Aufwandschätzung und Produktivität in Softwareprojekten

- Ergebnisse aus industriellen Entwicklungen -

Thorsten Spitta
Vatter GmbH Rheine/Schongau

Original Word 2.0. Daher bei quer "gelegten" Bildern verstümmelt.

Zusammenfassung

Es werden die üblichen Verfahren zur Aufwandschätzung grob dargestellt und bezüglich ihrer Praktikabilität bewertet. Danach werden die Schätzung, der tatsächliche Aufwand und die Produktivität von fünf Projekten ausgewertet. Die Ergebnisse werden diskutiert und hieraus Empfehlungen für eine empirische Schätzbasis abgeleitet. Mit der Schätzbasis wird gleichzeitig eine Datenbasis gewonnen, mit der die Produktivität der Softwareentwicklung beurteilt werden kann. Damit ist eine Grundlage für die Verbesserung der Produktivität geschaffen.

Stichworte: Aufwandschätzung, Function Points, Produktivitätsmessung, 4GL.

1 Einführung

Der folgende Aufsatz soll anhand von fünf Softwareprojekten die Möglichkeiten und Grenzen von Aufwandschätzungen unter industriellen Bedingungen zeigen. Außerdem werden im Nachhinein durchgeführte Produktivitätsvergleiche diskutiert. Hierdurch lassen sich Fragen zur Wirtschaftlichkeit der verwendeten Programmierwerkzeuge beantworten, speziell die Frage, ob eine "Sprache der 4. Generation" (4GL) wirtschaftlicher ist als eine konventionelle Sprache wie etwa COBOL.

Aufwandschätzung und Produktivitätsmessung hängen direkt zusammen, denn es gilt:

$$\mathrm{Produktivität} = \frac{output}{input}.$$

Bei der Aufwandschätzung wird output (z.B. LOC) geschätzt, um input (z.B. Stunden) ermitteln zu können. Dabei wird eine bestimmte Produktivität unterstellt. Bei der Produktivitätsermittlung sind beide Größen bekannt.

Im folgenden werden die aus der Literatur bekannten Schätzmethoden kurz dargestellt, ihre Verwendbarkeit unter industriellen Bedingungen diskutiert und dann auf den Verlauf der Fallstudien angewendet. Die empirischen Ergebnisse werden unter Produktivitätsgesichtspunkten bewertet und daraus Hypothesen abgeleitet.

2 Verfahren zur Aufwandschätzung

Bei allen aus der Literatur bekannten Verfahren geht es um zwei Fragen:

1. Welches Verfahren wird bevorzugt bzw. ist für die gegebene Problemstellung angemessen?
2. Welche Größen werden geschätzt?

Nach Boehm [7] sind folgende grundlegenden Verfahren zu unterscheiden:

- **algorithmische Modelle:** Auf Basis einer geschätzten Größe, meist LOC, wird der voraussichtliche Aufwand errechnet. Die Parameter der Schätzgleichungen der verschiedenen Verfahren basieren auf empirischen Untersuchungen. Das bekannteste Modell dieser Art ist COCOMO von Boehm [6]. Es muß genau definiert werden, welche Bestandteile des Quellcodes man in die Größe LOC einbezieht und welche nicht.
- **Expertenschätzung:** Experten unterschiedlicher Spezialisierungen schätzen und diskutieren den Aufwand, bis Einigkeit erzielt wird.
- **Analogieschluß:** Der Aufwand wird durch Analogieschluß mit ähnlich gelagerten, abgeschlossenen Projekten geschätzt.
- **Parkinson:** Wir haben Leute und werden sie zu beschäftigen wissen. Anmerkung: Dem Autor ist dieser

Fall in 15 Jahren Industrietätigkeit noch nicht begegnet.
- **Preis für den Gewinner:** Der günstigste Schätzer -meist ein externer Anbieter- erhält den Zuschlag. Dies ist besonders dann problematisch, wenn keine Qualitätsprüfung stattfindet.
- **Top-Down:** Die Kosten werden global geschätzt (also in diesem Fall der input) und dann in Pakete eingeteilt.
- **Bottom-up:** Der Aufwand für jede Komponente wird geschätzt und die Schätzwerte addiert.

