten Schritt mehrere Experten unabhängig voneinander eine Schätzung abgeben. Dann im zweiten Schritt gibt es verschiedene Verfahrensweisen : Zum einen gibt es eine Variante bei der jeder Experte die Schätzungen der anderen erhält und daraufhin eine erneute Schätzung abgibt. Zum Anderen gibt es eine Variante bei der die Experten ihre Schätzungen an einen Diskussionsleiter abgeben, dieser die Schätzungen vergleicht und die Schätzer ihm Abweichungen vom Mittel erklären müssen. Eine weitere Variante ist, dass man sich nach jeder Schätzrunde an einen Tisch setzt und die Ergebnisse diskutiert. Auch weitere Varianten mit kleineren Abwandlungen werden praktiziert. Dabei sind jedoch immer Abweichungen vom Mittel zu begründen. Der zweite Schritt wird jetzt solange wiederholt bis alle Schätzungen nur noch eine bestimmte Abweichung untereinander aufweisen. Der Durchschnittswert aller Schätzungen der letzten Runde ist dann das endgültige Schätzergebnis. Die Delphi Methode liefert weitaus zuverlässigere Schätzungen als eine Expertenschätzung, indem Ungereimtheiten und ausreißende Schätzungen durch Diskussion und immer weitere Annäherung kompensiert werden. Auf der anderen Seite ist die Delphi Methode selbstverständlich weitaus kostenintensiver durch höheren Schätz- und Personalaufwand.

3.2 Algorithmische Schätzverfahren

3.2.1. Function Point Verfahren

Dieses Verfahren nutzt die so genannten Function Points (FP) zur Berechnung des Aufwands. Function Points sind erst einmal nur eine relative Größe zur Bewertung von Funktionalität. Das Function Point Verfahren wurde 1979 von Allan J. Albrecht bei IBM entwickelt und seither vielfach erweitert und ergänzt, zum Beispiel von der IFPUG (International Function Points User Group), deren Zahlverfahren für Function Points von 1994 hier dargestellt wird.

Um Function Points zu berechnen benötigt man fünf entscheidende Größen :

- x Dateneingaben (external input)
- x Datenausgaben (external output)
- x Anfragen (external inquiry)
- x Schnittstellen zu externen Datenbeständen (external interface file)
- x Interne Datenbestände (logical internal files)

Bei der zu analysierenden Software wird jeder dieser Größen ein ganzzahliger Wert zugewiesen, der einfach anhand der nachfolgenden Tabellen durch Klassifikation von Programmvorgängen in einfache, mittlere und komplexe Operationen gewonnen wird.

		Datenelemente in der Eingabe		
		1 – 4	5 – 15	> 15
Anzahl bearbeiteter Datenbestände	0 – 1	einfach	einfach	mittel
	2	einfach	mittel	komplex
	> 2	mittel	komplex	komplex

Gewichtungskriterien für Eingabedaten (IFPUG 1994)

		Datenelemente in der Ausgabe u. Anfragen		
		1 – 5	6 – 19	> 20
Anzahl bearbeiteter Datenbestände	0 – 1	einfach	einfach	mittel
	0 – 2	einfach	mittel	komplex
	> 3	mittel	komplex	komplex

Gewichtungskriterien für Ausgabedaten und Anfragen (IFPUG 1994)

		Datenelemente in den Datenbeständen und Referenzdaten		
		1 – 19	20 – 49	> 50
Anzahl bearbeiteter Datenbestände	1	einfach	einfach	mittel
	2 – 5	einfach	mittel	komplex
	> 5	mittel	komplex	komplex

Gewichtungskriterien für Datenbestände und Referenzdaten (IFPUG 1994)

Mit Hilfe der mit diesen drei Tabellen ermittelten Punkte kann man nun die sogenannten „Unadjusted Function Points“ (UFP) berechnen, auch Function Points Rohwert genannt. Dazu trägt man die Ergebnisse in die folgende Tabelle ein und bildet die Summe :

	Schwierigkeitsgrad			Summe
	einfach	mittel	komplex	
Dateneingaben	__ * 3 = __	__ * 4 = __	__ * 6 = __	__
Datenausgaben	__ * 4 = __	__ * 5 = __	__ * 7 = __	__
Anfragen	__ * 3 = __	__ * 4 = __	__ * 6 = __	__
externe Schnittstellen	__ * 5 = __	__ * 7 = __	__ * 10 = __	__
interne Datenbestände	__ * 7 = __	__ * 10 = __	__ * 15 = __	__
Unadjusted Function Points (UFP)				__

Schema zur Berechnung der UFP (IFPUG 1994)

Nach der Ermittlung der Unadjusted Function Points folgt nun noch die Berechnung des so genannten Technical Complexity Factors (TCF), bevor man sich endgültig an die Berechnung der Function Points begeben kann.

