Endstation Wien

BoD
BOOKS on DEMAND

Harry M. Sneed

Endstation Wien

45 Jahre Projekterfahrung in der deutschsprachigen IT-Welt

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Illustration: Harry M. Sneed

Herstellung und Verlag: BoD – Books on Demand, Norderstedt

ISBN: 978-3-7448-8364-1

Inhaltsverzeichnis

Vorwort

Das vorliegende Buch schildert die Geschichte der deutschsprachigen IT-Welt von den Anfängen bis zum heutigen Tag aus den Augen eines Zeitzeugen. Ich kam 1970 als junger amerikanischer Programmierer nach Deutschland. Ich hatte schon vorher drei Jahren im amerikanischen Marine-Ministerium als Systemanalytiker gearbeitet. Begonnen habe ich hier als Entwickler in der deutschen Hochschulverwaltung. Später bin ich zu Siemens rüber gewechselt und habe dort in der Datenbankentwicklung mitgearbeitet. Von Siemens aus bin ich nach Budapest gegangen um dort ein Software Testlabor für Siemens Projekte aufzubauen. Daraus ist später eine Software-Toolfabrik hervorgegangen in der CASE Werkzeuge für den deutschen Markt entwickelt wurden. Mit jenen Werkzeugen und dem SoftOrg Lebenszyklusmodell bin ich in mehrere namhafte deutsche Unternehmen hineingekommen und half dort die IT-Welt zu gestalten. In den 80er Jahren des vorigen Jahrhunderts erlebte ich hautnahe das Projektgeschehen in den bundesdeutschen Großbetrieben, mal als Held, mal als Schurke. Ich nahm Teil an den innerbetrieblichen Machtkämpfen und wurde nicht selten deren Opfer. Das Buch schildert nicht nur die damalige Technologie, sondern auch die vorherrschenden Methoden und Vorgehensweisen jener Zeit. Ich habe sie alle durch gemacht angefangen mit der strukturierten Programmierung, über CASE und 4GL Sprachen, bis zur objektorientierter Programmierung. Meine Erfahrungen habe ich in 23 Büchern und über 400 Fachartikel festgehalten.

Ich war lange Zeit ein Wanderer zwischen zwei Welten – der sozialistischen in Ungarn und der marktwirtschaftlichen in Deutschland. In Ungarn war ich Entwicklungsleiter, in Deutschland Projektberater. Mit der Wende verlor ich meine Entwicklungsunterstützung durch den ungarischen Staat und musste dort die Produktentwicklung einstellen. Danach bin ich in die Schweiz gegangen um dort Reengineering-Projekte für eine Großbank durchzuführen. Dort habe ich eine andere IT-Welt miterlebt – eine straff organisierte Welt. Über sieben Jahre habe ich in der Schweiz an vorderster Front der Softwaremigration mitgewirkt. Es war beruflich meine erfolgreichste Zeit. Nebenbei blieb ich weiterhin in der wissenschaftlichen Welt aktiv sowohl auf der deutschen Szene durch die GI und auf der internationalen Szene durch die IEEE Konferenzen. 1997 musste ich die Schweiz verlassen. Nach einem kurzen Aufenthalt in Deutsch bekam ich 1998 eine Einladung nach Wien zu kommen und dort bin ich auch bis zum heutigen Tag hängen geblieben, nicht weil ich Wien so mag, sondern weil die Wiener die Einzigen sind die mich beschäftigt haben.

Der Titel „Endstation Wien“ ist an dem Buch von Tom DeMarco und Tim Lister angelehnt. Das letzte Kapitel in ihrem Buch heißt „Wien wartet auf Dich“. Eigentlich geht es in diesem Buch darum den Leidensweg eines Softwareentwicklers und –Testers zu schildern. Der Wandel der IT-Welt ist hier anhand von Projekterfahrungen geschildert – gute und schlechte. Leider dreht sich die IT-Welt oft im Kreis. Auf der einen Seite ändert es sich, auf der anderen Seite werden dieselben Fehler wiederholt - La plus change, la plus de meme chose. Um aus diesem Teufelskreis herauszubrechen muss die nächste Generation aus den Erfahrungen der letzten lernen. Die entscheidende Frage ist ob es sich lohnt an diesem Wettbewerb teilzunehmen. Jeder muss das für sich entscheiden. Ich habe mich daran beteiligt und ich bereue es nicht, obwohl ich in Wien als alter Tester gelandet bin. Es ist nicht das Ende was zählt, sondern der Weg dahin.

Harry M. Sneed
Arget, 2017

1 Wien wartet

Die erste Ausgabe eines Buches mit dem Titel „Peopleware – Productive Projects and Teams“ von Tom De Marco und Tim Lister erschien in New York im Jahre 1987 [DeMa87]. Die zweite Ausgabe erschien 10 Jahre später. In dem Buch geht es darum, Menschen in IT-Projekten zu führen. De Marco und Lister waren nämlich der Meinung, dass die meisten Probleme in IT-Projekten weniger technischer Natur, sondern viel mehr menschlicher Natur sind. Sie entstehen durch fehlende Motivation, fehlende zwischenmenschliche Kommunikation und fehlende Führung. Die Menschen in einem IT-Projekt werden nicht geführt, sondern nur verwaltet und wenn sie geführt werden, dann allzu oft in eine falsche Richtung. Die beiden Autoren mit ihrer 30jährigen Projekterfahrung haben es sich vorgenommen, ein Buch für IT-Projektleiter zu schreiben, die sie auf die gefährlichsten Tücken und die wichtigsten Erfolgsfaktoren hinweist. Das Buch war in Amerika ein großer Renner. Deshalb hat der Hanser Verlag bereits 1990 entschieden, das Buch ins Deutsche zu übersetzen. Der Übersetzer war der renommierte Software-Berater Dr. Peter Hruschka aus Wien, der lange Zeit in Deutschland mit der CASE Entwicklung beschäftigt war und der später zu der Beratungsassoziation von De Marco und Lister, das Atlantic Guild zugestoßen ist. In Anbetracht der vielen soziologischen und psychologischen Aspekte, die in dem Buch behandelt werden, hat Hruschka es ziemlich frei übersetzt und der mitteleuropäischen Denkweise angepasst. Der neue Titel hieß: „Wien wartet auf dich! Der Faktor Mensch im DV-Management“. Der deutsche Titel wurde aus einem Kapitel des Buches mit der Überschrift „Vienna is waiting on you“ abgeleitet. Das Kapitel erzählt von gescheiterten Projektleitern, die in der Psychologie landen, weil sie die Prinzipien, die in dem Buch propagiert werden, missachtet haben.

In der zweiten Ausgabe, die 1999 erschien, wurde ein neuer Teil hinzugefügt, ein Teil über Prozessverbesserungsprogramme, lernende Organisationen, interner Wettbewerb und Projektpolitik. Dieser neue Teil trug den Titel “Wien wartet noch immer“ [DeMa99]. Was aber, um Gottes Willen, hat Wien mit überarbeiteten Projektleitern zu tun? Warum nicht New York oder San Francisco, Berlin oder Zürich? Warum ausgerechnet Wien, wo die Wiener von der Mentalität her am wenigsten unter solchem Stress leiden. Sie haben zwar viele Wehen, über die sie ständig raunzeln aber diese Krankheit bleibt ihnen erspart. Man fragt sich also, warum De Marco und Lister Wien als Endstation ausgewählt haben.

Über diese Frage könnte man eine Doktorarbeit schreiben. Es gibt verschiedene Interpretationen. Alle gehen auf ein Lied von Billy Joel aus den 70er Jahren zurück, eine Zeit in der viele Protestsänger den Sinn der Arbeit in Frage stellten.

„Da kommt ein Tag in Wien“
übersetzt von Ulla Meinecke

Langsam, du verrücktes Kind !
Dein Ehrgeiz ist zu sehen wo die Juwelen sind.
Doch sag, wenn du so gut bist,
wovor hat du dann so viel Angst? hmm
Ey, wo brennt's? Wozu der Alarm?
Du zündest alles an.
Doch es wird noch nicht warm.
Du hast so viel zu tun, dass du endlose Tage verlangst. hmm
Da kommt ein Tag, da zählt nur heiß und kalt.
Da wirst du sein was du willst oder du wirst nur alt.
Wenn du nur Sterne siehst fällst du auf's Gesicht. hmm
Und warum merkst du nicht:
Wien wartet auf dich.

Langsam, du machst es gut!
Du kannst nicht ernten bevor die Zeit das ihre tut,
obwohl es so romantisch ist auf Messers Schneide
heut' Nacht, heut' Nacht.
Zu lang mit Vollgas gehst du drauf.
Du bist dir so weit voraus, dass du verlierst was du brauchst.
Obwohl du weißt was dir fehlt kannst du nicht seh'n was du hast!
Du hast, hast deinen Stolz und deine Leidenschaft.
Doch nur ein Narr kann glauben er hat endlos Kraft.
Mach's gut! Doch steh' nicht da und sag: Es werde Licht!? hmm
Und warum merkst du nicht:
Wien wartet auf dich.

Langsam, du verrücktes Kind!
Lass das Telefon geh'n und gib dir Sonne und Wind.
Es ist okay für ein paar Tage nur ein kleines Licht. hmm
Und warum merkst du nicht:
Wien wartet auf dich.