Die letzten beiden Ansätze unterstellen ein entsprechendes Entwicklungsparadigma. Allen Konzepten ist gemeinsam, daß die Schätzgenauigkeit zu Projektbeginn extrem klein ist und mit dem Entwicklungsfortschritt zunimmt. Dies ist eigentlich trivial, doch befindet sich jeder Auftraggeber in der Situation, weit vor der Spezifikation ein Budget bereitstellen oder das Projekt verwerfen zu müssen. Es gilt heute als sicher, daß erst auf Basis einer Spezifikation Festpreise angegeben werden können. Die Schätzungenauigkeit zum Ende der Spezifikation beträgt nach Boehm immerhin noch +/- 33% [7, p.8], das wären z.B. Projektkosten, die von 670.000 DM bis 1,3 Mio DM streuen.

Die Problematik algorithmischer Modelle sei am Beispiel von COCOMO gezeigt [vgl. 7, p.13ff]: Für 'embedded software' wird der Aufwand in Mitarbeitermonaten (MM) wie folgt ermittelt:

$$MM = 2{,}8\,(LOC)^{1{,}2} * f \quad \text{mit f= 1...1,5.}$$

Der Gewichtungsfaktor **f** wird aus einer Vielzahl von Einflußfaktoren ermittelt, die qualitative Größen der Problemstellung quantifizieren, z.B. Komplexität, Fähigkeiten der Entwickler, Verfügbarkeit von Tools usw.

Man muß also eine Größe des Produktes auf der technischen Ebene schätzen, die erst sehr spät verläßlich angegeben werden kann. Außerdem ist die Größe LOC stark von der Programmiersprache abhängig, wenn man bedenkt, daß sich die Kompaktheit des Codes zwischen Assembler, COBOL und 4GL nach Boehm wie 8 : 3 : 1 verhält [8, p.55f.].

Schon früh gab es Bemühungen um andere quantitative Maße als Schätzgrößen, die die Komplexität der Software besser erfassen als LOC. Hier seien ohne nähere Beschreibung Jackson [11], McCabe [14], und Orr [16] erwähnt, die die Komplexität der Daten- oder der Kontrollstrukturen messen. Die Problematik aller dieser Schätzverfahren ist, daß die Schätzgrößen zu dem Zeitpunkt, zu dem eine Schätzung gebraucht wird, nicht zur Verfügung stehen, da sie am Produkt (der Software) ansetzen und nicht am Problem (den Daten oder der Funktionalität).

Diesem Mangel begegnet das Verfahren **Funktionspunkte** von Albrecht [1], das unter die bottom-up-Verfahren fällt. Hierbei muß eine -zumindest grobe- Spezifikation vorliegen, um folgende Größen zu ermitteln: Die Zahl der Eingaben, Ausgaben, Abfragen, gemessen in Bildschirmseiten bzw. Fenstern, sowie die Zahl der (logischen) Dateien und Schnittstellen. Sie werden nach Schwierigkeitsgrad der Software gewichtet und die gewichteten Funktionspunkte addiert. Das Ergebnis wird noch einmal um qualitative Einflußgrößen erhöht oder verringert. Das Verfahren wurde bei IBM entwickelt und mehrere Jahre empirisch erprobt [vgl. 1, p.640].

Aus Sicht der Entwicklung von Software praktischer Größenordnungen und unter den Restriktionen eines industriellen Anwenders (hoher Zeitdruck, Budgetknappheit, i.d.R. aber auch keine sicherheitsrelevanten Systeme) bleiben folgende Schätzverfahren als praktisch anwendbar übrig:
- Expertenschätzung,
- Analogieschluß,
- bottom-up-Kalkulation.