Nr.	Faktor	Wert
1	Datenkommunikation	
2	Verteilte Funktionen	
3	Leistungsanforderungen	
4	Belastung der Hardware	
5	Verlangte Transaktionsrate	
6	Online-Dateneingaben	
7	Effiziente Benutzerschnittstelle	
8	Online-Datenänderungen	
9	Komplexe Verarbeitungen	
10	Wiederverwendbarkeit	
11	Einfache Installation	
12	Einfache Benutzbarkeit	
13	Installation an mehreren Orten	
14	Änder- und Erweiterbarkeit	
Summe der Faktoren (TDI) :		

Einzusetzen sind Werte zwischen 0 und 5 wie folgt :

0 nicht vorhanden
1 unbedeutender Einfluss
2 mäßiger Einfluss
3 durchschnittlicher Einfluss
4 erheblicher Einfluss
5 starker Einfluss

Schema zur Bestimmung der Einflussfaktoren (IFPUG 1994)

Dieser Technical Complexity Factor beschreibt Anforderungen, Eigenschaften und Entwicklungsbedingungen für das zu analysierende Projekt. Um den Technical Complexity Factor zu berechnen, wird jeder einflussnehmende Faktor von „kein Einfluss“ (0 Punkte) bis zu „starker Einfluss“ (5 Punkte) bewertet. Dann werden die Punkte zusammenaddiert wie in obiger Tabelle und ergeben den Total Degree of Influence (TDI). Mit Hilfe des Total Degree of Influence kann man mit folgender Formel den Technical Complexity Factor berechnen :

$$TCF = 0,65 + 0,01 * TDI$$

Letzenendes hat man nun alle benötigten Werte zusammengestellt um mit Hilfe der folgenden Formel die Function Points berechnen zu können :

$$FP = UFP * TCF$$

Nun stellen die Function Points wie gesagt nur eine relative Größe dar. Die Aufwandschätzung in Mitarbeitermonaten mit Hilfe der Function Points erfolgt durch einen Vergleich mit geeigneten Tabellen oder Datenquellen.

In diesen Datenquellen wird dann einer bestimmten Anzahl von Function Points ein konkreter Wert an Mitarbeitermonaten zugeordnet. Dabei gilt es, die Vergleichsdaten von einem möglichst ähnlichen Projekt zu nehmen, beziehungsweise von einer ähnlich aufgebauten und programmiertechnisch erfahrenen Firma. Im Idealfall hat die durchführende Firma selbst schon eine Tabelle mit Werten aus vergangenen, vergleichbaren Projekten erstellt, an denen die Function Point Schätzung angelehnt werden kann.

In diesem Zusammenhang sollte noch kurz auf die so genannte Faustregel von Jones hingewiesen werden. Sie stellt eine Formel zur Berechnung des Aufwands mit Function Points dar, wenn man keine Vergleichswerte hat :

$$\text{Aufwand} = \text{Durchlaufzeit} * \text{Anzahl der Mitarbeiter}$$

hierbei sind :

$$\text{Durchlaufzeit (in Monaten)} = FP^{0,4}$$

$$\text{Anzahl der Mitarbeiter} = FP / 150$$

$$\text{Es folgt also : Aufwand} = FP^{0,4} * FP / 150$$

Natürlich ist diese Formel stark generalisiert und Ergebnisse aus ihrer Berechnung können oft weit von den realen Entwicklungszeiten abweichen, vor allem wenn man den Aufwand eines sehr kleinen Projekts damit berechnen möchte.

Insgesamt betrachtet wird klar, dass das Function Point Verfahren stark unternehmens- und projektspezifisch ist und ohne adäquate Vergleichswerte gestaltet sich das Schätzen schwierig. Somit ist es auch praktisch nicht übertragbar auf neuartige Entwicklungssituationen; da keine Erfahrungswerte vorhanden sind, deren Gewinn und Analyse oft sehr zeitraubend und kostenintensiv sein kann.

Augenfällig ist auch die Auslegung des Modells auf Informationssysteme, dies sieht man schon bei einem Blick auf die fünf Eingangsgrößen, womit sich Probleme bei der Übertragung auf anders aufgebaute Software ergeben. Leider fehlt beim Function Point Verfahren von 1994 auch der Bezug zur Objektorientierung.

