Jochen Ludewig[1]

Fredy Oertly[2]

Peter Dünki[3]

Claudio Galasso[3]

Eine empirische Untersuchung:

Der Stand des Software-Engineering im Raume Zürich

Im Rahmen einer Diplomarbeit am Institut für Informatik der ETH Zürich wurde eine Untersuchung über den Stand des Software-Engineering im Raum Zürich durchgeführt. Ziel dieser Untersuchung war eine Bestandesaufnahme der Vorgehensweisen, der verwendeten Methoden und der eingesetzten Werkzeuge und Programmiersprachen bei der Software-Entwicklung in der Praxis. Die Arbeit hat gezeigt, dass an den meisten Orten nach einem bestimmten Vorgehensmodell (Life cycle model) gearbeitet wird, dass aber noch viele Punkte zu verbessern sind, bis die Softwareherstellung als ausgereifte Ingenieurdisziplin betrachtet werden darf.

1. Aufgabenstellung und Durchführung

Aus den USA und Japan sind Untersuchungen bekannt, die sich mit der Praxis des Software-Engineering in grösseren Unternehmungen befassen (z.B. Zelkovitz et al., 1984). Da uns keine entsprechende Untersuchung in der Schweiz bekannt ist, wurde in der Gruppe «Software-Engineering» des Instituts für Informatik der ETH Zürich eine Diplomarbeit ausgegeben, die dies nachholen sollte (Leitung: Prof. Jochen Ludewig[1], betreuender Assistent: Fredy Oertly[2], Diplomanden Peter Dünki[3], Claudio Galasso[3]).

Da eine Diplomarbeit nach vier Monaten abzugeben ist, musste die Untersuchung von Anfang an geographisch eingeschränkt werden. Es lag auf der Hand, den Raum Zürich als Untersuchungsgebiet zu wählen, genauer die Agglomeration Zürich. Aufgabe der Diplomanden war es:

- Betriebe in diesem Gebiet so auszuwählen, dass ein möglichst repräsentativer Mix entstand
- Fragebögen zu entwerfen
- Kontakte aufzunehmen und die Befragung durchzuführen
- Die Ergebnisse zusammenzufassen und zu bewerten

Fredy Oertly

2. Untersuchung

Die «Agglomeration Zürich» ist weder ein politischer noch ein rechtlicher Begriff. Das Statistische Amt der Stadt Zürich führt jedoch eine Liste aller Gemeinden, die zur Agglomeration Zürich gezählt werden. Wir haben diese Liste um die Städte Baden, Uster und Winterthur erweitert.

Für dieses Untersuchungsgebiet haben wir versucht, eine möglichst vollständige, objektive und reproduzierbare Liste aller Unternehmungen zu erstellen, die Software entwickeln. Wir haben dabei als Minimalkriterium drei Mitarbeiter in der Software-Entwicklung und mindestens ein Projekt von mindestens einem Mannjahr oder mehr gefordert. Betriebe, die diese Anforderung erfüllen, finden sich einerseits unter den «Softwareherstellern» (Softwarehäuser, die Herstellung und Verkauf von Software als hauptsächliche Geschäftstätigkeit haben), anderseits unter den grossen «Anwendern» (Unternehmungen, die einen grossen Bedarf an Software haben, wo die Herstellung von Software aber nicht die hauptsächliche Geschäftstätigkeit ist).

Zum Auffinden der Softwarehersteller haben wir uns auf das «Schweizerische Ragionenbuch» (offizieller Firmenführer) gestützt, bei den Anwendern lieferte die Zusammenstellung «Die grössten Unternehmen der Schweiz» (Hrsg. «Schweiz. Handelszeitung») die benötigten Angaben. Aus einer Liste, die aus diesen Quellen zusammengetragen wurde und 430 Unternehmungen umfasste, wurden nach verschiedenen Kriterien sowie einer Zufallsreihenfolge 50 EDV-Abteilungen aus 44 Betrieben ausgewählt und untersucht.

Die Untersuchung wurde in zwei Phasen gegliedert: Der erste Teile wurde im Januar 1987 in Form persönlicher Inter-

[1] Professor Dr. Jochen Ludewig, Leitung, Institut für Informatik, Eidgenössische Technische Hochschule Zürich (ETHZ): Nachwort

[2] Dipl. math. ETH Fredy Oertly, betreuender Assistent, Institut für Informatik, ETHZ: Aufgabenstellung und Durchführung

[3] Peter Dünki, dipl. Informatik-Ing. ETH, und Claudio Galasso, dipl. Informatik-Ing. ETH, Diplomanden: Untersuchung, Ergebnisse, Auswertung

Die Diplomarbeit, die diesem Beitrag zugrunde liegt, ist als «Gelber Bericht» Nr. 80 des Instituts für Informatik erschienen (Bezug: Institut für Informatik, ETH-Zentrum, 8092 Zürich, Preis Fr. 10.–)

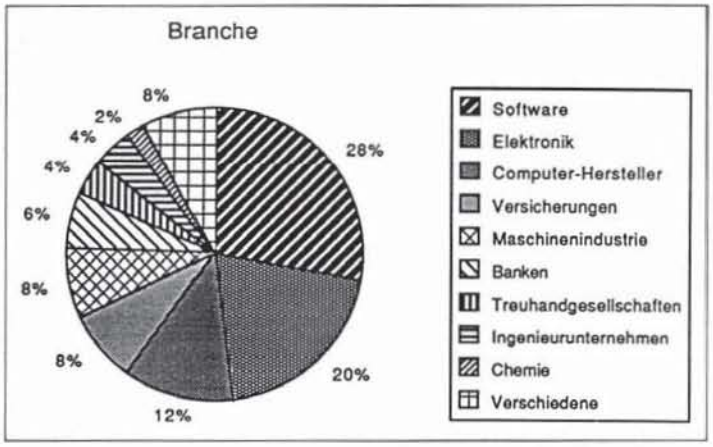

Bild 1 Die 50 besuchten EDV-Abteilungen stammen aus Unternehmungen der hier dargestellten Branchen

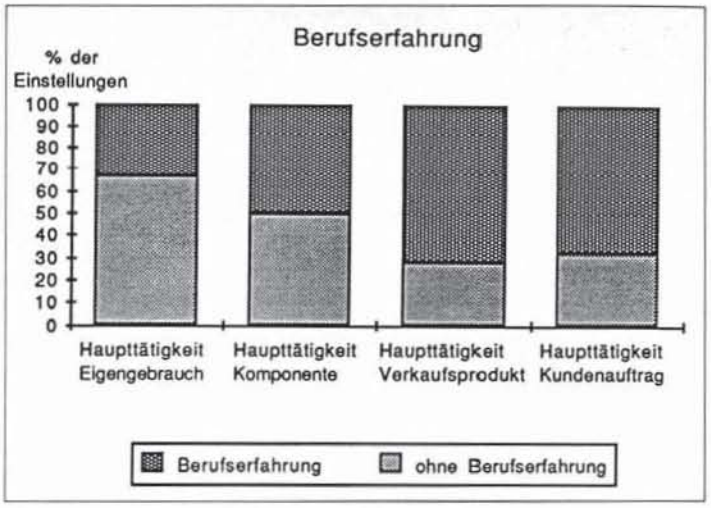

Bild 2 Anteile der im Jahre 1986 eingestellten Personen mit EDV-Berufserfahrung

views durchgeführt. Gesprächspartner waren jeweils die Leiter der besuchten EDV-Abteilungen sowie häufig auch Projektleiter. Der umfangreiche Fragebogen enthielt generelle Fragen zur Software-Erstellung sowie zum Umfeld der Software-Enwicklung. Zudem wurde in jeder Abteilung ein konkretes Projekt betrachtet, das bezüglich Art, Grösse und Vorgehensweise für die jeweilige Abteilung als typisch erachtet wurde.