Die bottom-up-Kalkulation wird vom Autor als das verläßlichste, der Analogieschluß als das riskanteste Schätzverfahren gesehen. Beim Analogieschluß neigt der Schätzer dazu, die Ähnlichkeiten über- und die Unterschiede unterzubewerten. Überhaupt nicht tauglich für Schätzungen ist die pauschale Angabe der prozentualen Einteilung eines geschätzten Gesamtaufwandes (top-down-Verfahren) auf Phasen bzw. Entwurfsschritte (**VOR**studie, **IST**analyse, **S**o**LL**konzept, **SP**e**Z**ifikation, **KON**struktion, Codierung und

Te**ST**, **INT**egration, **EIN**führung). Hierzu einige Angaben aus der Literatur, die bereits in der Definition der Schritte und nicht nur bei den Prozentangaben variieren: <Tab.1 Folgeseite>

Man sieht, daß Boehm als ein maßgeblicher Autor seine Zahlenangaben im Laufe der Zeit erheblich revidiert hat, und zwar im Sinne einer Aufwandsverschiebung in die Realisierungsphasen. Dies könnte mit einer Ernüchterung über die Möglichkeiten der 1975 mit viel Euphorie gesehenen Disziplin

\ Autor / Entwurfsschritt \	Brooks (1975)	Boehm (1975)
Vorstudie/Istanalyse (Planung)		
Sollkonzept	33	44
Spezifikation		
Software-Entwurf		
Codierung	16	28
Komponententest	25	28
Integration/Systemtest	26	
Einführung		
Dokumentation		

Tab. 1: Prozentualer Aufwand je Entwurfsschritt in der Literatur

Software Engineering zusammenhängen.

Als Schätzbasis sind diese Angaben offensichtlich nicht verwendbar. Ganz grob kann man aber daraus schließen, daß ca. 50% des Aufwandes durch die Programmiersprache und die Entwicklungsumgebung beeinflußt werden können.

Bezüglich einer detaillierten Darstellung der verschiedenen Schätzverfahren in deutscher Sprache sei auf das Buch von Noth und Kretzschmar verwiesen [15]. Dort werden eine Vielzahl von Verfahren experimentell überprüft mit dem Ergebnis, daß die meisten Verfahren außerhalb der ursprünglichen Umgebung zu nicht nachvollziehbaren Resultaten führen oder daß Schätzgrößen verlangt werden, die erst am Projektende bekannt sind.

Im folgenden werden nach einer Kurzdarstellung der fünf Softwareprojekte die empirischen Ergebnisse dargestellt und interpretiert.

3 Projekte und Datenerhebung

3.1 Beschreibung der Projekte

TOUR: *Tourenplanung Außendienst*

Anlaß: 140 Reisende besuchen täglich bundesweit 1.120 Läden nach Vorgaben eines Tourenplanungsprogrammes. Das vorhandene Programmsystem ist batchorientiert und enthält hartcodierte Besuchsdistanzen. Es ist nicht mehr durchschaubar. Die Altsoftware behindert einen wirtschaftlichen Einsatz des Außendienstes und soll daher erneuert werden.

Typ: Online mit Batch-Listen; Datenbank mit Brückenprogramm zu einer konventionellen Stammdatei.

Programme:

	TOUR1	TOUR2
online	17 NATURAL1	30 NATURAL2
batch	4 NATURAL1	11 NATURAL2
	6 CONLIST (Generator + COBOL)	

Entwicklungsumgebung: ADABAS/NATURAL online-System unter IBM/CICS.

Versionen: TOUR1: 1,5 Jahre
TOUR2: ab 10/89

PRAEM: *Prämiensystem Außendienst*

Anlaß: Die umsatzabhängige Entlohnung der Außendienstmitarbeiter soll auf eine Prämienvergütung umgestellt werden, da die bisherige Vergütung Retouren am Saisonende nicht "bestraft". Stattdessen soll tourgerechtes Verhalten belohnt werden, damit sichergestellt ist, daß in den Läden kein Umsatz verlorengeht, weil im Warenträger Ware fehlt.

Typ: Batch mit online-Parameterpflege; konventionelle Dateien mit Datenbank.

Programme: 10 COBOL (groß und komplex)
46 NATURAL1 (sehr einfach)

Entwicklungsumgebung: IBM VSE/SP für Batchprogramme (COBOL)

Versionen: Keine geplant; es gibt eine Betriebsvereinbarung.