Das Function Point Verfahren hat aber auch gute Seiten. Zum einen lassen sich benötigte Eingangsgrößen zur Berechnung in frühen Phasen eines Projekts leicht bestimmen und die Anwendung des Verfahrens ist nicht zuletzt wegen der durchschaubaren und klaren tabellarischen Struktur relativ einfach. Dies ist wohl auch der entscheidende Grund, warum das Function Point Verfahren das immer noch am häufigsten genutzte Verfahren ist.

3.2.2. COCOMO

COCOMO steht für **Constructive Cost Model** und wurde 1981 von Barry J. Böhm vorgestellt. Dieses Modell zur Software Aufwandschätzung basiert auf einer Schätzung der so genannten KDSI.

KDSI ist die Abkürzung für Kilo Lines of Devlivered Source Instructions, also tausend Zeilen Instruktionen im Source Code. Hierbei ist Vorsicht geboten, denn dies ist nicht gleich zu setzen mit der Anzahl der Codezeilen, sondern es handelt sich hierbei nur um Instruktionen (Schleifen, Zuweisungen, Vergleiche etc.).

Die Anzahl der KDSI wird entweder vom Programmierer oder Schätzer geschätzt oder mit Hilfe von Function Points (siehe 3.2.1) und Konvertierungstabellen, mit denen man die Function Points in KDSI umwandeln kann, ermittelt.

Bei COCOMO unterscheidet man zunächst 3 Arten von Projekten, jeder Projektart wird eine charakteristische Formel zur Bestimmung des Entwicklungsaufwandes und der Entwicklungszeit (TDEV) zugeordnet :

x Organic Mode : Der Organic Mode beschreibt ein kleines Projekt mit erfahrenen Programmierern.

Entwicklungsaufwand : $MM = 2,4 * KDSI^{1,05}$

Entwicklungszeit : $TDEV = 2,5 * MM^{0,38}$

x Semidetached Mode : Er beschreibt grössere Projekte als im Organic Mode, bei denen Programmierer mit unterschiedlichen Erfahrungsleveln mitwirken.

Entwicklungsaufwand : $MM = 3,0 * KDSI^{1,12}$

Entwicklungszeit : $TDEV = 2,5 * MM^{0,35}$

x Embedded Mode : Hierbei muss das Projekt strikt an äußere Gegebenheiten angepasst werden. Es existiert wenig Spielraum bei der Bearbeitung des Projekts und oft herrscht auch noch Zeitdruck. Der Embedded Mode beschreibt also schwierige Bedingungen.

Entwicklungsaufwand : $MM = 3,6 * KDSI^{1,05}$

Entwicklungszeit : $TDEV = 2,5 * MM^{0,32}$

Diese Unterscheidung ermöglicht erst einmal eine bessere Bezugnahme auf die Größe und Art des Projekts, was im Allgemeinen schon mal eine genauere Schätzung ermöglicht. Des weiteren bietet COCOMO nun aber noch eine weitere Spezifizierung bei der Schätzung und zwar mit Hilfe von 3 Stufen der Genauigkeit für grobe bis detaillierte Schätzungen :

x Basic Model

Die Berechnung im Basic Model erfolgt nach den gegebenen Formeln für den zutreffenden Projektmodus. Zuerst erfolgt die Schätzung der KDSI und damit dann mit der entsprechenden Formel die Berechnung der Zeit in Mitarbeitermonaten (MM) und der Entwicklungszeit (TDEV) durch einfaches Einsetzen und Ausrechnen.

x Intermediate Model

Die Berechnung im Intermediate Model erfolgt unter Hinzunahme weiterer Projektparameter, sogenannter Cost Drivers (CD, Kostenfaktoren). Der insgesamt Wert der Cost Drivers wird durch Klassifikation anhand der nachfolgenden Tabelle und anschließender Multiplikation der einzelnen Faktoren erreicht:

$CD = RELY * DATA * CPLX * TIME * STOR * VIRT * TURN * ACAP * AEXP * PCAP * VEXP * LEXP * MODP * TOOL * SCED$

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	.75	.88	1.00	1.15	1.40	
DATA		.94	1.00	1.08	1.16	
CPLX	.70	.85	1.00	1.15	1.30	1.65
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		.87	1.00	1.15	1.30	
TURN		.87	1.00	1.07	1.15	
ACAP	1.46	1.19	1.00	.86	.71	
AEXP	1.29	1.13	1.00	.91	.82	
PCAP	1.42	1.17	1.00	.86	.70	
VEXP	1.21	1.10	1.00	.90		
LEXP	1.14	1.07	1.00	.95		
MODP	1.24	1.10	1.00	.91	.82	
TOOL	1.24	1.10	1.00	.91	.91	
SCED	.91	1.08	1.00	1.04	1.10	