Kommt ein Tag, da zählt nur heiß und kalt.
Da wirst du sein was du träumst oder du wirst nur alt.
Wenn du nur Sterne siehst fällst du auf's Gesicht. hmm
Und warum merkst du nicht:
Wien wartet auf dich.
Und warum merkst du nicht:
Wien wartet auf dich! [Mein77]

Es gab in der 70er Jahren viele Lieder dieser Art. Die deutsche Version dieses Liedes gibt Hruschka in seiner Übersetzung wieder:

„Aber du verstehst, wenn dir jemand die Wahrheit sagt,
du kannst haben, was du willst, oder einfach nur alt werden,
du wirst vielleicht ins Gras beißen, bevor du die Hälfte der Strecke erreichst
Wann merkst du endlich, dass Wien auf dich wartet!“

Laut Hruschka, der es als geborener Wiener wissen sollte, ist das Wien, das in dem Lied von Billy Joel auf dich wartet, die letzte Station in einer persönlichen Reise. Wenn du dort ankommst, ist alles vorbei. Warum aber Wien als Endstation eines arbeitsreichen Lebens? Dafür gibt es mindestens drei Erklärungen.

Zum einen haben viele Amerikaner Wien als Reiseziel im Auge. So wie viele Deutsche sich eine Reise nach Bangkok wünschen, wünschen viele Amerikaner sich eine Reise nach Wien, aber nicht wegen der schönen Frauen, die es dort gibt, sondern wegen der vielen Schlösser. Außerdem drängen ihre Ehefrauen sie dazu, weil sie in einem Wiener Café sitzen und echte Sachertorte essen wollen. Die Nachbarfrau war auch schon dort und es darf nicht sein, dass sie überall über ihre Erlebnisse erzählt und die eigene Frau kann nicht mitreden. Nach dieser Auslegung ist Wien ein fernes Reiseziel, das wegen Überbeschäftigung immer wieder verschoben wird.

Zum zweiten ist Wien der Geburtsort der Psychotherapie. Welcher Platz wäre besser geeignet für die Behandlung eines arbeitssüchtigen Projektleiters, der mit seinen Nerven am Ende ist, als die Heimat von Sigmund Freud und Alfred Adler. Man musste schnell nach Wien zur Behandlung noch vor dem endgültigen Nervenzusammenbruch. So geht das Lied von Billy Joel weiter:

„Sei nicht so verrückt, schalte einen Gang zurück,
lege den Hörer ab und verschwinde für einige Zeit.

Das ist schon ok, du kannst dir ein, zwei freie Tage leisten,
wann merkst du endlich, dass Wien auf dich wartet."

Also, was auf dich wartet, ist die Psychiatrie, die in Wien ihren Ursprung hatte. In diesem Zusammenhang gehen De Marco und Lister auf die Grenzen des Menschen ein. Man darf weder von sich noch von seinen Projektmitarbeitern unmenschliches erwarten. Jeder stößt an seine Grenzen. So schreibt das Autorenduo „Kein Mensch kann wirklich viel mehr als vierzig Stunden arbeiten, zumindest nicht über längere Zeiträume hinweg und mit der Intensität, die für kreative Arbeit erforderlich ist..." Und wenn er das tut, denn endet er früher oder später in der Psychiatrie, bzw. in Wien.

Dies ist eine unbewiesene Hypothese. Früher haben alle Menschen weit mehr als 40 Stunden die Woche gearbeitet und sind nicht daran erkrankt. Wer 60 Stunden oder mehr die Woche arbeitet, ist noch lange kein Fall für den Psychiater. Es kommt darauf an, wie er zu der Arbeit steht und was er mit den restlichen Stunden macht. Ich habe in den letzten 40 Jahren kontinuierlich mehr als 50 Stunden die Woche gearbeitet. Vielleicht bin ich seelisch krank, das merkt man selber nicht, aber beim Psychiater bin ich noch nicht gelandet, auch wenn mich das Leben nach Wien geschlagen hat.

Zum dritten ist Wien, zumindest in den 70er Jahren, als das Lied „The Stranger" von Billy Joel entstand, eine Grenzstadt gewesen [Joel75]. Hinter Wien kam gleich der Eiserne Vorhang. Die westliche Zivilisation hörte auf. Wien war in diesem Sinne eine wahre Endstation. Hier war die Reise zu Ende, zumindest für westliche Touristen. In dem Lied von Billy Joel kommt das zum Vorschein.

„Tritt doch ein wenig auf die Bremse, du bist ohnehin gut.
Du kannst nicht alles jetzt erreichen, deine Zeit kommt noch
obwohl es draußen an der Grenze sicherlich spannend ist.
Wann merkst du endlich, dass man auf dich wartet?"

Welche Grenze ist hier gemeint? Es kann wohl nur die hinter Schwechat gemeint sein. Dort endet die gute Welt, so wie es die Amerikaner gekannt haben, und es beginnt das Reich des Bösen sowie es Präsident Reagan betitelt hat. Also, gehe nicht über Wien hinaus, auch wenn es sicherlich spannend ist. In Wien müssen alle aussteigen. Es ist die Endstation.

Eine letzte Erklärung kommt von Billy Joel selbst. Als Autor des Liedes musste er es am besten wissen. Das Lied hat er gerade nach einem Besuch in Wien, wo sein

Vater wohnte, geschrieben. Dort hat es ihn sehr beeindruckt, wie respektvoll alte Menschen behandelt werden. Er meinte, dort könnte man als alter Mensch gut leben. Also braucht man keine Angst vor dem Alter zu haben, nur ab nach Wien. Dort kann man sein Lebensende genießen. Wien, als Paradies für Rentner. Wer das weiß, muss sich in jüngeren Jahren nicht abhetzen, denn Wien wartet auf ihn.

Eigentlich passen alle diese Interpretationen. Wien ist auf jeden Fall ein lohnenswertes Reiseziel, vor allem für gestresste amerikanische und asiatische Manager. Dort werden sie von der Last der Gegenwart durch eine Reise in die Vergangenheit abgelenkt, denn Wien riecht nach dem Vergangenen. Die Stadt ist ein einziges Denkmal für eine glorreiche Vergangenheit. Wer also der Gegenwart entkommen will, der reist nach Wien. Wien symbolisiert nach Stephan Zweig die Welt von Gestern [Zweig82].

Wer eine psychotherapeutische Behandlung braucht, ist ebenfalls in Wien gut aufgehoben. Pro Kopf der Bevölkerung hat Wien mehr Psychiater als irgendeine andere Stadt einschließlich New York. Das soll nicht heißen, dass Wien so viele psychisch Kranke hat, obwohl die Wiener in der Tat zu Depressionen neigen und die Selbstmordrate hoch ist. Es liegt eher an der staatlichen Krankenversicherung, die jeden gegen alles versichert. In Amerika können die wenigsten es sich leisten zum Psychiater zu gehen, in Wien kann es sich ein jeder leisten, eine Folge von 60 Jahren sozialdemokratischer Herrschaft.

Auch das mit der Grenze hat was für sich. Hinter Wien beginnt nicht mehr das Reich des Bösen, sehr wohl aber der wilde Osten. Bis Wien herrscht Recht und Ordnung. Darüber hinaus folgt Armut und Kriminalität. Nach dieser Ansicht ist Wien sehr wohl eine Grenzstadt. Als Einer, der fast wöchentlich zwischen Wien und Budapest pendelt, werde ich an den Unterschied im Lebensstandard ständig erinnert. Hier gibt es nach wie vor zwei Welten, die zwar nur durch einen Grenzstreifen getrennt sind aber doch so unterschiedlich sind wie Tag und Nacht. Alle hoffen, dass dies sich mal ändert aber bis jetzt ist das nur eine Vision geblieben.

Schließlich haben wir Wien als Paradies für Rentner. In der Tat werden alte Menschen dort recht gut behandelt und nicht wie in vielen anderen Orten missachtet. Gescheiterte IT-Projektleiter können sicherlich dort noch einen Platz unter einer Brücke mit den restlichen Sandlern finden, aber so gut haben sie es auch nicht. Und weil jeder Wiener weiß, dass dieser Zustand auf ihn wartet, sehnt er sich danach. Statt sich mit dem jeweiligen Projekt zu beschäftigen, beschäftigt er sich lieber mit seiner Rente und jeder IT-Manager neigt dazu, die Probleme vor sich her zu schieben in der Hoffnung, er wird vorher den Ruhestand erreichen ehe die Probleme ihn einholen. Der

Wiener IT-Manager muss nicht auf Wien warten, er ist schon angekommen. Auf ihn wartet nur noch ein schattiger Platz im Zentralfriedhof, hoffentlich nicht zu weit vom Eingang, damit seine alten IT-Kollegen ihn gelegentlich besuchen. Vielleicht ist es deshalb so schwierig, IT-Projekte mit Wienern durchzuführen. Sie sind schon einen Schritt voraus.