DE-PC: *Datenerfassung für Host auf PC*

Anlaß: Die Firma muß täglich Massendaten in sehr kurzer Zeit in den Host-Rechner eingeben. Dies ist mit dem

Transaktionsmonitor auf dem Host technisch nicht möglich. Der für die Datenerfassung benutzte Vorrechner ist fast 10 Jahre alt und wird nicht mehr hergestellt. Die Wartung läuft aus. Es soll über PC (XT 1988!) erfaßt werden, wobei unklar ist, ob das technisch mit der notwendigen Erfassungsgeschwindigkeit möglich ist.
Typ: offline Datenerfassung am Bildschirm.
Programme: 103 COBOL.
Entwicklungsumgebung: MF-Workbench; COBOL auf PC mit Maskengenerator.
Versionen: Keine geplant, da die Lösung nach 3-4 Jahren durch mobile Datenerfassung abgelöst werden soll.

LGSTEL: Erweiterung *Lagerbestandsführung* um *Stellplatzverwaltung*

Anlaß: Wegen Aufgabe eines Standortes zu einem festen Stichtag muß die Möglichkeit geschaffen werden, mit weniger Lagerfläche für Fertigware auszukommen. Hierzu muß die in einem sehr schlechten Zustand befindliche Altsoftware um eine Stellplatzverwaltung erweitert werden, da die Zeit nicht ausreicht, eine moderne Fertigsoftware zu installieren.
Typ: online-Software, 12 Jahre alt.
Programme: 18 COBOL neu (einfach), 4 COBOL geändert (sehr komplex).
Entwicklungsumgebung: IBM VSE/SP für Dialogprogramme (COBOL/CICS).
Versionen: Keine geplant, da die Ablösung des Systems überfällig ist. Mit Inbetriebnahme des Systems mußte es "eingefroren" werden, da Änderungen zu gefährlich wurden. Es gab unerklärliche Abstürze im Produktionsbetrieb.

3.2 Datenerhebung

Empirische Erhebungen aus Softwareprojekten haben zwei Quellen:
1. **Produktdaten:** Sie lassen sich mit Werkzeugen ermitteln, etwa die Zahl der Programmzeilen abzüglich der Kommentare oder die Zahl der bedingten Anweisungen und Schleifen.
2. **Aufwandsdaten:** Sie beruhen auf Aufschreibungen (Kontierungen) der Entwickler.

Detailliert äußern sich Basili und Rombach [2] zur Erfassung von Daten.

Da viele Entwickler an der Entstehung der Aufwandsdaten mitwirken, fließen zwangsläufig subjektive Einflüsse in die Ergebnisse ein. Es ist nicht durchführbar und auch nicht ratsam, alle Kontierungen einzeln zu hinterfragen. Also wird angenommen, daß sich subjektive Verfälschungen durch die große Zahl der Entwickler gegenseitig aufheben. Da man die Beachtung von Kontierungsvorschriften nicht überwachen kann, ist es nicht empfehlenswert, zu stark ins Detail gehende Aufschreibungen zu verlangen, da hiermit die Wahrscheinlichkeit für systematische Fehler wächst.

Von den drei Kontierungsmöglichkeiten, Zeitaufwand je
- Projekt,
- Projekt und Phase / Entwicklungsschritt,
- Projekt, Phase und Aktivität,

wurde die mittlere Möglichkeit gewählt. Die Aktivität ist in den von den Entwicklern erfaßten Texten enthalten (Freitext "Tätigkeit"). Sie kann ohne formale Konventionen für die Erfassung nicht maschinell ausgewertet werden. Dies ist ein Nachteil, der allerdings folgenden Vorteil mit sich bringt: Bei einer manuellen Auswertung besteht ein Zwang, Unklarheiten im Gespräch mit den Entwicklern zu hinterfragen. Grundlage eines solchen Systems muß eine Vertrauensbasis zwischen Entwicklern und Management sein und nicht eine Kontrollsituation. Üblicherweise wird die Tätigkeit nicht ausgewertet.