Der zweite Teil der Umfrage wurde in Form eines Fragebogens gestaltet, der den Interviewpartnern des ersten Teils zugestellt wurde. Dieser zweite Teil behandelte Aspekte, denen im ersten Teil zu wenig Beachtung geschenkt worden war.

Die Auswertung der Umfrage ist in der erwähnten Diplomarbeit festgehalten. Nachstehend sind die wichtigsten Ergebnisse vorgestellt. Abschnitt 2 stellt einen Überblick der gesamten Arbeit dar. In Abschnitt 3 bis 6 werden einige besonders bedeutungsvolle Gesichtspunkte vertieft behandelt. Abschnitt 7 schliesslich enthält einige Vorschläge zur Verbesserung der bestehenden Praxis.

3. Untersuchungsergebnisse

3.1. Klassifizierung der Betriebe und Abteilungen

Die meisten besuchten Organisationseinheiten der Software-Entwicklung – wir nennen sie einheitlich *Abteilungen* – entwickeln vorwiegend Applikationssoftware im kommerziellen oder technischen Bereich. Werkzeuge und Systemsoftware stellen nur in wenigen Abteilungen einen bedeutenden Anteil des Entwicklungsaufwandes dar, während mehrere Abteilungen in geringem Ausmass Werkzeuge und Systemsoftware entwickeln. *Bild 1* zeigt, aus welchen Branchen die besuchten Unternehmungen stammen.

Die untersuchten Abteilungen entwikkeln hauptsächlich Software für eine einzige bestimmte Art von Kunden. Wir haben deshalb die Abteilungen in vier Haupttätigkeitsbereiche eingeteilt:

- 20 Abteilungen erstellen Software für betriebsinterne Erfordernisse (Eigengebrauch)
- 5 Abteilungen entwickeln Software, die als Komponente eines elektronischen Gerätes eingesetzt wird (Produktkomponente)
- 6 Abteilungen erstellen Software für ein breiteres Anwenderspektrum (Verkaufsprodukt)
- 19 Abteilungen entwickeln Software im Kundenauftrag (Kundenauftrag)

Auffällig ist, dass Abteilungen, die hauptsächlich für den betriebseigenen Gebrauch tätig sind, grossen Unternehmungen angehören und selber eine grössere Anzahl Mitarbeiter beschäftigen. Diese Tatsache ist leicht zu erklären: Grosse Unternehmungen benötigen für ihre Datenverarbeitungsaufgaben (Personal-, Finanz-, Materialverwaltung, Planung und Dienstleistungen aller Art) integrierte Programmsysteme, deren Erstellung und Wartung einen grossen Aufwand erfordern (grosse Anwender).

Die Kategorie *Produktkomponente* befindet sich ebenfalls oft in grösseren Unternehmungen. Die eigentlichen Produkte müssen ja auch entwickelt und produziert werden. Die Abteilungsgrösse variiert aber je nach gewählter Organisationsform stark: bei zentraler EDV sind die Abteilungen grösser. Besteht hingegen für jede Fachabteilung je eine Softwaregruppe, dann sind das oft nur wenige Mitarbeiter.

Die Bereiche *Kundenauftrag* und *Verkaufsprodukt* stammen aus der Kategorie der Softwarehersteller. Es handelt sich dabei um eher kleinere Unternehmungen, die Software entwickeln sowie Software und Hardware verkaufen. Dementsprechend sind auch die Software-Entwicklungsabteilungen im Vergleich zu denen der Grossbetriebe eher klein. Zu sagen wäre noch, dass es neben den besuchten Betrieben der Branche «Software» noch eine grosse An-

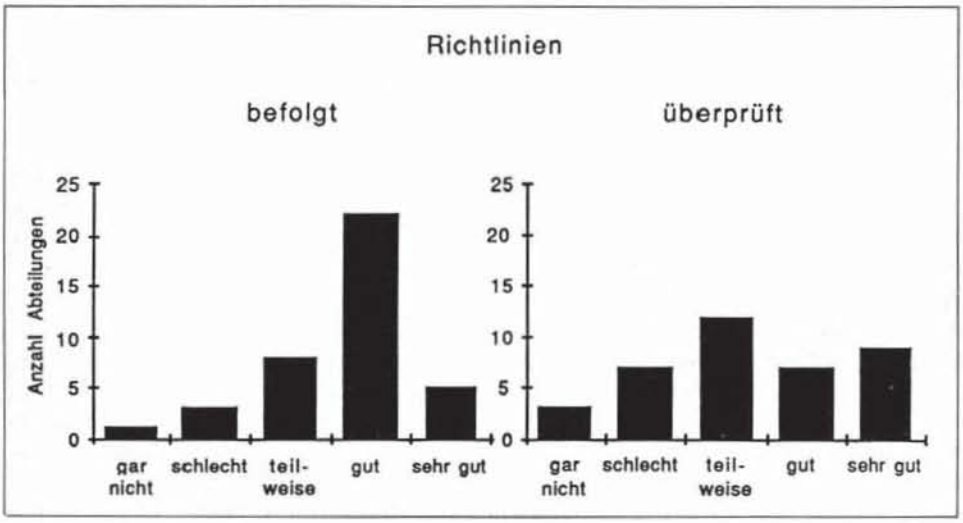

Bild 3 Bewertung der Befolgung und der Überprüfung der Richtlinien durch die Gesprächspartner

zahl von Klein- und Kleinstanbietern mit ein bis drei Mitarbeitern gibt, welche aber unsere Auswahlkriterien nicht erfüllen.

3.2. Personal in den EDV-Abteilungen

In 39 Abteilungen, bei denen die entsprechenden Zahlen vorlagen, waren rund 900 Personen mit der Software-Entwicklung beschäftigt. Im Jahr 1986 wurden davon rund 200 Mitarbeiter neu eingestellt. Es fällt auf, dass Abteilungen, die im Bereich der Produktkomponenten tätig sind, technisch ausgebildete Personen mit ETH- oder HTL-Abschluss bevorzugen. Abteilungen, die kommerzielle Software entwickeln, bevorzugen hingegen Personen, die bereits eine gewisse Erfahrung im Anwendungsgebiet besitzen. Die Hälfte der neueingestellten Personen verfügte über keine EDV-Berufserfahrung. Besonders hoch ist dieser Anteil (70%) bei den Abteilungen des Bereichs Eigengebrauch *(Bild 2)*.

Informatikkenntnisse werden den Mitarbeitern im Operating, in internen oder externen Kursen, vermittelt. Die Einschulung dauert durchschnittlich 40 Tage, wobei Anfängerkurse 3 bis 6 Monate oder 1 Jahr dauern. Für Weiterbildung wurden im Jahre 1986 durchschnittlich 11 Arbeitstage aufgewendet, sie leidet allgemein unter Termindruck.

3.3. Entwicklungscomputer und Datenbanksysteme

Durchschnittlich entwickeln die besuchten Abteilungen ihre Software zu 55% auf Grossrechnern, zu 35% auf Mini/Midi und zu 10% auf Personal-Computern.

Zwei Drittel aller Abteilungen arbeiten nur auf einer der oben erwähnten Computerkategorien. Jene Abteilungen, die über mehrere Typen verfügen, bevorzugen oft einen davon sehr stark. Für einzelne losgelöste Anwendungen werden auch andere Systeme verwendet.

Wir haben in jeder Abteilung auch ein konkretes Projekt betrachtet, dabei wurde in 29 von 50 Abteilungen ein Datenbanksystem eingesetzt. IMS nimmt mit 13 Anwendungen zwar immer noch eine Vorrangstellung ein, doch werden in verschiedenen Abteilungen in neueren Projekten modernere (relationale) Datenbanksysteme wie DB2 eingesetzt.