Jedenfalls ist Wien das Ende meiner persönlichen Reise durch die IT-Welt in Mitteleuropa. Begonnen hat sie in Hannover bei der H.I.S. Hochschul-Informationssystem im Jahre 1970 und enden tut sie in Wien bei der ANECON GmbH. Dazwischen liegen viele Zwischenstationen bei Anwendern und Herstellern von Softwaresystemen in Deutschland, der Schweiz, Österreich, Italien und Ungarn. In den 40 Jahren habe ich an mehr als 100 Projekten teilgenommen, über 50 Softwarewerkzeuge entwickelt, 22 Bücher verfasst oder mitverfasst, an 14 verschiedenen Universitäten und Fachhochschulen gelehrt und mehr als 400 Fachartikel veröffentlicht. Dabei habe ich viele Erfahrungen gesammelt, Erfahrungen, die ich mit diesem Buch weitergeben möchte. Denn trotz der äußerlichen Veränderungen in der IT-Technologie in dieser Zeit ist vieles gleichgeblieben. Es sind weniger die Probleme der neuen Technologie, die uns in den IT-Projekten zu schaffen machen, als vielmehr die uralten Probleme der Projektplanung und der Arbeitsorganisation, wie De Marco in seinem Peopleware Buch feststellt. Insofern sind die Erfahrungen aus der IT-Welt doch noch zeitlos.

Die junge IT-Generation tut gut daran, aus den Erfahrungen der letzten Generation zu lernen. Sie könnte dadurch manche Falle vermeiden. Außerdem helfen Kenntnisse der alten DV-Methoden und Techniken die neuen IT-Technologien besser zu verstehen. Es ist erstaunlich viel von den alten Problemlösungsansätzen in den neuen übernommen worden, viel mehr als die neue Generation ahnen kann. Wer versteht wie und warum diese alten Ansätze zustande gekommen sind, wird die neuen Ansätze besser einschätzen können. Leider fehlt hierfür die Einsicht. Es ist erstaunlich, wie wenig die junge Generation von Informatikern von der bisherigen Informatik wissen. Dabei wäre dies für das Verständnis der Welt, in der sie arbeiten von großer Wichtigkeit. Sie würden nämlich wissen, dass die neuen Konzepte, die sie für die endgültige Lösung halten, weder neu noch endgültig sind. Sie sind nur eine weitere Iteration in einer endlosen Schleife bei der immer nur ein paar Variablen sich ändern. Die Mehrzahl der Daten bleibt konstant.

GAST-KOMMENTAR

HARRY SNEED
SES, Ottobrunn

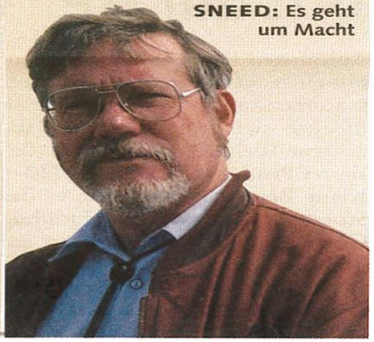
Die IT braucht „junge Alte“

Eine verteilte, vernetzte, objektorientierte Datenverarbeitung auf der Basis von Corba oder Active X mit Internet- und Intranet-Anschluß, grafischen Benutzeroberflächen und flexiblen, frei gestaltbaren Geschäftsprozessen gilt allerseits als das Modell der Zukunft. IT, die nicht dieser Welt angehört oder sich wenigstens auf dem Wege dahin befindet, wird als rückständig eingeschätzt. Dennoch zählen noch immer 75 Prozent der operativen Anwendungssysteme zu den Host-zentrierten, monolithischen, prozeduralen Programmen mit textbasierten Oberflächen und satz- oder segmentorientierten Datenbanken.

Argumente, die Systeme nicht abzulösen, lassen sich immer finden: Termindruck, Rückwärtskompatibilität, die hohen Kosten der Migration sowie Unsicherheit über den Nutzen der neuen Techniken. Manager beweisen ein allzu offenes Ohr für diese Argumente. Der wahre Grund bleibt jedoch unausgesprochen: der Machterhalt.

Cobol, PL/1, IMS, VSAM, UDS, Adabas und Natural sind Strandgut vergangener DV-Zeiten. Heute kümmert sich eine Clique „alter“ Programmierer darum, die nur im Sinn haben, ihre Vormachtstellung so lange wie möglich zu erhalten. Hier spielt sich ein gewöhnlicher Generationskonflikt ab.

Dagegen hilft nur eines: Bereitwillige „Alte“ müssen gefunden und in junge Projekte integriert werden, zumal es ohnehin noch zuwenig Experten für die neuen Technologien gibt. Sie haben sich dort den jungen Mitarbeitern anzupassen nach dem Motto: Opa fährt Skateboard. Durch Migration oder, wo das nicht möglich erscheint, durch objektorientiertes Verschalen (Wrapping) läßt sich alter Code peu à peu ablösen. Die „Alten“, die nicht so flexibel sind, dürfen so lange ihre Legacy-Systeme pflegen, bis diese endlich verschwunden sind.

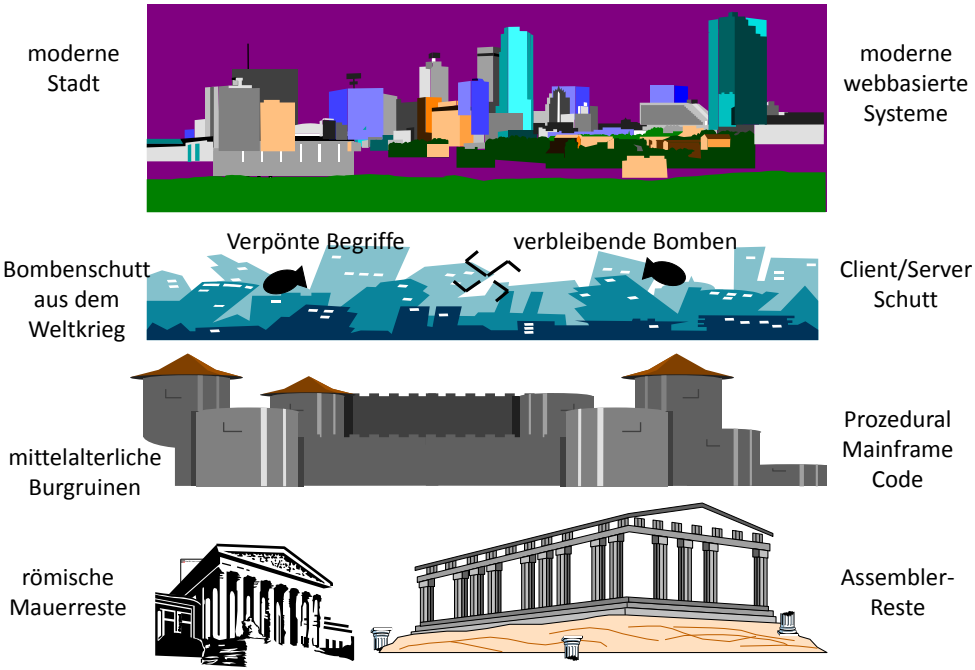


SNEED: Es geht um Macht

Es wäre deshalb anzustreben, so etwas wie eine Geschichte der Informatik als Pflichtfach für alle angehenden Informatiker vorzuschreiben. Sie sollen damit lernen, wie die Welt in der sie arbeiten wollen, zustande gekommen ist, welche Probleme ihre Vorgänger hatten und wie sie sie gelöst haben. Vor allem sollten sie lernen, wa-

rum sie so gelöst wurden wie sie sind. Dann hätten sie mehr Verständnis für die vielen alten Systeme, die heute noch die Hauptlast der IT-Welt tragen und die sie ablösen sollten.

Dieses Buch sollte dazu einen Beitrag leisten. Es beginnt mit der Datenverarbeitung am Anfang der 70er Jahre und endet mit der Cloud Technologie des 21. Jahrhunderts. Vier Jahrzehnte IT-Geschichte liegen dazwischen. Sie werden aus der Sicht eines Zeitzeugen geschildert, der immer bemüht war, die neuesten Themen in die Praxis umzusetzen. Es ist also kein Geschichtsbuch, das nur Fakten und Daten weitergibt, sondern eine Erlebnisgeschichte, die die erlebten Erfahrungen des Autors im Kampf ums Überleben in einer hart umkämpften IT-Welt schildern. Nichts hilft mehr, den Blick für das Wesentliche einer Welt zu schärfen, als die Notwendigkeit darin bestehen zu müssen. Dies war ja auch die Lebenssituation des Autors, der als selbstständiger Unternehmer sich immer am Rande des Überlebens bewegte, im Spannungsfeld zwischen dem großen Erfolg und dem totalen Absturz. Zum Glück ist weder das eine noch das andere passiert. So konnte es der Autor schaffen, als freischaffender Software-Berater in einem kleinen Ziviltechnikerbetrieb in Wien dieses Buch zu Ende zu bringen.



Das Buch beginnt mit dem Deutschland der 70er Jahre als die meisten DV-Systeme im Batchbetrieb liefen. Der Autor begann als Angestellte bei der HIS in Hannover und wechselte später zu Siemens in München wo er im Datenbankbereich tätig war. Dann kam das große I.T.S. Projekt und der Beginn seiner Karriere als Softwaretester. Die Umstände haben dazu geführt, dass er zusammen mit dem amerikanischen Testexperte Ed Miller in Budapest ein Testlabor gründete. Dieses Software Testlabor war das erste seiner Art in Europa. Mit dem Ende des Testlabors ging die Kooperation mit den Ungarn auf dem Gebiet der Software Engineering Werkzeuge weiter. Der Autor hatte eine Vision den gesamten Softwareentwicklungszyklus von der Projektplanung bis zur Produktabnahme zu automatisieren. Diese Vision wollte er mit einer Werkzeugkette namens SoftOrg verwirklichen. Zu einer Zeit waren 26 Institutsmitarbeiter in Ungarn an dem SoftOrg Projekt beteiligt. Um die Werkzeuge zu vertreiben gründete er eine Firma in Deutschland – Software Engineering Services – zur Förderung der deutschen Softwaretechnologie. Die 80er Jahre waren das Jahrzehnt des IT-Aufbruches. Es gelang tatsächlich viele der Werkzeuge bei großen Anwenderfirmen einzubringen und damit Projekte abzuwickeln. Die Anwender gehörten zu den führenden Unternehmen Deutschlands. Ein Kapitel befasst sich mit der Ent-

wicklung der Werkzeuge in Ungarn und ein anderes mit dem Einsatz der Werkzeuge in Deutschland. Dort stellte sich heraus, dass die klassische Top-Down, phasenweise Entwicklung nicht immer zum gewünschten Ziel führt, auch dann nicht, wenn sie von Tools unterstützt wird.