Damit sowohl Erfassung als auch Auswertung effizient erfolgen können, wurde eine online-Erfassung in einer Datenbank gewählt. Näheres zu diesem Projekt- und Wartungsmanagement-System in [17, Kap.12]. In dem System sind inzwischen in 6 Jahren ca. 130 MJ Entwicklungsaufwand aus über 90 Projekten und ca. 400 Wartungsaktivitäten von über 50 Entwicklern festgehalten.

Es dürfte deutlich geworden sein, daß das Erheben von Daten in Industrieprojekten nicht den Ansprüchen streng kontrollierter Experimente genügen kann.

4 Schätzungen und Aufwände

4.1 Aufwand je Phase

Die sehr verschiedenen Projektsituationen und -abläufe spiegeln sich in der Aufwandsverteilung je Phase wider:

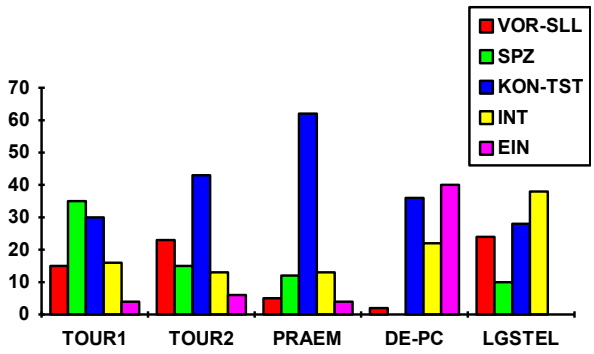

Abb. 1: Prozentuale Aufwandsverteilung je Phase

Die der Grafik zugrundeliegenden Werte sind weiter hinten in Tab.3 enthalten. Die Verteilung läßt viele Interpretationen über die Projektverläufe zu, die hier nicht ausgeführt werden können. Lediglich zwei auffällige Erscheinungen seien erläutert:

- Bei DE-PC fehlt die Spezifikation, entsprechend aufwendig ist die Einführung.
- Bei LGSTEL fehlt die Einführung. Sie fällt mit der Integration zusammen, da es kein Testsystem gab. Es mußte nachts im Produktivsystem integriert werden, während tagsüber das unveränderte Altsystem produktiv war.

Diese beiden Projekte liefen unter der Bedingung eines umumstößlichen Fixtermines ab, nämlich eines Umzuges in einen anderen Standort. Die Ergebnisse erhärten die Literaturbefunde: Fast jedes Projekt läuft in einer speziellen Konstellation ab, die die prozentualen Aufwände je Phase stark gegeneinander verschiebt. Für Schätzungen sind Phasenanteile ungeeignet. Sie lassen jedoch im Nachhinein Schlüsse auf Verbesserungsmöglichkeiten bei der Projektabwicklung zu.

4.2 Schätzungen und Istaufwände

Folgende Schätzverfahren wurden angewendet:

- Expertenschätzung auf Basis von
  - Spezifikation
  - Systemkonstruktion

  bei Batchsoftware (PRAEM),
  - Prototyp
  - Spezifikation

  bei Dialogsoftware (TOUR1).
- Bottom-up-Kalkulation auf Basis
  - Vorstudie/Istanalyse
  - Spezifikation
  - Änderungs-Konstruktion

  bei Altsoftware (LGSTEL).
- Function-point-Methode (FP) auf Basis
  - Prototyp
  - Spezifikation
  - Datenmodell
  - Nachkalkulation Version 1

  bei Versionen (TOUR2).

Nach Auswertung der Projekte stellte sich heraus, daß es ratsam ist, in Zukunft alle Projekte mit FP zu schätzen. Warum wurde nicht einheitlich geschätzt?

1. FP kann man für Schätzungen erst anwenden, nachdem man eine Erfahrungsbasis durch Nachkalkulation mehrerer Projekte geschaffen hat.
2. Die Entwickler waren mit der erstmaligen Anwendung eines SE-Vorgehensmodells gefordert. Zu viele Neuerungen sollten vermieden werden.
3. FP kann man nur auf Basis einer Spezifikation anwenden. Diese fehlte z.B. bei DE-PC.