3.4. Richtlinien für das Erstellen von Software

In 39 der 50 untersuchten Abteilungen gibt es Richtlinien zum Erstellen von Software. Die Unterschiede bezüglich Art und Umfang sind gross. Die Richtlinien umfassen in den allermeisten Fällen die Projektorganisation und die Projektabwicklung. Es werden in der Regel die Verantwortlichkeiten und ein Phasenablauf festgelegt. Das Vorgehen innerhalb der einzelnen Phasen ist weniger häufig vorgeschrieben.

Die Methoden und Werkzeuge für die Software-Erstellung sind nur in seltenen Fällen in den Richtlinien behandelt. Die Richtlinien werden in 14 Abteilungen durch Abgabe des entsprechenden Ordners und in 22 Abteilungen durch spezielle Kurse vermittelt. Das «Training on the job» trägt jedoch wohl zum Erlernen der Richtlinien den grössten Teil bei. Die Einhaltung der Richtlinien wird in vielen Abteilungen mit *gut* bewertet, doch werden sie häufig nur teilweise überprüft *(Bild 3)*.

Die Abteilungen, die keine derartigen Richtlinien besitzen, kennen meistens einen De-facto-Standard, an den sich die Betroffenen halten. Dieser wird bei kleineren Abteilungen oft durch eine einzelne Person bestimmt. De-facto-Standards sind nur dort anzutreffen, wo sich nur wenige Personen mit der Entwicklung von Software beschäftigen oder wo kleine Projektteams in sich abgeschlossene Projekte durchführen.

3.5. Vorgehen bei der Entwicklung

In den allermeisten Fällen wurde die Entwicklung der untersuchten Projekte in Phasen gegliedert, wobei die Schritte häufig nicht strikt getrennt waren. Wir haben wohl alle möglichen Varianten angetroffen, vom strikten Ablauf nach Phasenplan mit formalem Abschluss und Überprüfung jeder Phase mit Reviews, bis hin zum «Trial and error»-Vorgehen mit eher schwach ausgeprägten Phasen.

In elf Fällen wurde ein eigentliches Prototyping durchgeführt. Der Prototyp beanspruchte jeweils einen kleinen Teil des Gesamtaufwandes für das Projekt und diente in allen Fällen zur Abklärung und Optimierung der Benützerschnittstelle. Die Prototypen wurden in etwa gleich vielen Fällen nach Gebrauch weggeworfen oder als Grundlage für das zu entwickelnde System gebraucht.

3.6. Einbezug des Kunden und Verrechnung der Entwicklungskosten

Der Kunde ist in der Regel an der Pflichtenhefterstellung, in der Entwurfsphase, wenn es um die Klärung der Benützerschnittstellen geht, und während des Systemtests am Projekt beteiligt. Der Kunde ist ebenfalls oft in einem Projektausschuss vertreten, aber nur selten nimmt er an der Projektleitung teil.

Die Verrechnung der Kosten ist oft von Projekt zu Projekt verschieden. In 30% der Abteilungen existiert keine direkte Verrechnung der Kosten; in der Hälfte der Abteilungen wird nach Aufwand

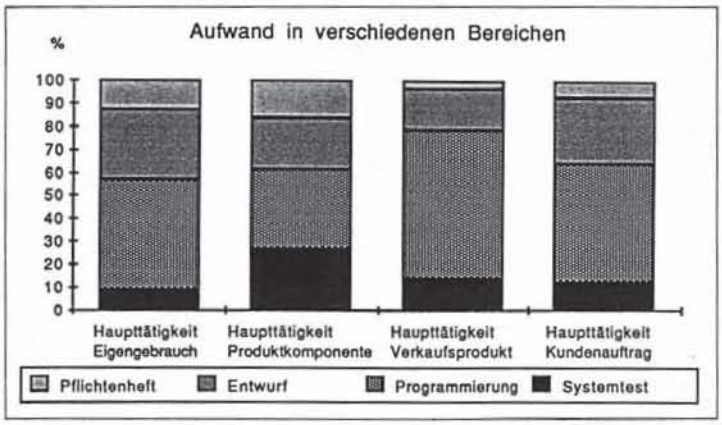

Bild 4 Durchschnittliche prozentuale Verteilung der Aufwände für die einzelnen Phasen der Entwicklung in den betrachteten Projekten

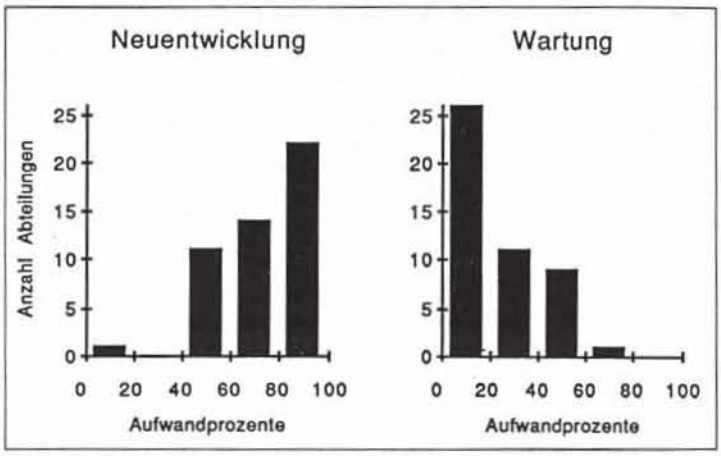

Bild 5 Verteilung der Aufwände für Neuentwicklung und Wartung

abgerechnet. In einem Drittel der Abteilungen wird die Vorstudie nach Aufwand und die Realisierung mit Festpreis verrechnet, und in 20% der Abteilungen wird ein Festpreis für das gesamte Projekt vereinbart.

3.7. Vorgehen im konkreten Projekt

Wie bereits festgestellt, ist eine Gliederung in Phasen weit verbreitet. Wir haben darum die Untersuchung von Vorgehen, Methoden und Werkzeugen ebenfalls nach Phasen gegliedert, und zwar unterscheiden wir vier Hauptphasen: Erstellen des Pflichtenheftes, Entwurf, Programmierung und Systemtest.

Bild 4 zeigt für jede Abteilungsklasse die durchschnittliche prozentuale Verteilung des Aufwandes jeder Phase in den konkreten Projekten. Auffallend klein ist der Systemtestanteil im Bereich Eigengebrauch, dafür ist dort der Wartungsanteil höher als in anderen Bereichen.

Hohe Programmieranteile sind bei Abteilungen zu erkennen, die sich nicht strikt an ein Lebensphasenmodell halten. Kleine Programmieranteile hingegen sind dort gegeben, wo eine klare Trennung zwischen Analyse und Programmierung, also Analytikern und Programmierern besteht. Dort werden die Programme oft bis zur Programmvorgabe entworfen. Es gibt jedoch auch einige Abteilungen, die eine solche Unterscheidung zwar machen, deren Programmieranteile aber dennoch nicht klein sind.

Beim Komponententest wird die Software oft mit synthetischen Daten getestet, um Spezialfälle abzudecken. Von den 29 Abteilungen, die ein Datenbanksystem einsetzen, verwenden 18 beim Systemtest echte Daten. Unter *echten Daten* sind Informationen zu verstehen, welche im produktiven Betrieb anfallen, wenn vielleicht auch in einer anderen Form. Keine echten Daten können dort verwendet werden, wo es diese noch gar nicht gibt. Mancherorts liefert der Kunde die Testdaten oder testet die entwickelte Software auf einem eigenen System.

Der Gesamtaufwand der Projekte liegt zwischen 0,5 und 800 Mannjahren, wobei oft nur Teilprojekte betrachtet werden. Der Aufwand für die meisten Projekte liegt zwischen 1 und 16 Mannjahren.