Dann folgt der Zusammenbruch der sozialistischen Länder und das Ende der SoftOrg Entwicklung. Der Autor war gezwungen einen neuen Weg einzuschlagen. Dieser Weg führte über das Reengineering in die Schweiz. Dort hat der Autor zusammen mit einigen seiner ehemaligen ungarischen Mitarbeiter jahrelang Reengineering und Migrationsprojekte durchgeführt. Nach 7 Jahren wurde die Arbeitsbewilligung für den Autor und die Ungarn nicht länger gebilligt. Er musste die Schweiz verlassen. Nach dem Abschied von der Schweiz folgten einzelne Reengineering, bzw. Kapselungsprojekte in Deutschland, aber ein dauerhafter Auftrag blieb aus. So kam es, dass er 1998 nach Wien zu einem großen Bankprojekt gezogen ist. Dort hat er Jahre lang geprüft, getestet und neue Werkzeuge entwickelt bis er 2003 zusammen mit 140 Anderen entlassen wurde. Der Versuch in Deutschland wieder Fuß zu fassen ist nicht gelungen, so kehrte er bald wieder als Tester nach Wien zurück. Zwischen 2003 und 2013 hat er bei der Firma ANECON an mindestens 20 Testprojekten teilgenommen. Aber nicht nur an Testprojekten. Er war auch an Migrations- und Messprojekten beteiligt oder hat sie federführend durchgeführt. Wieder kam eine Menge Projekterfahrung dazu, so dass bis zu seiner Entlassung von der ANECON im Jahre 2014, er beinahe 45 Jahre Projekterfahrung im deutschsprachigen IT-Raum gesammelt hat. Über die späteren Erfahrungen im Bereich der Qualitätssicherung gibt es zwei Kapitel, eins über die Testprojekte und eins über die Messprojekte.

Diese Geschichte erstreckt sich über vier Jahrzehnte. Jedes Jahrzehnt war von einer bestimmten Technologie geprägt. Die 70er Jahren waren das Jahrzehnt der standalone-Batch-Systeme, die 80er Jahren waren das Jahrzehnt der integrierten Online-Systeme, die 90er Jahren das Jahrzehnt der verteilten, objektorientierten Systeme und die 00er Jahren das Jahrzehnt der Web-basierten Systeme. Jedes Jahrzehnt hatte seine eigene Ideale und Risiken. Die IT-Welt ist immer dabei sich neu zu definieren. Die Lösungen des einen Jahrzehnts sind die Probleme des Nächsten. Es ist eine endlose Schleife aus der es kam Entkommen gibt.

Die ständig wandelnde IT-Praxis wird hier aus der Sicht eines Beteiligten geschildert, der viele Rollen in diesem Geschehen wahrgenommen hat:

- als Programmierer
- als Designer
- als Manager

- als Tester
- als Toolentwickler
- als Publizist und
- als Lehrer.

Dabei werden die Ereignisse immer aus der Sicht der jeweiligen Rolle betrachtet. Der Fortschritt in der Informatik wird zum einen von den Visionen der Forschenden und zum anderen von den Herausforderungen der Praxis vorangetrieben. Diese beiden Triebkräfte bedingen sich gegenseitig. Ohne den Druck die alltäglichen Probleme zu lösen gebe es keine Visionen. Es ist keineswegs so, dass die IT-Welt von oben bzw. von der Vision der Verantwortlichen ausgetrieben wird. In der IT-Welt sind die Verantwortlichen eher die Getriebenen. Sie reagieren auf die Ereignisse, die von unten ausgelöst werden. Die treibende Kraft ist die Hardware und Kommunikationstechnologie. Sie bestimmt die Basis-Software, die Betriebssysteme, Datenbanksysteme und die Kommunikationssysteme. Diese bestimmen wiederum welche Werkzeuge bzw. welche Sprachen zur Verfügung stehen und diese bestimmen die Entwicklungsmethoden. Die Prozesse folgen erst am Ende dieser Kausalkette. Ausschlaggebend für die Entstehung der Werkzeuge, Methoden und Prozesse einer Epoche sind die typischen Projekte, die in einer Epoche ausgeführt werden. Sie stellen die Ziele, die angestrebt werden und die Ziele verbinden die herrschende Technologie und die verfügbaren Werkzeuge mit den Entwicklungsmethoden und der Managementphilosophie. Die Teilaspekte bekommen erst durch zielgerichtete Projekte einen Sinn und schmelzen zu einem Ganzen zusammen, was wir als IT-Technologie eines Zeitalters bezeichnen können. Zu Beginn eines jeden Zeitabschnittes findet einen technologischen Durchbruch statt. Dieser zieht neue Produktionsmittel, bzw. Sprachen, Techniken und Werkzeuge nach sich. Nach Marx bestimmen die Produktionsmittel die Produktionsweise [Marx94], d.h., die Sprachen, Techniken und Werkzeuge bestimmen hier die Methoden und diese bestimmen die Prozesse. Am Ende entsteht eine Vision wie die Prozesse zu gestalten sind. D.h. zuerst kommt die Praxis, dann folgt die Theorie. Die Entstehung einer sozio-technologischen Welt wäre nicht möglich ohne die Zielrichtung, die von den laufenden Projekten gegeben wird. Es sind vor allem die Projektanforderungen, die die IT-Technologie vorantreiben. Deshalb ist es so wichtig, die Lösungen der Vergangenheit zu kennen um die Probleme der Gegenwart zu lösen. Die größte Herausforderung der Informatik ist die Bewältigung der eigenen Vergangenheit. Durch die Schilderung vergangener Projekte soll dieses Buch dazu einen Beitrag leisten.

2 Datenverarbeitung in den 70er Jahren

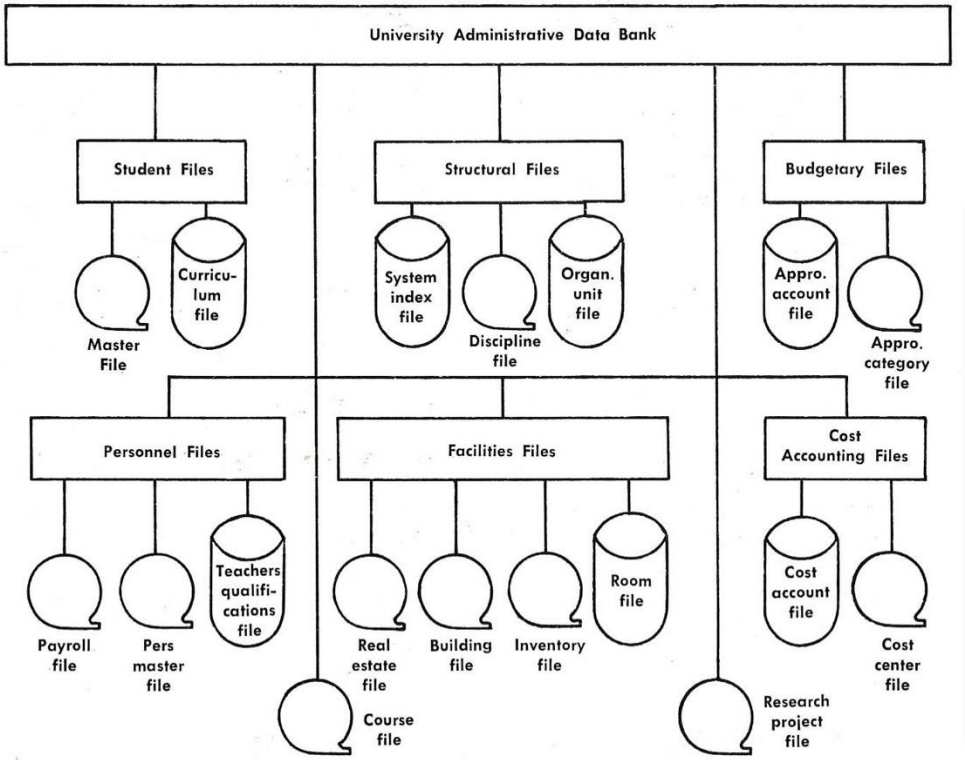
2.1 Anfang bei der H.I.S in Hannover

Es war 1970 als ich auf Wunsch meiner Frau meinen Job als Programmierer / Analyst im U.S. Navy Department in einem Vorort von Washington D.C. aufgab und nach Deutschland auswanderte. Meine Frau ist Deutsche und wollte in ihr Heimatland zurückkehren. In Amerika standen mir zwei Wege offen – an der Universität Maryland zu bleiben, zu promovieren und eventuell Professor zu werden oder im Civil Service zu bleiben und eventuell Manager zu werden.