Bis auf TOUR2 mußte bei allen Projekten die ursprüngliche Schätzung nach oben revidiert werden. Dies erfolgte nicht infolge nachgeschobener Anforderungen, sondern wegen zu geringer Kenntnisse über den Istzustand der abzulösenden Systeme. Bei DE-PC betraf dies den Leistungsumfang, bei den anderen Systemen den **Zustand der Altdaten**, der zu einer erheblichen Aufblähung der Integrationstests führte. Die Ergebnisse:

<Tab.2 Folgeseite>

5 Projektergebnisse

5.1 Funktionspunkte

Funktionspunkte lassen sich verläßlich und mit relativ wenig Aufwand ermitteln. Subjektive Einflußfaktoren fließen lediglich bei der Bestimmung der Gewichtungsfaktoren ein. TOUR2 wurde auf der Basis von Funktionspunkten von einem Softwarehaus kalkuliert, bei den übrigen Systemen wurden die Funktionspunkte im Nachhinein ermittelt. Das System DE-PC wurde als einfach, die übrigen als schwierig gewichtet. Die Ergebnisse können hier aus Platzgründen nicht im einzelnen dargestellt werden und sind summarisch in Tab.3 weiter hinten enthalten.

Fall	Schätzbasis	Schätz werte Zeit Monate	Aufwand MM
TOUR1	Spezifikation	7	14,5
	Testentwurf	2	7
	å	**9**	
TOUR2	TOUR1 und SPZ	**3**	
PRAEM	SPZ und KON	4	10
	Testentwurf	2	3
	å	**6**	
DE-PC	Prototyp	3	2
	SPZ und KON	4	5
	å	**7**	
LGSTEL	Analyse Altsystem	2	3
	SPZ und Änderungs-KON	1	1
	å	**3**	

Tab. 2: Schätzungen, Nachschätzungen und Istaufwand der Projekte

5.2 Produktivität und sonstige Werte

In Tab.3 sind noch einmal die Randbedingungen der Systeme (Entwicklungsumgebung, Entwickler, Benutzer), die Aufwände und Kosten und die entsprechenden Produktwerte (Sprache, LOC, Funktionspunkte) zusammengestellt. Hieraus wurden folgende Produktivitätsmaße ermittelt:

- **LOC/Std** zur Ermittlung des allgemein üblichen Maßes.
- **LOC/FP** zur Ermittlung eines Maßes für die Kompaktheit des Quellcodes.
- **DM/LOC** zum Vergleich der Wirtschaftlichkeit der Entwicklungsumgebung, verknüpft mit der Projektsituation. Dieses Maß hat mehr informativen Charakter, um das folgende besser bewerten zu können. Bei DM/LOC wirken sowohl eine unproduktive Entwicklungsumgebung als auch eine große Problemkomplexität kostenerhöhend.
- **DM/FP:** Dieses Maß eliminiert den Einfluß der Problemkomplexität. Es scheint gut geeignet, die Wirtschaftlichkeit der Entwicklungsumgebung zu messen.
- **FP/Tag:** Dieses Maß eliminiert gegenüber dem vorherigen noch inflationäre Einflüsse. Es ist deshalb vor allem für langfristige Produktivitätsvergleiche geeignet ["Tag"= 8 Std].

Die Ergebnisse finden sich in Tab.3. <Folgeseite>.

6 Folgerungen

6.1 Interpretation der Ergebnisse

In den Projekten wurde NATURAL1 als typischer Vertreter aus der Vielzahl von 4GL benutzt, von denen hier nur FOCUS, MANTIS, PROGRESS und CSP erwähnt seien. Hierfür wird eine um bis zu 60% höhere Produktivität gegenüber COBOL angegeben, allerdings auf der Basis relativ kleiner Programmbeispiele [Boehm, 9, p.1464; Jones, 13, S.165].

NATURAL2 ist eine universelle Programmiersprache auf Basis einer 4GL.

Aus den Ergebnissen der Projektstudien können keine allgemeinen Schlüsse gezogen werden. Es ist jedoch zulässig, erste Hypothesen zu folgenden Fragen aufzustellen:

- Ist es sinnvoll, Funktionspunkte zu ermitteln und damit zu schätzen?
- Ist NATURAL1 als typische 4GL produktiver als COBOL?
- Ist NATURAL2 als vollwertige Programmiersprache produktiver als NATURAL1 und COBOL?
- Hat die bei NATURAL deutlich einfachere Entwicklungsumgebung einen erkennbaren Einfluß auf die Produktivität?