3.8. Wartung

Die erwartete Lebensdauer der konkreten Projekte beträgt durchschnittlich neun Jahre. Den Begriff «Wartung» interpretieren wir restriktiv: Unter Wartung verstehen wir lediglich die Fehlerbehebung an bestehenden Programmen und Anpassungen derselben aufgrund von Änderungen in der Systemumgebung (Release-Wechsel bei Compiler, Betriebssystem, Datenbanksystem). Wir schliessen jede funktionale Erweiterung eines Programms aus.

In den 50 untersuchten Projekten werden die Wartungsarbeiten nach Abschluss der Entwicklung in 38 Fällen vom Projektteam und in 6 Fällen von einem Unterhaltsteam des Entwicklers übernommen. In den anderen 6 Fällen übernimmt der Kunde die Wartung, oder Wartungsaufträge werden als neue Projekte aufgefasst. Der von den Software-Entwicklern geleistete Aufwand für Wartung liegt bei der Hälfte der Abteilungen zwischen 1 und 20% *(Bild 5)*.

In zehn Fällen beansprucht die Wartung mehr als 40% des Aufwandes; es handelt sich dabei oft um grosse Systeme, die nicht ohne weiteres ersetzt werden sollen.

Gesamthaft liegt das Aufwandverhältnis von Neuentwicklung zu Wartung ungefähr bei drei zu eins. Die Tatsache, dass im Bereich der Eigenentwicklung die Entwickler im eigenen Haus zur Verfügung stehen, führt in vielen Fällen zu häufigeren Anpassungen und Erweiterungen (durchschnittlich 30% Wartungsaufwand).

Generell ist zu sagen, dass an vielen Orten keine genauen Angaben über den Wartungsaufwand festgehalten werden und dass Wartung im Sinne von Fehlerkorrekturen häufig anlässlich von funktionellen Erweiterungen von Programmen durchgeführt wird.

3.9. Probleme bei der Software-Entwicklung

Als wichtigstes Problem wurde die Personalbeschaffung bezeichnet. Lediglich in Abteilungen, die Personallimiten kennen, ist diese Schwierigkeit vordergründig nicht vorhanden. Durch den Mangel an Mitarbeitern, insbesondere an qualifizierten Mitarbeitern, steigt der Termindruck angesichts der zu bewältigenden Aufgaben und stellt somit ebenfalls ein wichtiges Problem dar. Oft genannt wurden Schwierigkeiten mit der Planung (Abschätzung des Aufwandes) sowie der Mangel an Methoden und Werkzeugen für die Entwicklung.

Die Vergangenheitsbewältigung und die Parallelität von Neuentwicklung und Wartung wird dort als Problem taxiert, wo eine grosse Software-Umgebung existiert oder verschiedene Versionen auf dem Markt sind, die zu unterhalten sind. Gesamthaft gesehen bilden finanzielle Aspekte keine grossen Probleme (Preise für vorhandene Werkzeuge, Kosten für die Infrastruktur, Scheu vor Investitionen). Das Verhältnis zum Kunden bereitet vielerorts grosse Sorgen. Der Kunde könne seine Anforderungen nicht EDV-gerecht darstellen oder ändere seine Wünsche während der Entwicklung. Die Software-Entwickler ihrerseits glauben hingegen, das jeweilige Anwendungsgebiet gut zu kennen.

4. Programmiersprachen

Wir haben unsere Interviewpartner gefragt, welche Programmiersprachen in ihrer Abteilung eingesetzt werden. Uns interessierte aber nicht nur die Verwen-

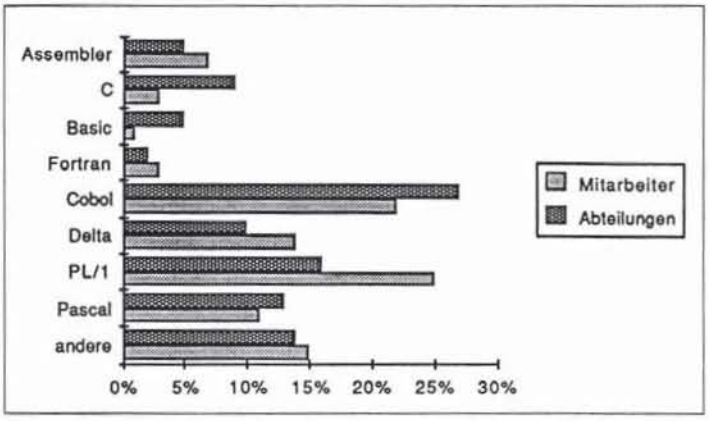

Bild 6 Gewichtete (helle Balken) und ungewichtete (dunkle Balken) durchschnittliche Prozentanteile der wichtigsten Programmiersprachen

dung einzelner Sprachen, wie dies in anderen Umfragen untersucht wurde, sondern wir wollten wissen, wie gross die Prozentanteile der einzelnen Programmiersprachen an der gesamten Software-Entwicklung sind.

Wir haben zwei Fragestellungen untersucht: den durchschnittlichen Prozentanteil der wichtigsten Sprachen in den Abteilungen und die mit der Zahl Mitarbeiter einer Abteilung gewichteten Prozentsätze.

Die dunklen Balken in *Bild 6* stellen den durchschnittlichen Prozentanteil der wichtigsten Programmiersprachen am gesamten Software-Entwicklungsaufwand dar. Die Prozentanteile der eingesetzten Programmiersprachen wurden mit der Anzahl Mitarbeiter dieser Abteilung multipliziert. Die hellen Balken stellen somit die gewichtete Verteilung der Programmiersprachen dar und zeigen in etwa, wie viele Personen mit einer Sprache arbeiten.

Programmiersprachen, deren gewichteter Anteil grösser ist als der ungewichtete, werden somit vornehmlich in grösseren Abteilungen eingesetzt («Delta», PL1 und Assembler). Hingegen zeigt die umgekehrte Tendenz auf, welche Sprachen mehrheitlich in Abteilungen mit wenigen Mitarbeitern zur Anwendung kommen (Cobol, Basic und C). Die Tatsache, dass auch Cobol zu diesen Sprachen gehört, erklärt sich dadurch, dass nebst einigen grösseren Abteilungen auch in mehreren kleineren Betrieben in Cobol programmiert wird. Diese arbeiten häufig im Auftrag von Kunden, die Cobol vorschreiben. Pascal, Fortran und «andere» zeigen keine wesentlichen Unterschiede.

Der ungewichtete Prozentanteil von Cobol steigt von 27% auf 35% an, wenn man den von Delta erzeugten Cobol-Code dazuzählt, und der gewichtete Anteil steigt von 22% auf 33%. Bei PL1 ergeben sich nur geringfügige Änderungen.

In der Klasse «andere» sind hauseigene Programmiersprachen, RPG, Natural, Modula 2 und verschiedene Sprachen der 4. Generation enthalten. Die Sprachen der vierten Generation (ohne «Delta») machen mit insgesamt 1,5% noch einen kleinen Anteil aus.

In der Regel dominiert in jeder Abteilung eine Programmiersprache ganz deutlich. Nur für spezielle Anwendungen, Pilotprojekte oder zur Wartung bestehender Software gelangen eine zweite oder eine dritte Sprache zur Anwendung.

Abteilungen mit der Haupttätigkeit Eigengebrauch setzen, meist historisch bedingt und mit Rücksicht auf die bereits vorhandene grosse Systemumgebung, Assembler, PL1, Cobol und neuerdings Cobol- und PL1-Code erzeugendes Delta ein.

Abteilungen der Haupttätigkeit Produktkomponenten sind eher frei bei der Wahl der Programmiersprache, sie haben keine vorhandene Umgebung, der man sich anzupassen hat. Innovationen sind darum eher möglich. Deshalb findet man hier moderne Sprachen wie Pascal und C.

Abteilungen der Haupttätigkeit Verkaufsprodukt sind ebenfalls oft frei in der Wahl der Sprache, sie richten sich aber häufig nach bestehenden Marktstandards.