Ich habe weder den einen noch den anderen Weg eingeschlagen. Ich bin nach Deutschland gekommen. Das Schicksal wollte es so. Meine Frau war schon früher zurückgekehrt und hatte für mich in der Zeitung inseriert. „Amerikanischer Programmierer / Analytiker mit Masters Degree in Public Administration, zwei Jahre Berufserfahrung und guten Deutschkenntnissen sucht Stelle in der Bundesrepublik als Systemanalytiker". Daraufhin bekam sie sage und schreibe 33 Angebote. Wir haben 7 ausgewählt und nachdem ich angekommen war, alle 7 aufgesucht – von Konstanz am Bodensee bis nach Hamburg an der Elbe. Es gab darunter einige sehr interessante Möglichkeiten in der Industrie zu arbeiten, aber da ich zum öffentlichen Dienst neigte – ich hatte nämlich eine Abneigung gegen die freie Marktwirtschaft – entschied ich mich für eine Gesellschaft des öffentlichen Rechts – die Hochschulinformationssystem Gesellschaft in Hannover. Es schien mir der Platz zu sein, wo ich am wenigsten dem Leistungsdruck der Marktwirtschaft ausgesetzt wäre. Ich suchte eine freundliche, stressfreie Umgebung mit Anschluss an die akademische Welt, in der ich mich in Ruhe entfalten konnte. Ich spielte damals noch mit dem Gedanken, die Promotion nachzuholen und doch noch Professor zu werden. Was konnte man sich besseres wünschen, als Professor in Deutschland zu werden? Bei der HIS wäre ich wenigstens in der Nähe dieses Traumes, und die Arbeitsatmosphäre war auch ansprechend. Viele junge Akademiker mit dem Ziel, die deutsche Hochschullandschaft aufzukrempeln. Da es unter ihnen kaum welche gab mit IT-Erfahrung, hatte ich dort die Chance, mich zu profilieren. Und so kam es auch. Ich konnte mich bei der HIS richtig ohne allzu großen Leistungsdruck entfalten und meine Sicherheit in einer fremden Welt gewinnen. Es war also der Wunsch, dem Leistungsdruck zu entgehen, der mich zu Beginn meiner Laufbahn in Europa nach Hannover schlug und es war der Wunsch, dem Leistungsdruck zu entkommen, der mich am Ende dieser Laufbahn nach Wien brachte.

2.2 Projekte in der deutschen Hochschulverwaltung

IT-Projekte zu Beginn der 70er Jahre und insbesondere Projekte im öffentlichen Dienst waren weit entfernt von dem Hochdruck, termingetriebener Projekte, die wir heute kennen. Die Beteiligten hatten Zeit. Ob der Termin um 6 Monate überschritten wurde, hat niemand gestört. Man war froh, dass sie überhaupt fertig geworden sind. Der Entwickler, oder der Programmierer, so wie er damals bezeichnet wurde, genoss eine ziemlich große Freiheit. Keiner wagte ihn in irgendwelcher Weise zur Rechenschaft zu ziehen, nicht einmal der Projektleiter. Zu kostbar waren seine Talente und zu wenig Programmierer waren es überhaupt. Ergo, konnte der Programmierer relativ ungestört ohne jegliche Kontrolle arbeiten. Dies ist der Hauptgrund für den katastrophalen Zustand mancher Legacysysteme, die in den 70er Jahren entstanden sind, aber das ist ein anderes Kapitel.



Es herrschte damals das Wasserfallmodell und keiner hätte gewagt, es in Frage zu stellen [Royce70]. Die Arbeitsteilung und damit die berufliche Rangordnung waren davon abhängig. Tester hat es noch nicht gegeben, getestet haben die Endanwender selbst, so dass der Programmierer trotz seiner großen Freiheiten am Ende der Rangordnung stand. Über ihn gab es einen Designer, der heutige Architekt, und über den der Systemanalytiker. Es war die Aufgabe des Analytikers, die Anforderungen der Benutzer zu analysieren und in ein sogenanntes Fachkonzept niederzuschreiben. War das Fachkonzept einmal fertig, konnte der Analytiker für den Rest des Projektes zuschauen. Seine Arbeit war erledigt. Der Designer hatte die Aufgabe, das Fachkonzept in ein technisches Konzept umzusetzen. Das tat er in Absprache mit dem vorgesehenen Programmierer. Danach übernahm der Programmierer das Design und setzte es in Codes um. So war es zumindest in der Theorie. In der Tat musste der Programmierer

sehr viel nacharbeiten, die Lücken im Fachkonzept schließen und die Fehler im Design ausbügeln.

So kam es, obwohl der Programmierer der niedrigste in der Rangordnung war, hatte er die größte Verantwortung. Er musste dafür sorgen, dass aus der inkonsistenten, unvollständigen Spezifikation und dem oft realitätsfernen Entwurf doch noch eine lauffähige IT-Lösung entstand, welche die Bedürfnisse der Anwender halbwegs abdeckte. Diese Rolle habe ich akzeptiert und versuchte das Beste daraus zu machen.

Außer mir gab es kaum jemand bei der HIS der programmieren konnte. Die meisten sahen sich als Systemanalytiker, einige andere als Designer und der Programmierer war ich – der nützliche Amerikaner. So sah es auch in den Projekten aus. Mein erstes Projekt war an der Universität Erlangen. Verantwortlich dafür war Professor Mertens von der betriebswirtschaftlichen Fakultät. Später sollte er einer der Gründer der Wirtschaftsinformatik werden. In dem Projekt waren seine Assistenten ergänzt durch ein paar Leute von der HIS. Eine externe Beratungsfirma aus der Schweiz, die B&O, war für die Implementierung zuständig. Diese Firma war an der Entwicklung der normierten Programmierungsmethodik maßgeblich beteiligt und schrieb sie für das Projekt vor. Diese Methode war dazu gedacht, mehrere vorsortierte sequentielle Dateien zusammenzuführen und immer die nächsten Sätze aus jeder Datei auszuwählen. Es gab einen Gruppenwechsel, wenn der nächst höhere Schlüssel vorkam und eine Gruppenendverarbeitung. Das Ganze wurde über GO TO Anweisungen gesteuert [Helm70]. Das konnte mich nicht so begeistern, denn Edger Djikstra hatte gerade seine bahnbrechende Erkenntnis veröffentlicht, dass die GO TO Anweisung schädlich sei [Dijk67]. Davon war jedoch in diesem Projekt nichts zu merken. Der Vorteil der normierten Programmierung war, dass alle Programme den gleichen Aufbau und die gleiche Ablauflogik hatten und das war schon ein Gewinn. Über die vielen GO TO Anweisungen konnte man hinwegsehen. Das habe ich auch getan und mich mit der Methodik abgefunden. Kurz danach habe ich die normierte Programmierung mit der strukturierten Programmierung in meinem ersten deutschsprachigen Fachartikel in der Zeitschrift für Datenverarbeitung verglichen [Sned71].

Technik, Methodik und Organisationsform

Online Zeitschrift für Datenverarbeitung Heft 7, Oktober 1971

Von Harry M. Sneed und Günter Jacobs, Hochschul-Informations-System GmbH

Gliederung:

2.2.1 Das Studenten Operationssystem

Das SOS-Projekt – kurz für Studenten Operationssystem – litt wie alle HIS Projekte unter der Uneinigkeit der Hochschulen. Es sollte ein System für die Verwaltung der Studenten in allen Hochschulen sein mit Immatrikulation, Exmatrikulation und Fortschrittsverfolgung. Da die Hochschulen alle unterschiedlichen Rechner hatten, sollte es zunächst nur auf einem Siemens Rechner mit dem Betriebssystem BS 1000 laufen. Es sollte jedoch so konstruiert sein, dass es später auf die anderen Rechnertypen transportiert werden konnte. Die Programmiersprache war COBOL-60. Damit wurde schon genug verlangt, denn es war damals nicht einfach, ein portables System mit einer so beschränkten Sprache zu bauen. Die verfügbare Speicherkapazität spielte eine ausschlaggebende Rolle. Die Daten wurden deshalb in sequentiellen Dateien auf Magnetbänder gespeichert. Die Schwierigkeit lag darin, die Verarbeitungsschritte in viele aufeinander folgende Batch-Jobschnitte aufzuteilen [Mert72].

Hinzu kamen die unterschiedlichen Anforderungen der beteiligten Hochschulen. Im Prinzip wollte jede Hochschule ein eigenes System haben. Über ein einheitliches Datenmodell konnten sie sich nicht einigen. Von Bundesland zu Bundesland gab es sowohl unterschiedliche Daten als auch unterschiedliche Vorschriften, sprich Geschäftsregeln. Sie auf einen Nenner zu bringen, war wie die Quadratur des Kreises.

25 Jahre später sollte ich mit der gleichen Problematik im Fiskus Projekt konfrontiert sein. Der Föderalismus in Deutschland erschwert die Entwicklung einheitlicher IT-Systeme. Man kann die Systeme noch so parametrisieren und flexibel gestalten, sie passen nicht, solange die Geschäftsprozesse anders ausgelegt sind. So war das auch mit den ersten Projekten für die deutsche Hochschulverwaltung. Das gekoppelt mit der Inkompatibilität der Rechnersysteme schaffte unüberwindbare Hindernisse. Kurzum stellten sich die ursprünglich gesetzten Ziele als unerreichbar heraus.