Kostenfaktoren nach Boehm (1981)

RELY : Required software reliability

DATA : Data base size

CPLX : Product complexity

TIME : Execution time constraint

STOR : Main storage constraint

VIRT : Virtual machine volatility

TURN : Computer turnaround time

ACAP : Analyst capability

AEXP : Applications experience

PCAP : Programmer capability

VEXP : Virtual machine experience

LEXP : Programming language experience

MODP : Use of modern programming practices

TOOL : Use of software tools

SCED : Required development schedule

Da man jetzt den CD Wert ermittelt hat kann man nun leicht die Berechnung des intermediate MM Wertes wie folgt durchführen:

$MM_{intermediate} = CD * MM_{Basic}$

Mit diesem intermediate MM Wert kann nun auch mit der zur Art des Projekts passenden Formel die Entwicklungszeit ausgerechnet werden.

x Detailed Model

Im Detailed Model erfolgt eine Unterteilung des Gesamtprojekts in kleinere Einheiten. Die Analyse dieser Einheiten wird wie im Intermediate Model durchgeführt und dann wird durch Aufsummieren der Ergebnisse, der zum Gesamtprojekt passende $MM_{Detailed}$ Wert, beziehungsweise der $TDEV_{Detailed}$ Wert ermittelt.

Zuerst wählt man also aus, ob man ein Projekt schätzt, welches dem Organic, Semidetached oder Embedded Mode zuzuordnen ist und dann führt man grobe bis detaillierte Schätzungen durch; je nachdem welche Schätzungsart man haben möchte, wie viel Zeit man für die Schätzung aufwenden möchte und wieviele Informationen schon über das Projekt vorliegen, beziehungsweise in welcher Phase der Entwicklung sich das Projekts schon befindet.

COCOMO stellt eine bewährte Methode zur Aufwandschätzung dar. Allerdings wird es weniger häufig genutzt als das Function Point Verfahren, denn die Schätzung scheint nicht so leicht durchzuführen zu sein und der mathematische Charakter von COCOMO wirkt offenbar abschreckend.

Wie bei anderen algorithmischen Modellen auch, ist es wünschenswert eine möglichst genaue Kalibrierung vorzunehmen, bei der durch Erfahrungswerte die Faktoren eines Projekts möglichst realitätsnah justiert werden.

Als großer Kritikpunkt ist zu nennen, dass COCOMO als eine der Hauptgrößen auf der Größe des Projekts (den KDSI) aufbaut, die von Experten geschätzt wird oder zum Beispiel mit Hilfe von Function Points berechnet wird. Damit macht sich COCOMO abhängig von der Fehleranfälligkeit anderer Verfahren.

Genau wie das Function Point Verfahren, nimmt auch COCOMO kaum Bezug auf Objektorientierung und moderne Programmierstechniken. Dies führte unabhängig zur Problemen in der heutigen Zeit und zog eine Anpassung von COCOMO in Form von COCOMO II nach sich.

3.2.3. COCOMO II

COCOMO II steht für **Constructive Cost Model II** und wurde ebenfalls von Barry J. Boehm entwickelt. Vorgestellt wurde COCOMO II 1995 und es stellt im weitesten Sinne eine zeitgemäße Erweiterung von COMOMO dar.

COCOMO II berücksichtigt :

- x Rapid-development Modelle
- x Schlüsselfertige Programmteile
- x Wiederverwendung von Code
- x Komposition von Programmen
- x Automatisch generierter Programmcode
- x Objektorientierte Ansätze

Dies alles sind mittlerweile gängige Praktiken durch deren Einbeziehung Boehm konsequent der Entwicklung der Softwareentwicklung Rechnung trägt.

Eine gravierende Änderung von COMOMO II gegenüber COCOMO ist, dass es nur noch eine Mitarbeitermonate Grundgleichung gibt. Diese berechnen sich mit der Formel :

$$MM_{Nominal} = A * (KDSI)^B$$

Der Faktor A ist eine Konstante, die dazu dient unternehmensspezifische Effekte einzufangen. Der Faktor B beschreibt verschiedene Größenvorteile und Nachteile bei der Entwicklung von Softwareprojekten unterschiedlicher Größe.