Abteilungen der Haupttätigkeit Kundenauftrag können zur Hälfte selber entscheiden, welche Sprache sie einsetzen wollen, und in der anderen Hälfte gibt dies der Kunde (wegen seiner Systemumgebung) vor. PL1, Cobol und Pascal sind hier die meistgenannten Sprachen.

Assembler wird in 13 Abteilungen als eingesetzte Programmiersprache genannt. Aber in elf dieser Abteilungen bleibt der Gesamtentwicklungsanteil unter 20%. Assembler wird in Applikationen für zeitkritische und hardwarenahe Teilaufgaben angewendet. Beträchtliche Assembleraufwände sind dort zu beobachten, wo grosse bestehende Systeme, die in Assembler geschrieben wurden, gewartet und erweitert werden.

Die *Sprache C* dient als Alternative zum Assembler für zeitkritische und system-

nahe Aufgaben. Als Hauptgrund für die Verwendung von C wurden Portabilität und Performance genannt. C wird im Zusammenhang mit Unix eingesetzt. Im Gegensatz zu Assembler werden jedoch ganze Applikationen in C geschrieben.

Cobol, PL1 und *Delta* sind die bevorzugten Sprachen für kommerzielle Applikationen und sind deshalb stark verbreitet. Häufig sind bestehende Systemumgebungen vorhanden, so dass es in diesen Fällen praktisch undenkbar ist, eine andere Sprache einzusetzen. Durchgesetzt hat sich einzig Delta, eine Sprache der vierten Generation, welche auf Cobol oder PL1 aufbaut. Es handelt sich dabei um ein Werkzeug, das die bestehende Umgebung ergänzt und erweitert, aber nicht ersetzt. Die Schnittstellen zur bestehenden Umgebung bleiben damit erhalten.

Fortran wurde in den untersuchten Abteilungen vorwiegend zur Wartung und Weiterentwicklung bestehender Applikationen eingesetzt.

Die Wahl der Programmiersprache ist für ein Projektteam oft kein Thema, weil sie vorgegeben ist. Neben der Anlehnung an die bestehende Systemumgebung sind oft Firmenphilosophie (historischer Entscheid), die vorgegebene Entwicklungsumgebung (vorhandener Compiler usw.), das Know-how, die Ausbildung der Mitarbeiter und die Anlehnung an gewisse Marktstandards (als Supportgarantie) von grosser Bedeutung.

Der betrachtete Fortran-Anteil ist relativ klein. Wenn wir vermehrt wissenschaftliche Anwendungen berücksichtigt hätten, dann wäre der Anteil wahrscheinlich um einiges höher. Wir vermuten, dass moderne Programmiersprachen, wie Pascal, nur dort eingesetzt werden, wo vollständige Neuentwicklungen angegangen werden oder wo bestehende Software vollständig neu geschrieben wird. Wir glauben, dass im technisch-wissenschaftlichen Bereich in den letzten Jahren Pascal auf Kosten von Fortran Anteile gewonnen hat.

Nur wenige der besuchten Abteilungen und Projektteams konnten ihre Programmiersprache frei wählen. Ihre Projekte waren oft Insellösungen oder stammten aus dem Bereich der Produktkomponenten.

Oft besteht der Wunsch nach dem Einsatz höherer Programmiersprachen. Dort, wo allerdings ganze Softwarepakete umgeschrieben werden müssten, kommt dies aus finanziellen Gründen nicht in Frage. Die Wahl der Programmiersprache ist, vor allem in Unternehmungen mit grosser Softwareproduktion, ein strategischer Entscheid; dadurch wird der Einsatz moderner Hochsprachen, wie Pascal, Modula 2, Ada, und Sprachen der vierten Generation stark verzögert.

Kommerzielle Applikationen sind in unserem Gebiet wohl die verbreitetsten Anwendungen von Computerlösungen, so dass der Cobol-, PL1- und Delta-Anteil bei den Programmiersprachen wohl auch in Zukunft etwa gleich hoch bleiben wird, auch wenn eine Tendenz zu vermehrtem Einsatz von Sprachen der vierten Generation zu erkennen ist.

5. Methoden und Werkzeuge

Bild 7 zeigt die Entwurfsmethoden, die in den 37 Abteilungen eingesetzt werden, die die entsprechenden Fragen im Fragebogen beantwortet haben.

Andere Entwurfsmethoden (Entscheidungstabellen, Petri-Netze, De Marco, Mascot und N^2-Charts) werden nur in einer oder zwei Abteilungen eingesetzt.

Die Häufigkeiten des Einsatzes einer Methode zeigt die Reihenfolge in *Bild 7.*

Die Tauglichkeit der Methoden Yourdon, Hipo/Eva und Jackson wurden häufig mit gut bewertet. Werkzeuge für die Unterstützung der Methode gibt es auf dem Markt fast immer, doch sind diese nicht in allen Abteilungen vorhanden oder auf dem jeweiligen Entwicklungssystem nicht erhältlich:

- In 8 Abteilungen kann jeder Analytiker die Methode selber wählen
- In 20 Abteilungen wird innerhalb eines Projektes eine einheitliche Methode angewendet
- In 9 Abteilungen wird eine Methode einheitlich in der gesamten Abteilung verwendet

In rund 40% der Abteilungen gibt es eine Stelle (Person oder Gruppe von Personen), welche den Einsatz von Methoden koordiniert und neue Methoden evaluiert und einführt.

15 von 40 Abteilungen verfügen über ein «integriertes Entwicklungssystem» (8 Delta, 1 Opus, 1 Epos und 5 Eigenentwicklungen). Je nach Wahl des Entwicklungssystems und der eingesetzten Programmiersprachen arbeiten die Abteilungen auch mit Werkzeugen, die durchwegs als taugliche Hilfsmittel bei der Software-Entwicklung gewertet wurden. Auffallend ist, dass viele der eingesetzten Maskengeneratoren Eigenentwicklungen sind.

In 80% der Abteilungen gibt es eine Stelle (Person oder Gruppe von Personen), welche den Einsatz von Werkzeugen überwacht und koordiniert und neue Werkzeuge evaluiert und einführt. Meistens wird der Kauf von Werkzeugen über das ordentliche Gesamtbudget der Abteilung abgewickelt. Nur gele-

Methode	Einsatz unterstützt durch regelmässig	sporadisch z.Teil	nein	beurteilte Tauglichkeit gut	mittel	schlecht	Werkzeug	ja
Jackson	10	7	65 %	29 %	6 %	20 %	40 %	40 %
Pseudo-Code	7	9	56 %	44 %	0 %	27 %	20 %	53 %
Flussdiagramme	3	9	42 %	33 %	25 %	8 %	0 %	92 %
Nassi-Shneiderman	2	5	44 %	29 %	29 %	29 %	29 %	44 %
HIPO / EVA	4	2	83 %	17 %	0 %	50 %	0 %	50 %
Präzedenzdiagramme	2	4	50 %	50 %	0 %	0 %	17 %	83 %
Zustandsdiagramme	2	3	60 %	40 %	0 %	20 %	20 %	60 %
Yourdon	3	1	100 %	0 %	0 %	50 %	0 %	50 %

Bild 7 In den 37 Abteilungen, die die entsprechenden Fragen beantwortet haben, werden die dargestellten Methoden eingesetzt

gentlich besteht ein separates Budget für Werkzeuge, und gelegentlich wird der Kauf auf ein laufendes Projekt abgewälzt.