\ **System** / **Entwicklungsdatum**	**TOUR1**	**TOUR2**
Randbedingungen:		
- Entwicklungsumgebung	NAT1/370	NAT2/370
- Programmiersprache	NATURAL1	NATURAL2
- Zahl Entwickler	10	9
- Zahl Benutzer	140 + 2	140 + 2
- Dauer [Monate]	10	10
Aufwand [Stunden (Std)]		
- VOR/IST/SLL	511	427
- Spezifikation	1.239	273
- KON/Modultest	1.058	791
- Integration	577	233
- Einführung	135	115
å [Std]	**3.523**	**1.839**
Kosten [DM]	394.730	203.100
Produktmaße:		
- LOC		
. gesamt	17.350	13.400
. *bereinigt*	11.300	9.126
- FP		
. *ung*ewichtet	270	372
. *gewichtet*	324	372
Ergebnisse:		
- LOC/Std		
. gesamt	4,9	11,2
. bereinigt	3,2	7,6
- LOC/FP-ungewichtet	41,9	24,5
- DM/LOC	34,90	22,30
- DM/FP 1)	1.218	546
- FP/Tag 1)	0,72	1,62

1) FP-gewichtet unterstellt.

Tab. 3: Ergebnisse der Projekte

Aus der Tabelle läßt sich ablesen:

- Die **Sprache NATURAL1** war **nicht produktiver als COBOL** (LOC/Std und FP/Tag für TOUR1 und PRAEM).
- Die besonders schlechte Produktivität in TOUR1 ist in der Problemkomplexität und Neuartigkeit der Lösung zu suchen und nicht in der Sprache (35% SPZ-Aufwand).
- **NATURAL1** ist entgegen den Behauptungen von Boehm und Jones **nicht kompakter** und allein dadurch wohl nicht produktiver **als COBOL.** Die Erklärung, warum dies in größeren Programmsystemen nicht so zu sein scheint, ist einfach: In NATURAL1 wie anderen 4GL fehlen Sprachkonstrukte zur Wiederverwendbarkeit (externe Unterprogramme, Makros / Copy-Strecken). Dadurch enthalten größere Systeme erhebliche Mengen an kopiertem Code, der den Quellcode aufbläht und schwer wartbar macht. Diese Mängel weist NATURAL2 nicht mehr auf.
- **NATURAL2** ist deutlich **kompakter** und damit ziemlich sicher auch produktiver **als NATURAL1 und COBOL.** Durch Reduzierung der zu wartenden LOC kann nach Boehm die Produktivität um 5-10% erhöht werden [8, p.47ff.].
- COBOL+CICS-commands als **Entwicklungsumgebung** (LGSTEL) erscheint produktiver als die übrigen "modernen" Umgebungen. Hier müssen Meßfehler durch das Änderungsprojekt

vermutet werden. Der subjektive Eindruck der Entwickler ist, daß die Umgebung NATURAL/370 erheblich produktiver ist als COBOL/CICS, wenn die Antwortzeiten in der Entwicklungsumgebung gut sind. Diese Frage kann erst durch Auswertung weiterer Projekte beantwortet werden, und zwar durch Ermittlung von **DM/FP** und **FP/Tag**.

- Die **neue Version** eines Systems ist auch bei Neuimplementierung erheblich **produktiver als die Erstentwicklung** (TOUR2 mit 546 DM/FP gegenüber TOUR1 mit 1.218 DM/FP).
- Die **Produktivität in LOC/Std** ist bei **COBOL** höher als der in der Literatur angegebene Wert von 2,5 bis 5 LOC/Std [8,p.56; 13,S.63f.]. Dies dürfte daran liegen, daß die publizierten Werte auf wesentlich größeren Projekten basieren, in denen die Produktivität pro Einheit sinkt [vgl. 13, S.100].