Um überhaupt weiterzukommen, mussten die Ziele geändert werden. Es sollten nur modellhafte, maßgeschneiderte Systeme für einzelne Hochschulen auf einem bestimmten Rechnertyp realisiert werden. Statt alle Hochschulen mit den gleichen Verwaltungssystemen auszustatten sollte jede Hochschule zusehen, wie sie zu eigenen Systemen mit ähnlichen Funktionen kommen. Die IT-Systeme, die HIS zusammen mit ausgewählten Hochschulen entwickelte, sollten als Muster für die anderen Hochschulen dienen. So wurde das Studenten-Operationssystem letztendlich nach den Anforderungen der Universität Erlangen fachlich ausgerichtet und auf einem Siemens BS 1000-Rechner zum Laufen gebracht. Ich habe das Ende des Projektes nicht mehr miterlebt, da ich inzwischen für ein anderes Projekt abgezogen wurde, aber dank der Einsatzbereitschaft des Erlanger Teams unter der Leitung Professor Mertens, ist das System tatsächlich zwei Jahre später in Betrieb genommen worden.

2.2.2 Das kameralistische Abrechnungssystem

Das nächste Projekt, wozu ich aus Erlangen abberufen wurde, war das Haushaltsverwaltungssystem. Der Auftrag für die Entwicklung dieses Systems wurde an die Universität Köln vergeben, nämlich an den Lehrstuhl des Professor Grochlas. Zuständig für die Durchführung des Projekts war ein gewisser Dr. Strünz, der später Geschäftsführer des MBP Softwarehauses in Dortmund wurde. Damals war er bekannt für seine Veröffentlichungen über die Entscheidungstabelle als Mittel zum Entwurf von EDV-Programmen [Stru72]. Um ihn herum gab es jede Menge Doktoranden, die alle mit Entscheidungstabellen gut umgehen konnten, aber nicht mit COBOL, der vorgesehenen Programmiersprache. Dazu war ich, der Programmierer aus Amerika zuständig. Der Lehrstuhl war wie in Erlangen für Betriebswirtschaft und kein angehender Betriebswirt wollte sich mit Codierung befassen. Diese gleiche Einstellung herrscht

heute noch in der Wirtschaftsinformatik, wo jeder sich als Modellierer empfindet. Das Programmieren, sprich codieren, überlässt man den Indern. Damals gab es noch keine Inder, zumindest nicht am Rhein. Dafür gab es einen Amerikaner. Später kamen auch noch ein pakistanischer Kollege und zwei Studenten der Betriebswirtschaft hinzu. Wir waren die Programmiermannschaft. Es hat deshalb entsprechend lange gedauert bis die ersten COBOL Programme auf der Siemens-BS1000-Maschine der Universität liefen. Die Vorschriften für die öffentliche Haushaltsführung waren schon immer sehr komplex, viel komplexer als die der Studentenverwaltung. Ich war mit vielen neuen Begriffen konfrontiert. Es gab eine Haushaltsaufstellung, einen Haushaltsvollzug und eine Haushaltsabrechnung. Aus der Zeit habe ich gelernt, dass es in der deutschen Verwaltung auch einen Antrag auf einen Antrag gibt. Zu der Haushaltsaufstellung gehörten:

- die Mittelanforderung durch die mittelbewirtschaftenden Stellen
- die Mittelplanung auf der Grundlage der Mittelanforderungen
- die Mittelgenehmigung und
- die Mittelbereitstellung

Zum Haushaltsvollzug gehörten

- die Anordnung von Auszahlungen
- die Entgegennahme von Zahlungen und
- die Führung von Überwachungslisten.

Zur Haushaltsabrechnung gehörten schließlich

- die Sachkontenführung
- die Zeitbuchführung und
- die Jahresabschlüsse.

Da mit dem kammeralistischen Rechnungswesen keine Wirtschaftlichkeitskontrolle möglich war, kam noch eine Kostenrechnung dazu mit

- Kostenartenrechnung
- Kostenstellenrechnung und
- Kostenanalyse [Mund74].

Die letzteren Funktionen waren auf den kammeralistischen Funktionen aufgestülpt. Dies führte zu einer weiteren Steigerung der Komplexität. Neben den HIPO Funktionsbäumen gab es Bäume von verschachtelten Entscheidungstabellen mit Verweise auf die Knoten in den HIPO Bäumen.

All das baute auf einer netzartigen Datenbank mit allen haushaltsrelevanten Daten. Das Datenbanksystem war das SESAM System der Firma Siemens, das gerade von dem späteren Gründer der Software AG – Peter Schnell – für Siemens entwickelt worden war. Es war noch äußerst unstabil, so dass man nicht nur mit der Komplexität der Haushaltsführung, sondern auch mit den Unwägbarkeiten einer unberechenbaren Datenhaltung zu kämpfen hatte. Aus dieser Zeit habe ich gelernt, was die Datenmodellierung für das Projekt bedeutet. Neben dem komplexen Funktionsmodell gab es ein ebenso komplexes Datenmodell, allerdings ohne Entity/Relationship Diagramme, da sie zu diesem Zeitpunkt noch gar nicht erfunden waren. Für solche komplexen Projekte waren weder die Datenbanktechnik noch die Modellierungstechnik ausgereift.

Als Konsequenz hat sich das Projekt in Köln ewig herumgeschleppt. In dem Jahr, in dem ich dort war, sind nur einzelne Probeprogramme fertiggestellt worden. Es würde noch vier Jahre dauern, bis das System einsatzfähig war. Das machte aber nichts aus. Niemand in der Hochschulverwaltung drängte darauf. Im Gegenteil. Sie sahen es eher als eine Bedrohung, die es möglichst abzuwehren galt. Die Nächte in Köln waren lang und im Institut gab es viel zu feiern – Eintritte, Austritte, Promotionen, Geburtstage und Karneval. Dort lernte ich, dass das Arbeitsleben in Deutschland eher ein Ritual ist. Es kommt weniger auf die Leistung als auf die Integration in der Gruppe an – soziale Kompetenz nennt man das heute. Ich ließ mich gut integrieren vor allem mit den Sekretärinnen dort am Institut, und so verging die Zeit zwischen dem Kampf mit der unzulänglichen Technik und dem stressvollen Feiern in den Kölner Kneipen. Am Ende hatte ich genug von beidem und war froh, dass ich nach eineinhalb Jahren und zwei Karnevaljahreszeiten aus Köln abgezogen wurde.

2.2.3 Das Raumbestandsaufnahmesystem

Mein drittes Projekt für die Hochschulverwaltung war die Raumbestandsaufnahme. Dieses Vorhaben war im Gegensatz zu dem Haushaltsverwaltungsprojekt in Köln wohl definiert und überschaubar. Es gab auch einen Termin und einen Anwender – die Universität Karlsruhe, die daran interessiert war, das Ergebnis so bald wie möglich zu bekommen. Hier ging es darum, die Daten über sämtliche Lehrräume der Universität, z. B. die Fläche, die Höhe, die Fenster usw. zu sammeln, zu speichern und auszuwerten. Vorgesehen waren zwei sequentielle Dateien und eine Gebäudedatei und eine Raumbestandsdatei, die über Matrizen im Kernspeicher zu vereinigen waren. Der Kernspeicher war groß genug. Benutzt wurde der für die damalige Zeit Superrechner, der CDC-6600, an der Universität Stuttgart-Vaihingen. Als Programmiersprache sollte die Sprache FORTRAN dienen. Dies kam mir sehr entgegen, denn FORTRAN war neben Englisch meine Muttersprache. Meine ersten Programme beim

Marine Forschungslabor außerhalb von Washington, wo ich programmieren gelernt habe, waren FORTRAN Programme. So gesehen war dieses Projekt für mich ein Heimspiel.

In dem Erlanger Projekt hat es viele Mitstreiter gegeben, auch wenn sie keine Entwickler waren. Die paar Entwickler außer mir kamen von außen. In dem Kölner Projekt war ein ganzer Lehrstuhl an dem Projekt beteiligt. In dem Raumprojekt blieb ich alleine und in diesem Fall war es auch gut so. Über mir gab es einen Raumplanungsspezialisten von der HIS, der vorgab welche Daten zu beheben und welche Berichte zu produzieren sind. Was dazwischen passierte, war mir überlassen. Die erhobenen Daten waren auf Lochkarten, meine FORTRAN Programme ebenso. Einmal fielen sie mir aus den Händen und ich verbrachte eine Nacht damit, sie wieder aufzusammeln und zu sortieren.

Der Rechenbetrieb mit dem großen CD-6000 Rechner an der Uni Stuttgart war sehr streng. Man musste seine Lochkartenkiste an der Pforte abgeben und fand sie zusammen mit den Listen in den Ausgangsfächern, wo jeder Benutzer ein eigenes Fach hatte. Falls es einen Computerfehler gab, wurde der Job abgebrochen und ich bekam eine Diagnostikliste. Falls es einen Laufzeitfehler gab, bekam ich einen Dump – eine dicke Papierrolle mit endlos vielen hexadezimalen Zeichen. Einen Dump zu lesen war für die damaligen Programmierer eine große Herausforderung. Man musste sich mit der Adressierungstechnik auskennen und diese war bei den verschiedenen Computerherstellern immer anders. Im Gegensatz zu den IBM und Siemens Rechnern, die Bytemaschinen waren, war der CDC-Rechner eine Wortmaschine. Die Daten wurden in Wörtern à 60 Bit abgespeichert. Dies war für die Speicherung nummerischer Werte, vor allem Gleitkommazahlen eine gute Einrichtung, aber für die Speicherung von Zeichenfolgen ein Graus. CDC und die anderen Wortmaschinenhersteller hatten auf die nummerische Datenverarbeitung gesetzt. Sie optimierten die Rechenoperationen und die Speicherung in Zahlen auf Kosten der Zeichenverarbeitung. Insofern war es nicht leicht, gewöhnliche Datenverarbeitungsaufgaben mit ihnen zu erledigen.