Die Einführung von Werkzeugen wird nach den Antworten im Fragebogen durch folgende Punkte erschwert:

- Zeitmangel für Schulung und Einführung
- Mangelnde Akzeptanz bei neuen Methoden und Werkzeugen
- Erhältliche Werkzeuge decken nur einen Teil der individuellen Erfordernisse ab
- Werkzeuge sind nicht für alle Entwicklungsumgebungen erhältlich
- Werkzeuge sind zu teuer

6. Qualitätssicherung

Nach Ludewig (1985) umfasst die Qualitätssicherung im wesentlichen zwei Aktivitäten:

- Sicherstellen, dass gewisse Prinzipien des Softwaremanagements auch in der Praxis eingehalten werden (vor allem Aufstellung der wichtigsten Pläne)
- Prüfen der Ergebnisse aller Phasen der Software-Erstellung. Neben dem Test kommen hier vor allem informale Verfahren in Frage, zum Beispiel Code inspection, Walkthrough. Für alle derartigen Prüfungen gilt, dass ein nützlicher Effekt nur zu erzielen ist, wenn der Programmierer kooperiert, sich also nicht angegriffen, sondern gefördert fühlt

Bei Zehnder (1986) ist die folgende Liste von Software-Qualitätskriterien zu finden: Vollständigkeit der Funktionen, Genauigkeit, Zuverlässigkeit, Fehlertoleranz, Benützerfreundlichkeit, Flexibilität in der Verwendung, Modularität, Einfachheit, Ausbaufähigkeit, Kompatibilität, Übertragbarkeit, Unterhaltsfreundlichkeit, Entwicklungseffizienz, Betriebseffizienz, Datenintegrität, Datenkonsistenz.

Wir haben versucht festzustellen, ob sich in den untersuchten Abteilungen eine Person oder eine Stelle explizit mit Qualitätssicherung beschäftigt und ob in den betrachteten Projekten die Qualitätssicherung ausdrücklich als Managementfunktion wahrgenommen wird.

In 13 der 50 untersuchten Abteilungen war eine spezielle Qualitätssicherung anzutreffen. Verantwortlich für die Durchführung der damit verbundenen Massnahmen waren: die Stabsstelle QS, eine Abteilung oder ein Verantwortlicher für Methoden und Werkzeuge, der Kunde, ein Review-Team, ein spezielles Gremium, der Projektleiter oder dessen Vorgesetzter. In den anderen 37 Abteilungen wird die Qualitätssicherung implizit durchgeführt.

Diese Liste von Kriterien ist bekannt, ihre Überprüfung ist jedoch nicht systematisiert und institutionalisiert. Sie erfolgt mehr oder weniger bewusst durch den Projektleiter und die am Projekt beteiligten Personen.

Das oben erwähnte Sicherstellen gewisser Prinzipien des Software-Engineering, also das Überprüfen der formalen Aspekte der Software-Entwicklung, wird nur in wenigen Fällen gründlich durchgeführt. In einigen der untersuchten Projekte gibt es Projektausschüsse oder ähnliche Gremien, die Organisations-, Koordinations- und Kontrollaufgaben übernehmen.

Eine inhaltliche Überprüfung der entwickelten Software, das heisst Kontrolle, dass die obigen Qualitätskriterien eingehalten werden, war hingegen vermehrt anzutreffen. Folgende Massnahmen gehören in diesen Bereich:

a) In nahezu allen Fällen werden die vorhandenen Pflichtenhefte oder Grobkonzepte dem Kunden, der Fachabteilung, dem Vorgesetzten oder einer entsprechenden Instanz (z.B. einem Pro-

jektausschuss) zur Diskussion und Genehmigung vorgelegt.
b) Drei Viertel aller Entwürfe werden wiederum von derselben Instanz eingesehen und genehmigt. Dies betrifft jedoch häufig nur Dokumente, die konzeptionelle Aspekte beschreiben. Entwürfe für die technische Realisierung hingegen werden nur in seltenen Fällen von projektunabhängigen Stellen kontrolliert.
c) In einem Drittel der betrachteten Projekte werden die geschriebenen Programme in einem Walkthrough überprüft. Es handelt sich in den meisten Fällen jedoch nicht um ein institutionalisiertes Vorgehen, sondern um eine Massnahme, die nur stichprobenweise oder in Ausnahmesituationen durchgeführt wird.
d) Programmierung und Komponententest werden nur selten von verschiedenen Personen durchgeführt. In der Regel testet der Programmierer seinen Code selber. Beim Systemtest hingegen werden vermehrt der Kunde, der Benützer oder die Fachabteilung einbezogen.

Bewertung
Wir haben den Eindruck, dass die Existenz einer Stelle für Qualitätssicherung noch keine Gewähr für eine qualitativ hochwertige Software bietet. Andererseits kennen einige Abteilungen zwar keine explizite Stelle für Qualitätssicherung, beschäftigen sich jedoch stark mit diesem Thema. Allgemein kann gesagt werden, dass die Qualitätssicherung stärker beachtet werden sollte. Termingründe, mangelnde Kapazitäten und psychologische Aspekte verhindern eine verstärkte Qualitätssicherung. Problematisch erscheinen uns vor allem die psychologischen Gesichtspunkte, sind doch viele Personen nicht fähig oder nicht willig, über die eigene Arbeit kritisch nachzudenken und nehmen Kritik an der Arbeit persönlich.

7. Projekt-Nachkontrolle

Die Projekt-Nachkontrolle einer realisierten Applikation umfasst gemäss Zehnder (1986) das Überprüfen der folgenden Punkte nach einer gewissen Einlaufzeit von 6 bis 12 Monaten.
- Überprüfen der effektiven Leistungszahlen im Betrieb mit den entsprechenden Schätzungen, Annahmen und Forderungen in Pflichtenheft und Konzept
- Aufdecken von Schwachstellen in der realisierten Lösung und in der Entwicklung der Applikation

Wir haben versucht festzustellen, ob in den betrachteten Projekten eine Nachkontrolle stattgefunden hat. Wir wollten von unseren Gesprächspartnern wissen, ob während der Realisierung des Projektes Daten über Verlauf, Schwierigkeiten, Probleme usw. gesammelt und ausgewertet wurden und ob somit eine kritische Auseinandersetzung mit der geleisteten Arbeit stattgefunden hat.
In 39 der 50 besuchten Abteilungen werden Tages-, Wochen- oder Monatsrapporte gesammelt und ausgewertet. In der Regel handelt es sich dabei aber lediglich um eine Zeiterfassung für die Lohn- oder Aufwandberechnung. Weniger häufig haben wir eine ausführliche Berichterstattung über die durchgeführten Tätigkeiten angetroffen. Der damit verbundene administrative Aufwand wurde häufig als zu hoch erachtet.
In rund der Hälfte der Abteilungen werden während der Entwicklung aufgetretene Probleme, Fehler und Schwierigkeiten rapportiert oder einfach gesammelt. Häufig werden diese jedoch lediglich in eine Pendenzenliste eingetragen, bis sie behoben sind. Nur in ganz wenigen Fällen wird die Behebung der Fehler während des Systemtests überprüft oder werden die gesammelten Daten zum Zweck einer bewussten Vorgehenskontrolle ausgewertet.

Interpretation
Es gibt mehrere Gründe für die teilweise Vernachlässigung der Nachkontrolle:
a) Das Sammeln aller nötigen Daten für eine effiziente Vorgehenskontrolle ist mit einem gewissen administrativen Aufwand verbunden. Termingründe und Personalmangel zwingen die Entwickler oft zu einer kurzfristigen Denkweise, so dass alle nicht direkt produktiven Tätigkeiten vernachlässigt werden. Zu diesen zählen zum Beispiel die Dokumentation, die Qualitätssicherung und auch die Sammlung von Rapporten.
b) Die grosse Anzahl anstehender Aufgaben führt oft dazu, dass sich Mitarbeiter eines Teams noch vor Abschluss des Projekts in andere Aufgaben einarbeiten müssen und dass sie bei Abschluss des Projekts bereits anderweitig engagiert sind. Man nimmt sich also kaum die Zeit, gründlich und kritisch über die eigene Arbeit nachzudenken.