6.2 Empfehlungen

Die aus den Ergebnissen ableitbaren Schlußfolgerungen lassen sich zu Empfehlungen zusammenfassen:

- **Aufwandschätzungen** für administrative Software sollten **mit Funktionspunkten** durchgeführt werden, nachdem man sich durch Nachkalkulation eine Datenbasis für die Überprüfung der Schätzungen geschaffen hat. Hierzu dürften ca. 20 nicht zu kleine Projekte genügen.
- **Intuitive Analogieschlüsse** ohne Istdaten sind **zu gefährlich**. Nur nachkalkulierte Istdaten bieten die Möglichkeit, eine solche Schätzung zu überprüfen.
- Wegen der Vielzahl qualitativer Einflußfaktoren (vor allem Entwicklungsumgebung, Qualifikation und Projektumfang) wird es **nicht möglich** sein, die in einer Organisation gewonnene **Schätzbasis auf andere Organisationen zu übertragen**. Es dürfte jedoch deutlich geworden sein, daß jede Software herstellende Firma oder Behörde (Softwarehaus oder Anwender) sich die erforderliche Datenbasis mit relativ geringem Aufwand erzeugen kann.
- Für **Produktivitätsvergleiche** lassen sich sowohl **FP** als auch **LOC** gut verwenden.
- **Prozentanteile je Phase** sollten zur Beurteilung der **Projektorganisation** ermittelt werden.

Literatur

[1] Albrecht, A.J; Gaffney, J.E.: Software Function, Source Lines of Code and Development Effort Prediction, in: IEEE Trans. on SE 9(1983) No.6, pp.639-648.

[2] Basili, V.R.; Rombach, H.D.: The TAME Project: Towards Improvement-Oriented Software Environments, in: IEEE Trans. on SE 14(1988) No.6, pp.758-773.

[3] Bauman, P.; Richter, L.: Wie groß ist die Aussagekraft heutiger Software-Metriken? in: Wirtschaftsinformatik 34(1992) H.6, S.624-631.

[4] Behrens, C.A.: Measuring the Productivity of Computer Systems Development Activities with Function Points, in: IEEE Trans. on SE 9(1983) No.6, pp.648-651.

[5] Boehm, B.W.: The High Cost of Software, in: Horowitz, E. (ed.): Practical Strategies for Developing Large Scale Software Systems, Addison-Wesley, Reading /Mass. 1975.

[6] Boehm, B.W.: Software Engineering Economics, Prentice Hall, Englewood Cliffs/NJ 1981.

[7] Boehm, B.W.: Software Engineering Economics, in: IEEE Trans. on SE 10(1984) No.1, pp.4-21.

[8] Boehm, B.W.: Improving Software Productivity, in: Computer 20(1987) No.9, pp.43-57.

[9] Boehm, B.W.; Papaccio, P.N.: Understanding and Controlling Software Costs, in: IEEE Trans. on SE 14(1988) No.10, pp.1462-1477.

[10] Brooks, F.P.: The Mythical Man Month, Addison-Wesley, Reading/Mass. 1975.

[11] Jackson, M.A.: Principles of Program Design, Academic Press, London - New York - San Francisco 1975.

[12] Jones, C.: Program Quality and Programmer Productivity, TR 02.764, IBM Corp., San José/CA 1977.

[13] Jones, C.: Effektive Programmentwicklung - Grundlagen der Produktivitätsanalyse, McGraw Hill, Hamburg - New York et.al. 1987 (engl: Programming Productivity 1986).

[14] McCabe, T.J.: A Complexity Measure, in: IEEE Trans. on SE 2(1976) No.4, pp.308-320.

[15] Noth, Th.; Kretzschmar, M.: Aufwandschätzung von DV-Projekten, Springer, Berlin - Heidelberg - New York et.al. 1984.

[16] Orr, K.T.: Structured Systems

Development, Yourdon Inc., New York 1977.
[17] Spitta, Th.: Software Engineering und Prototyping, Springer, Berlin - Heidelberg - New York et.al. 1989.
[18] Wolverton, R.W.: The Cost of Developing Large-Scale Software, in: IEEE Trans. on Comp. C-24(1975) No.6, pp.615-635.