Zum Glück bestanden die Raumdaten größtenteils aus nummerischen Daten – Flächen, Höhen, Abständen usw. Es ist mir gelungen, sämtliche Raumdaten einer Universität wie Karlsruhe in eine einzige große Kernspeichermatrix unterzubringen, zu vergleichen und auszuwerten. Innerhalb von sechs Monaten war das Projekt Raumdatenerhebung abgeschlossen. Ich musste einige Nächte im Rechenzentrum in Stuttgart-Vaihingen verbringen, um meine Fehler zu beseitigen, aber es war bald überstanden. Dies war das erste HIS-Projekt, dessen Ende ich erleben konnte. Die anderen beiden –

das Studenten-Operationssystem und das Haushaltsverwaltungssystem schleppten sich ewig hin und erreichten erst nach Jahren einen Stand, den man benutzen konnten.

Das Raumerhebungsprojekt war kurz und preiswert. Zwei Mitarbeiter waren sechs Monate damit beschäftigt. Am Ende konnten alle Auswertungen abgeheftet werden. Der Erfolg lag mit der Größe und Überschaubarkeit zusammen. Es gab auch keine Konflikte. Der eine gab vor was zu tun war und der andere hat es ausgeführt. Lehman und Belady würden dies als Wegwerfsystem bezeichnen. Die anderen beiden Systeme waren evolutionäre Systeme bei denen sich die Ziele erst im Verlaufe des Projektes herausstellten. Es waren viele Beteiligte und jeder hatte eine andere Sicht auf die Dinge. Es dauerte schon Jahre, nur einen Konsens zu bilden.

Mir war das eine gute Lehre im Projektmanagement. Man muss zwischen Projekttypen streng unterscheiden: Es gibt planlose und nicht planbare Projekte, technische und politische Projekte. Wo viele Menschen beteiligt sind, überwiegt das Politische über das Technische? Die Probleme liegen weniger an der Implementierung, obwohl das auch schwer genug ist, sondern an der Definition dessen, was zu implementieren ist. Und je länger ein Projekt dauert, desto mehr unterliegt es den Änderungen in der Umgebung, die es selber verändern will. Somit entsteht eine Art Zyklus Virtuos.

Das Raumdatenerhebungsprojekt war mein letztes Feldprojekt bei der H.I.S. Danach wurde ich nach Hannover zurückgeholt um die EDV Ausbildung zu übernehmen. Von da an sollte ich nur noch Kurse für die Anderen organisieren und zum Teil selber halten nach dem Motto: „If you can't do it, teach it." Der Hauptvorteil dieser Tätigkeit war, dass ich einige interessante Leute kennen lernte, u.a. Frau Dr. Heidi Heilmann vom Integrata Institut mit der ich später zusammen Bücher schreiben sollte. Frau Dr. Heilmann brachte ein Jahresband heraus – das Handbuch für Datenverarbeitung – und bat mich um Beiträge [Sned75]. Ich hatte durch meine neue Position mehr Zeit Fachartikel zu schreiben und konnte nebenbei für das erste Buch arbeiten – Informationssysteme für Hochschulen.

2.3 Veröffentlichungen bei der H.I.S.

Schon im Frühjahr 1970 habe ich zusammen mit zwei Kollegen von der H.I.S. einen Artikel für die Zeitschrift für Datenverarbeitung mit dem Titel „Aufbau eines Informationssystems für Hochschulen" verfasst [Mund75]. Es war nicht nur mein erster Fachartikel in deutscher Sprache, sondern meine erste Veröffentlichung überhaupt. Der Artikel wurde sehr lange und dehnte sich über drei Ausgaben der Zeitschrift hinaus. Darin stellten wir ein DV-System vor, das alle Bereiche der Hochschulverwal-

tung abdecken sollte - Studentenoperationen wie Anmelden, Abmelden, Ausstellen von Bescheinigungen, usw., Personalabrechnung, Haushaltswesen, Kassenwesen, Raumverwaltung und vieles mehr. Das sollte alles von einem zentralen Rechner aus gesteuert werden. Ich habe beschrieben wie dieses allumfassende Management-Informationssystem zu implementieren wäre, natürlich nur rein theoretisch. Wie so ein komplexes System in Wirklichkeit zu entwickeln wäre konnte ich damals nicht ahnen. Später haben wir drei entschieden ein Buch darüber zu schreiben. Dieses erste Buchprojekt hat sich lange hingezogen. Als das Buch endlich im Jahre 1974 im deGruyter Verlag erschien war ich schon lange nicht mehr bei der H.I.S.

Das Buch war um einiges detaillierte als der erste Artikel war, nicht nur weil es länger war, sondern weil wir in der Zwischenzeit viel mehr Erfahrung gesammelt hatten. Ich hatte persönlich mitgewirkt an der Entwicklung des Studenten-Operationssystems, an dem Finanzverwaltungssystem, an dem Personalverwaltungssystem und zuletzt an der Raumverwaltung. Ich konnte daher zumindest meine Teile des Buches mit vielen Berichten aus der Entwicklungspraxis untermauern. Neben den diversen Anwendungssystemen habe ich ein allumfassendes, integriertes Datenmodell für die Hochschule vorgestellt. Die Mindest-Hardware Konfiguration habe ich auch vorgegeben, obwohl das nicht gerade mein Gebiet war. Das Schwierige war, dass sowohl die Anwendungssoftware als auch das Datenmodell auf alle damals gängige Rechner übertragbar sein sollte. Das war zu dieser Zeit ein großer Wurf und konnte nur gelingen, wenn die Systemarchitektur möglichst dehnbar und die Systemkomponente möglichst austauschbar blieben. Das Buch ist gut gelungen, aber leider haben sich nur wenige für dieses Thema interessiert.

Inzwischen hatte ich begonnen mich mehr technischen Themen zu widmen. Mit dem H.I.S. Entwicklungsleiter zusammen, habe ich Ende 1971 einen Fachbeitrag zum Thema „Modularprogrammierung“ für die Zeitschrift für Datenverarbeitung verfasst [Sned71]. Mein Anliegen war die großen, monolithischen COBOL Programme von damals nicht nur in einzelnen internen Prozeduren die per Perform-Anweisung aufgerufen werden, sondern darüber hinaus die Programme in möglichst viele, getrennt compilierbare Module aufzuteilen, die per Call-Anweisung aufgerufen werden. Es sollte dabei darauf geachtet werden, dass möglichst viele Module in verschiedenen Programmen wiederverwendet werden. Ich befürwortete sogar eine gemeinsame Bibliothek wiederverwendbarer Softwarebausteine. Dies war das erste Mal, dass der Begriff Softwarebaustein vorgekommen ist. Derartige Modulbildung war zu dem Zeitpunkt ein innovativer Ansatz und wurde in der Fachpresse oft zitiert.

Danach folgten weitere Artikel über den Aufbau großer Programme in der gleichen Zeitschrift. Ich zitierte Edgar Dijkstra und setzte mich für die strukturierte Programmierung ein. Mit Beispielen habe ich gezeigt wie man zumindest in COBOL die GOTO Anweisung vermeiden konnte [Sned73]. In der gleichen Zeitschriftausgabe erschien der erste Artikel in deutscher Sprache über Struktogramme als Entwurfstechnik von Peter Schnupp und Christianne Floyd von der Firma SoftLab. Zu diesem frühen Zeitpunkt habe ich auch begonnen mich mit dem Thema Test auseinander zu setzen. Ich hatte erkannte wie wichtig es war der Test vom Anfang an zu berücksichtigen. Später als ich bei Siemens war, setzte ich meine Berichtserstattung mit Beiträgen über Programmier-Management und Programmiersprachen fort. Ich hatte schon mit dem nächsten Buch angefangen – Strukturierte Programmierung in der Praxis. Meine Karriere als Fachspezialist für Softwareentwicklung war auf einem guten Wege. Aufgrund meiner vielen Fachartikel wurde ich eingeladen öffentliche Seminare zum Thema „Programmierung und Programmmier-Management“ zu halten. Deutschland stand damals erst an der Schwelle zur Software-Engineering.

2.4 Das Kommunale Verwaltungsprojekt in Göttingen

Ende 1973 ist die Aufgabe als Oberlehrer bei H.I.S mir doch zu langweilig geworden. Ich sehnte mich wieder nach der Projektarbeit. Also habe ich bei der H.I.S. gekündigt und bin zu der Siemens Geschäftsstelle in Hannover übergegangen. Ich hatte in meiner Tätigkeit als Lehrer bei H.I.S einen Mitarbeiter von Siemens kennen gelernt der mir erzählte Siemens würde nach Projektleiter für Projekte der öffentlichen Hand in Niedersachsen suchen. Ich bekam sofort ein Projekt und zwar bei der kommunalen Verwaltung in Göttingen. Ich müsste zwar jeden Tag von meinem Zuhause bei Lehrte mit der Bahn nach Göttingen reisen aber das schien mir das geringste Übel zu sein. Im Zug konnte ich die Zeit nutzen neue Fachartikel zu schreiben.