8. Bewertung der Ergebnisse

In diesem Abschnitt möchten wir drei – wie wir meinen, sehr wichtige – Punkte aufgreifen und besprechen. Diese Themenbereiche werden zwar nicht in allen,

aber in vielen der untersuchten Abteilungen zu wenig beachtet. Unsere Vorschläge können hierbei einen Denkanstoss liefern und somit zur Lösung oder zumindest zur Verminderung gewisser Probleme beitragen.

8.1. Beteiligung des Kunden an der Projektleitung

Wir haben das Verhältnis zwischen Software-Entwickler und Kunde untersucht und dabei folgende Tatsachen festgestellt:

a) Der Kunde ist stark beteiligt am Erstellen des Pflichtenhefts (70%) und des Entwurfs (65%) und macht häufig beim Systemtest mit (75%). Nur selten macht der Kunde bei der Codierung mit (8%) oder ist an der Projektleitung beteiligt (18%). Im Bereich des Eigengebrauchs ist die Beteiligung der Projektleitung (33%) höher als in Abteilungen, die im Kundenauftrag arbeiten (12%).

d) In den konkreten Projekten, die wir in den Interviews untersucht haben, war nur gelegentlich ein Kundenvertreter an der Projektleitung beteiligt.

c) Das Problem, dass der Kunde Mühe hat, seine Anforderungen und Wünsche zu äussern, macht in 46% der Abteilungen gewisse Sorgen, und in 27% der Fälle wird es als schwerwiegendes Problem taxiert. Besonders ausgeprägt scheint das Problem im Bereich des Eigengebrauchs zu sein (je 40%), weniger betont wird es in Abteilungen, die im Kundenauftrag arbeiten (53% bzw. 7%).

d) Ähnlich sehen die Resultate beim Problem der Änderung der Wünsche und Zielsetzungen des Kunden während der Entwicklung aus. In 35% der Fälle macht das Problem gewisse und in gar 41% der Abteilungen grosse Sorgen. Ebenfalls ein deutlicher Unterschied ist zwischen den Bereichen Eigengebrauch (26% bzw. 53%) und Kundenauftrag (46% bzw. 15%) festzustellen.

Es besteht demnach ein bedeutendes Kommunikationsproblem zwischen Entwickler und Anwender. Gemäss Zehnder (1986) hat die Projektleitung zwei wesentlich verschiedene Aufgabenbereiche:

a) Projektintern ist die professionelle Durchführung der Informatik-Entwicklungsarbeiten sicherzustellen.

Ziel: Informatisch saubere, gut realisierbare Anwendung.

b) Projektextern muss gewährleistet werden, dass die Zusammenarbeit mit dem künftigen Benützer klappt.

Ziel: Problemgerechte Informatikanwendung

Unsere Erfahrungen zeigen deutlich, dass dem zweiten Aspekt der Projektleitung häufig zu wenig Beachtung geschenkt wird; demnach könnte die von Zehnder (1986) beschriebene Doppel-Projektleitung wesentlich dazu beitragen, dieses Problem zu entschärfen. Zwei Projektleiter, ein Informatik- und ein Anwender-Projektleiter, werden innerhalb der gleichen Projektleitung eingesetzt. Die Verantwortung für das Projekt tragen beide gemeinsam. Das Schwergewicht der eigentlichen Projektführung verschiebt sich allerdings im Laufe der Projektdauer mehrfach, am Anfang (Pflichtenheft, Grobentwurf) und am Ende (Einführung) liegt sie eher beim Anwender-Projektleiter, dazwischen beim Informatik-Projektleiter.

8.2. Normierung der Vorgehensweise

In den untersuchten Projekten haben wir eine durchschnittlich erwartete Lebensdauer der entwickelten Software von knapp zehn Jahren festgestellt, wobei gemäss Aussagen der Interviewpartner die effektive Lebensdauer um einige Jahre höher ausfallen dürfte. Die entwickelte Software muss während ihrer Einsatzzeit gewartet werden. Wir machen in diesem Zusammenhang die folgenden Feststellungen:

a) Die Betrachtung der dargestellten Probleme zeigt, dass die Wartung bestehender Systeme in 14% der Abteilungen gewisse und in gar 32% der Abteilungen sehr grosse Sorgen macht. Dieses Problem ist besonders gross im Bereich Eigengebrauch (13% bzw. 46%); Abteilungen, die im Kundenauftrag arbeiten, kennen dieses Problem hingegen kaum (je 7%).

b) Gemäss Zehnder (1986) beträgt in Europa der Aufwand für den Unterhalt der bestehenden Software 70% des gesamten Software-Entwicklungsaufwandes, in Japan hingegen lediglich 30%. Dies wird durch eine aggressive Erneuerungsstrategie der Japaner begründet. Vorhandene Software wird rascher durch marktgängige Standardsoftware ersetzt. Zum Unterhalt werden hier Fehlerbehebungen, Anpassungen und Erweiterungen gezählt.

Die Vergangenheitsbewältigung ist also in vielen Abteilungen ein Problem. Um es zu entschärfen, ist der Unterhaltsfreundlichkeit der Software bereits bei der Entwicklung eine grosse Bedeutung beizumessen. Es gibt mehrere organisatorische Fragen; wir möchten aber hier eher auf die technische Seite des Problems eingehen. Technische Fragen sind gemäss Sommerville (1985) Abhängigkeiten der Module, die Programmiersprache, der Programmierstil, der Aufwand für das Testen der Module und des Systems, die Qualität und die Quantität der Dokumentation. Wir betrachten hier einen übergeordneten Aspekt.

Eine Normierung des Vorgehens auf allen Stufen der Entwicklung vermag

einen wesentlichen Beitrag zur Unterhaltsfreundlichkeit der entwickelten Software zu leisten. Wir haben festgestellt, dass in 39 Abteilungen Richtlinien vorhanden sind, die meist nur die Projektorganisation und den Projektablauf regeln, nicht aber Vorgehen, Methoden und Werkzeuge in den einzelnen Phasen vorschreiben. Die Richtlinien sollten in diesem Sinne erweitert werden. Die Vorteile einer konsequenten Normierung liegen auf der Hand. Durch ein normiertes und dadurch standardisiertes Vorgehen werden einheitliche Programme und Dokumente erstellt. Diese sind mit geringem Aufwand lesbar und verständlich. Das Erstellen der Richtlinien genügt noch nicht; diese sind den betroffenen Mitarbeitern zu vermitteln, und deren Einhaltung muss konsequent überprüft werden *(Bild 3)*. Einhaltung und Überprüfung werden wesentlich vereinfacht, wenn die vorgeschriebenen Vorgehensweisen und Methoden durch Werkzeuge unterstützt werden.

In verschiedenen Bereichen der Software-Entwicklung ist ein verstärkter Einsatz von Werkzeugen denkbar und wünschenswert. Hier einige Beispiele:

a) Die Richtlinien sollten eine oder mehrere Entwurfsmethoden vorschreiben. Ihre Einhaltung kann durch den Einsatz von entsprechenden Werkzeugen, die diese Methoden unterstützen, gefördert werden. Wir sind uns bewusst, dass das Angebot an effizienten und kostengünstigen Entwurfswerkzeugen relativ klein ist, doch haben wir in einigen Unternehmungen derartige Werkzeuge angetroffen.

b) Nur selten haben wir in den konkreten Projekten einen bedeutenden Einsatz von Standardmodulen festgestellt; in vielen Fällen wäre ein verstärkter Einsatz von Standardmodulen möglich. Diese würden wesentlich zu einer Normierung der entwickelten Software beitragen.

c) In einem Viertel der betrachteten Projekte wurden Syntaxeditoren eingesetzt, es handelt sich dabei häufig um einen gewöhnlichen Editor, der um einen Syntax-Checker erweitert wurde. Es sollten vermehrt Editoren eingesetzt werden, die nicht nur die Syntax überprüfen, sondern die die Darstellung des Source-Codes einheitlich gestalten. Der Verzicht auf einen persönlichen Programmierstil trägt viel zu einer besseren Lesbarkeit der Programme und damit zur Förderung der Unterhaltsfreundlichkeit des Systems bei.

d) Wenn der Programmcode einheitlich gestaltet wird, kann die Moduldokumentation durch ein Werkzeug aus dem Programmcode generiert werden. Dadurch kann mit geringem Aufwand die Dokumentation erzeugt und nachgeführt werden.