Auch wird bei COCOMO II wieder zwischen drei Schätzarten unterschieden, diesmal mit größerer Betonung auf den Entwicklungsstand des Projekt :

- x Application Composition

Die *Application Composition* basiert auf so genannten Object Points, die lassen sich anhand folgender Faktoren berechnen lassen : die Anzahl angezeigter Bildschirme, die Anzahl erzeugter Berichte und die Anzahl der 3GL-Module. Hierbei wird klar, dass die Application Composition auf moderne Entwicklungsumgebungen abzielt, bei welchen diese Faktoren überhaupt anwendbar sind.

- x Early Design

Beim *Early Design* Modell erfolgt die Hinzunahme weiterer 7 Aufwandsfaktoren (Effort Multipliers) zur genaueren Bestimmung der Mitarbeitermonate wie folgt :

$$MM_{EarlyDesign} = EM_{EarlyDesign} * MM_{Nominal}$$

$$EM_{EarlyDesign} = RCPX * RUSE * PDIF * PERS * PREX * FCIL * SCED .$$

Die Aufwandsfaktoren beschreiben folgende Größen :

- RCPX : Product Reliability
- RUSE : Required Reuse
- PDIF : Platform Difficulty
- PERS : Personnel Capability
- PREX : Personnel Experience
- FCIL : Facilities
- SCED : Schedule

Wie der Name schon sagt, kann das Early Design Modell schon in der frühen Entwicklungsphase des Projekts angewandt werden, da die oben genannten Effort Multipliers dann schon abzusehen sind.

- x Post Architecture

Beim *Post Architecture* Modell erfolgt eine erneute Berechnung der Mitarbeitermonate, diesmal mit 17 Aufwandsfaktoren (Effort Multipliers), denn zu diesem Zeitpunkt der Schätzung sind mehr Informationen über das Projekt verfügbar als im Early Design, deshalb kann man diese Faktoren nun auch zur Berechnung verwenden.

$$MM_{PostArchitecture} = EM_{PostArchitecture} * MM_{Nominal}$$

Wobei sich der $EM_{PostArchitecture}$ wie folgt zusammensetzt :

$$EM_{PostArchitecture} = RELY * DATA * CPLX * DOCU * RUSE * TIME * STOR * PVOL * ACAP * PCAP * PCON * AEXP * PEXP * LTEX * TOOL * SITE * SCED$$

Ich möchte diese Faktoren hier nur auflisten, denn eine komplette Ausführung der Faktoren und der Errechnung ihrer Werte würde den Umfang dieser Ausarbeitung zweifellos sprengen, aber an diesen Faktoren wird die Richtung klar, welche die COCOMO II Entwickler eingeschlagen haben, dahingehend Methoden des modernen Softwareengineerings in die Aufwandschätzung zu bringen.

- RELY : Required Software Reliability
- DATA : Data Base Size
- CPLX : Product Complexity
- DOCU : Documentation match to life-cycle needs
- RUSE : Required Reusability
- TIME : Execution Time Constraint
- STOR : Main Storage Constraint
- PVOL : Platform Volatility
- ACAP : Analyst Capability
- PCAP : Programmer Capability
- PCON : Personnel Continuity
- AEXP : Applications Experience
- PEXP : Platform Experience
- LTEX : Language and Tool Experience
- TOOL : Use of Software Tools
- SITE : Multisite Development
- SCED : Required Software Schedule

Abschließend gesehen kann man sagen, dass COCOMO II neue Cost Driver Faktoren einbezieht, die mit den Jahren in der Softwareentwicklung Einzug gehalten haben und somit auch bei objektorientierten Softwareprojekten weitaus genauere Schätzungen als zum Beispiel das Function Point Verfahren und COMOMO erlaubt und auch weitaus besser moderne Mittel der Programmierung in die Schätzung miteinbezieht. COCOMO II versucht besser auf die Phasen während der Entwicklung einer Software und die zu bestimmten Zeitpunkten vorhandenen Informationen einzugehen. Schätzungen vorzunehmen ist allerdings recht schwierig aufgrund der komplexen Formeln und der vielen Faktoren in COCOMO II. Fraglich ist deshalb auch ob COCOMO II vor allem bei Schätzungen von kleinen und mittleren Projekten, bei denen nicht viel Schätzaufwand betrieben wird, häufig Verwendung finden wird.

3.3 Andere Schätzverfahren

Es gibt noch viele weitere Schätzverfahren, die in der Praxis seltener Verwendung finden von denen ich aber hier einige der Vollständigkeit halber aufzählen möchte :

- x Top-Down Schätzung
- x Bottom-Up Schätzung
- x Surböck Verfahren
- x Object Point Verfahren
- x Esterling Methode

Keinesfalls zu vergessen die von Martin Glinz aufgeführten amüsanten gängigen Methoden :

- x So viel schätzen, dass man den Auftrag auf jeden Fall bekommt.
- x So viel schätzen, wie der Auftraggeber zu zahlen bereit ist
- x So viel schätzen, wie Arbeitskapazität vorhanden ist.

4 Résumé

Software Aufwandschätzung erweist sich als schwieriges Feld, denn es müssen viele externe und interne Faktoren beachtet werden, welche Schätzungen beeinflussen. Die in diesem Text beschriebenen Schätzverfahren sind wohl die gängigsten und versuchen die Aufgabe der Schätzung mit unterschiedlichen Methoden und Ansätzen zu meistern. Jedoch egal ob empirische, algorithmische oder eine andere Schätzung, es ist eigentlich immer nur möglich annähernde Schätzungen abzugeben und mehr sollte man auch nicht von der Software Aufwandschätzung erwarten, denn es spielen unglaublich viele Faktoren in den nachher tatsächlich benötigten Aufwand hinein, so dass eine absolut genaue Berechnung (und nicht Schätzung) der Kosten erst nach Abschluss des Projekts möglich ist.

In der Qualität der Aufwandschätzung spielt natürlich der Schätzer selbst eine große Rolle. In dem einen Verfahren ist dies mehr der Fall als in anderen, aber die Kompetenz, Erfahrung und Integrität des Schätzers beeinflussen das Ergebnis der Schätzung maßgeblich. Bei Verfahren, die zur Ermittlung des Aufwands auch noch Vergleichswerte heranziehen, ist es selbstverständlich umso wichtiger möglichst passgenaue Referenzdaten zu haben, die in möglichst vielen Punkten mit der durchzuführenden Schätzung übereinstimmen.

Für die Zukunft ist sicherlich zu erwarten, dass es neue Verfahren geben wird, die sich noch mehr auf Objektorientierung und aktuelle Programmierverfahren beziehen, denn auch wenn mit dem 1999 nochmal aktualisierten Function Points Counting Practise Manual beim Function Point Verfahren weitere Faktoren miteinbezogen werden und auch bei COCOMO II als konsequente Erweiterung von COCOMO versucht wird, die Gegebenheiten aktueller Programmierung einzufangen, sind beide Verfahren nur Aktualisierungen ihrer Vorgänger, die sich natürlich nicht vollständig von ihren Wurzeln gelöst haben und sind damit nicht so flexibel und gut angepasst wie ein komplett neuartiges, speziell auf heutige Entwicklungsbedingungen ausgerichtetes Verfahren.

Die kommenden Verfahren sollten versuchen Fehlerquellen zu minimieren, vor allem die Schätzlast und Einbindung des Menschen zu minimieren. Sie sollten möglichst viele Einfluss nehmende Faktoren miteinbeziehen, dazu flexibel und schnell anpassbar sein, um mit dem Tempo der Entwicklung der informationsverarbeitenden Industrie stand zuhalten. Nicht zuletzt sollten kommende Verfahren möglichst verständlich und einfach anzuwenden sein, denn sonst stehen ihre Chancen ungleich schlechter eine möglichst große Anwendung und Verbreitung zu finden.

All dies ist ganz klar nicht leicht zu vereinbaren, aber wir werden sehen was die Zukunft an neuen Schätzverfahren hervorbringt.

Quellenangaben

IEEE 6.10.12 - Standard Glossary of Software Engineering Terminology - IEEE, 1990

Function Point Counting Practices Manual - IFPUG, 1994

Software Engineering Economics - Barry J. Boehm, 1981

COCOMO II Model Definition Manual – Barry J. Boehm u.a., 1995

Software Engineering Skript v. Martin Glinz Universität Zürich - Martin Glinz, 2003
http://www.ifi.unizh.ch/groups/req/courses/se_I/

Software Engineering Skript MATA/RWTH Aachen - Kraft & Gatzmeier, 2003
http://www.rz.rwth-aachen.de/mata/veranstaltungen/software_eng.php

Die Function Point Methode - Lutz Michaelson, 2001
<http://danae.uni-muenster.de/~lux/seminar/ss01/Michaelson.pdf>

Alle Links geprüft am 29.09.2004