8.3. Projekt-Nachkontrolle

Wie bereits dargestellt, wird in 39 Abteilungen eine mehr oder minder detaillierte Zeiterfassung durchgeführt. Diese dient in der Regel zur Lohn- und Aufwandsabrechnung. In rund der Hälfte der Abteilungen werden Probleme, die aufgetreten sind, rapportiert, häufig aber lediglich bis zu ihrer Lösung in eine Pendenzenliste eingetragen. Eine ausführliche Nachkontrolle, wie sie von Zehnder (1986) beschrieben ist, in der die geschätzten und die effektiven Leistungszahlen und Aufwandsabschätzungen gegenübergestellt und die Schwachstellen sowohl der Lösung als auch der Projektentwicklung ehrlich aufgedeckt werden, haben wir nur in wenigen Fällen angetroffen.

Oft wird zu wenig aus den Fehlern gelernt, weil diese den Betreffenden gar nicht bewusst sind. Es fehlen die Zeit, der Wille und die Fähigkeit, kritisch über die eigene Arbeit nachzudenken. Demnach könnte eine bewusste und institutionalisierte Nachkontrolle mittel- bis langfristig wesentlich dazu beitragen, das Erfahrungspotential gezielt und bewusst zu steigern. Dadurch lässt sich vermeiden, dass die gleichen Fehler wiederholt werden. Derartige Nachkontrollen sind weder eine Frage der Qualifikation der Mitarbeiter noch eine Zeitfrage, sondern eine Frage der Selbstdisziplin. Diese sollte gezielt geschult werden.

In diesem Zusammenhang ist nochmals auf die Studie von Zelkowitz et al. (1984) hinzuweisen. Sie enthält die Erkenntnis, dass eine der Hauptstärken der japanischen Software-Industrie in einer gründlichen und effizienten Nachkontrolle besteht.

Peter Dünki, Claudio Galasso

9. Nachwort

Die in der vorstehenden Arbeit komprimierte Doppel-Diplomarbeit leidet unter einigen Mängeln, die nicht den Bearbeitern anzulasten sind: der Zeitraum war sehr knapp, die Terminologie ist allgemein schwammig, für den Inhalt der Fragen konnten wir wohl einige Vorschläge machen, für die richtige Befragungstechnik aber nicht. Schliesslich wollten wir gern einen Überblick gewinnen, wodurch eine – sicher vorteilhafte – Beschränkung auf ein schmales Band im weiten Spektrum ausgeschlossen war. Es ist das Verdienst der Diplomanden, trotzdem eine gute Arbeit angefertigt zu haben.

Einige Punkte sind diskutabel: So ist der im Abschnitt «Wartung» zugrundegelegte Wartungsbegriff eher unüblich, und die Ergebnisse in diesem Punkt werden dadurch verfälscht, dass bei den Abnehmern der Software nicht nachgefragt werden konnte.

Es ist auch nicht klar, dass die empfohlene Beteiligung des Kunden an der Projektleitung ein Allheilmittel ist; in vielen Fällen fehlen dem Kunden fachliche Kompetenz und Personal dafür, und der Hersteller wird ja sein Personal in der Regel nicht dem Kundendelegierten unterordnen.

Peter Dünki und *Claudio Galasso* haben vorsichtig gefragt und gewertet – das entsprach ihrer Aufgabe. Beim Lesen fühlt man sich aber allzuoft wie bei einem Spaziergang durch ein Potemkinsches Dorf. Man weiss ja, wie das in der Praxis aussieht:

- «Richtlinien, deren Einhaltung nicht überprüft wird» (4.4.)
- «De-facto-Standard, an den sich die Betroffenen halten» (4.4.)
- «Implizite Qualitätssicherung» (6.)

Auch wenn die Ergebnisse nicht spektakulär waren, so hat es doch einige kleine Überraschungen gegeben:

- 55% arbeiten auf Grossrechner
- Fortran ist fast bedeutungslos (offenbar wurden keine Physiker befragt)
- Basic wird wirklich verwendet (der Eiger kann halt auch von der Nordseite bestiegen werden)
- Moderne Sprachen wie Modula 2 und Ada liegen noch unter dem Horizont
- Einige Methoden sind bekannt und werden (wenigstens nominell) eingesetzt

Die Befragung hinterlässt insgesamt ein unscharfes Bild. Ich glaube nicht, dass daran die Durchführung schuld ist, sondern dass dies dem Stil der Antworten entspricht. Viel zu selten sind die Projektleiter, die selbstbewusst sagen können: *So* gehen wir vor, und *so* sieht unsere Erfolgsbilanz aus. Bei den meisten Unternehmungen liegt ein Durchbruch zum Software-Engineering noch weit in der Zukunft. Leider gilt auch bei uns die Bemerkung von Redwine und Riddle (1985, p. 196):

One would expect that a high cost would be all right as long as the derived benefit was high, leading to a high «benefit-to-pain» ration. Our case studies do not support this; rather they indicate that the cost must be reasonably low whatever the benefit gained.

Dass die zu erwartenden Gewinne beachtlich sind und auf jedem anderen Gebiet (ausser eben im Software-Engineering) zu hastigen Investitionen führten, belegen die Zahlen aus der «Guide»-Studie (hier zitiert nach dem sehr lesenswerten Artikel von Boehm, 1983): Produktivitätsgewinne von etwa 50% sind normal, und 100% sind nicht aussergewöhnlich. Es scheint aber, dass die Software-Entwickler diese Gewinne nicht aus eigenem Antrieb realisieren wollen, sondern darauf warten, bis sie von den Benützern dazu gezwungen werden, falls sie dann noch im Geschäft sind!

Prof. Dr. Jochen Ludewig

Situation du génie logiciel dans la région zurichoise

Dans le cadre d'un travail de diplôme effecturé à l'institut d'informatique de l'EPFZ, on a procédé à une étude de l'application du génie logiciel dans la région zurichoise. Le but de cette étude était d'inventorier la manière de procéder, les méthodes et les outils logiciels utilisés dans la pratique. Ce travail a montré qu'un modèle déterminé décrivant les étapes du déroulement (Life cycle) est utilisé dans la grande majorité des cas. Il n'en reste pas moins que de nombreux points seraient susceptibles d'amélioration avant que la production de logiciel puisse être considérée comme une branche adulte du génie technique.

Traduction: Softcraft AG, Untersiggenthal

Literatur

Boehm B.W. (1983): Seven basic principles of software engineering. «The Journal of Systems and Software» 3, 3–24

P. Dünki, C. Galasso (1987): Der Stand des Software Engineering im Raume Zürich – eine empirische Untersuchung. Bericht Nr. 80 des Instituts für Informatik, ETH Zürich

Ludewig J. (1985): Techniken der Programmierung im Kleinen und im Grossen. Brown-Boveri-Forschungszentrum, Baden-Dättwil, 1985

Redwine S.T., Riddle W.E. (1985): Software technology maturation. In IEEE: 8th International conference on software engineering, London, pp. 189–200

Sommerville I. (1985): Software engineering. Addison Wesley, Workingham et al., 1985

Zehnder C.A. (1986): Informatik-Projektentwicklung. Verlag der Fachvereine Zürich und Teubner-Verlag, Stuttgart 1986

Zelkowitz et. al. (1984), Yeh M.V., Hamlet R.T., Gannon R.G., Basili J.D., V.R.: Software engineering practices in the US and Japan. «IEEE Computer», June 1984, pp. 57–66