Das System für die kommunale Verwaltung in Göttingen habe ich so gut wie alleine ohne fremde Hilfe geschaffen. Es gab von der Stadtverwaltung einen Projektleiter der mir sagen konnte was ich machen sollte und zwei junge Entwickler die mir zugearbeitet habe. Heute würde man dieses Projekt als agile bezeichnen. Ich habe das Ganze in Pakete aufgeteilt und ein Paket nach dem Anderen in COBOL auf der Siemens BS1000-Anlage implementiert. Die Zeit dafür war sehr knapp, das System musste spätestens im März des Jahres 1974 in Produktion gehen und ich habe erst im September 73 mit der Implementierung begonnen. Als ich begann hatte das System nur einen Namen – KOFAS für Kommunales Finanzsystem - und einige Anforderungsdokumente sowie ein halbfertiges Datenmodell. Ich habe mit dem Datenmodell begonnen und es zu Ende gemacht. Währenddessen habe ich meine zwei Hilfsent-

wickler zu den Fachabteilungen geschickt um die Verarbeitungsregel mit Entscheidungstabellen zu dokumentieren.

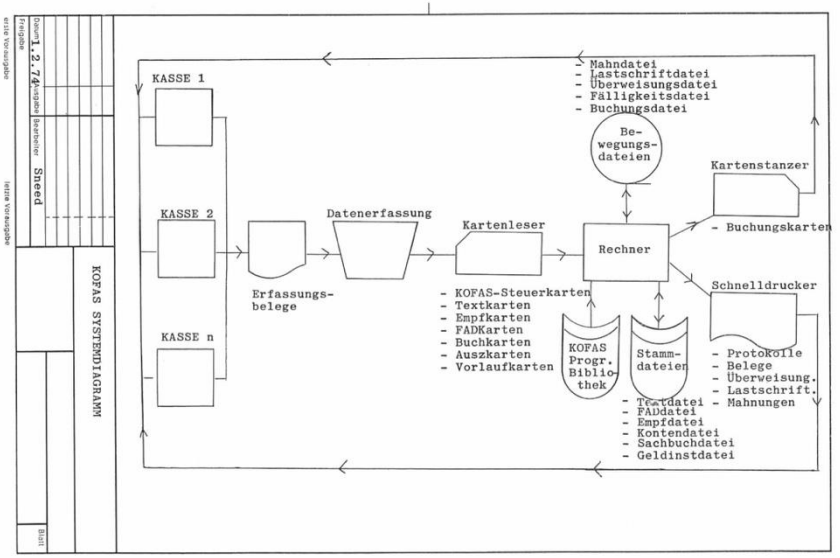


Abbildung 2 KOFAS Systemdiagramm

464

Es sollte sechs Stammdateien geben:

- Eine Textdatei mit Textbausteine für die Aussendungen
- Eine Adressdatei mit den Adressen aller Bürger
- Eine Bankdatei mit den Bankdaten aller Bürger
- Eine Sachbuchbestandsdatei mit einem Bestandssatz für jedes Sachbuchkonto
- Eine Geldbestandsdatei mit einem Bestandssatz pro Geldinstitut
- Eine Kontendatei einem Hauptsatz für jede Buchungsstelle und einem untergeordneten Satz für jede Buchung.

Dazu kamen fünf Bewegungsdateien:

- Eine Buchungsdatei mit einem Satz pro Einnahme-, bzw. Ausgabebuchung
- Eine Fälligkeitsdatei mit einem Satz für jede Zahlungsrate
- Eine Überweisungsdatei mit einem Satz für jede fällige Ausgabe
- Eine Lastschriftdatei mit einem Satz für jede fällige Einnahme
- Eine Mahnungsdatei mit einem Satz für jede überfällige Zahlung.

Die Bewegungsdateien waren reine sequentielle Dateien auf Magnetbändern. Sie wurden mit Lochkarten erstellt, die in den Fachabteilungen erfasst worden. Die Stammdateien waren index-sequentielle Dateien auf dem Plattenspeicher. Sie lagen im direkten Zugriff über mehrere alternative Suchmerkmale und wurden durch die Bewegungsdateien regelmäßig aktualisiert.

Auf der Funktionsseite gab es zehn Batch-Transaktionen. Heute würde man sie als Anwendungsfälle bezeichnen.

- Einrichtung der Stammdateien
- Fortschreibung der Text- und Anschriftendateien
- Korrektur der Buchungs- und Fälligkeitsdaten
- Abfrage der Kontendatei
- Bereitstellung der Kontoauszüge
- Durchführung des Lastschriftverfahren
- Durchführung des Mahnverfahrens
- Bereitstellung des Sachbuches
- Monats- und Jahresabschluss
- Sicherung der Stammdaten

In diesen Transaktionen wurden 17 verschiedene Listen erzeugt.

Diese Funktionen wurden durch 27 COBOL Hauptprogramme und 26 Unterprogramme sowie drei JCL Dienstprogramme – Sort, Merge und Split – abgedeckt. Die COBOL Programmen umfassten genau 13.606 Anweisungen.

Es lagen knapp 17 Arbeitswochen zwischen dem Anfang des Projektes im September 1973 und dem Ende des Projektes am 10. Feb. 1974.

- Die Systemanalyse und das Systemdesign dauerten 5 Wochen.
- Die Programmierung dauerte 5 Wochen
- Der Test nahm 6 Wochen in Anspruch.
- Zum Schluss gab es noch eine für die Inbetriebnahme des Systems.

Bezogen auf den Codeumfang von 13.606 COBOL Anweisungen ergab sich bei 390 gebuchten Personentagen (über 100 für meine Person) für Design, Kodierung und Test eine Produktivität von 35 Anweisungen pro Personentag, bzw. 700 Anweisungen pro Personenmonat. Siemens verlangte für mich damals DM 500,- pro Tag. Für die internen Mitarbeiter wurden DM 3000,- pro Monat berechnet. Die Personalkosten

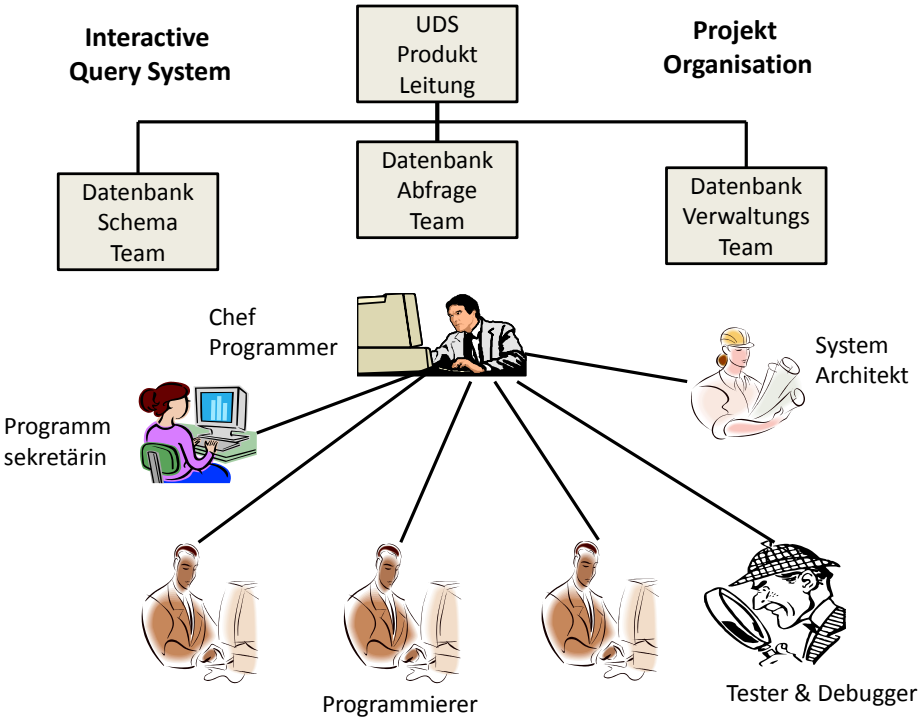
kamen insgesamt auf DM 152.000. Das machte knapp über DM 11,- pro Code-Anweisung. Eigentlich hatte der Kunde jeden Grund zufrieden zu sein. Der Termin wurde eingehalten, das System erfüllte die Erwartungen und die Kosten lagen im Rahmen des geplanten Budgets. Außerdem war das System mit strukturierten Diagrammen gut dokumentiert. Die zwei internen Entwickler konnten das System übernehmen und weiterführen. Die Stadtverwaltung bedankte sich mit einem schönen Bilderbuch über die Stadt Göttingen. Zuletzt habe ich zusammen mit dem Projektverantwortlichen von der fachlichen Seite aus einen Fachartikel über das Projekt verfasst [Walt74]. Schon im Februar 1974 trat ich meinen neuen Posten in München an. Siemens wollte mich dort in der Datenbankentwicklung haben.

2.5 Projekte bei Siemens in München

2.5.1 Das Interaktive Datenbank Query System - IQS

In der ersten Hälfte der 70er Jahren war Siemens gerade dabei ein neues universales Datenbanksystem – UDS – zu entwickeln. An diesem Mammutprojekt waren viele Teams in verschiedenen Dienststellen beteiligt. Das ganze Produkt war in mehrere Teilprodukte aufgeteilt. Jedes Team war für ein Teilprodukt zuständig. Mein Team war für den Datenbankabfragedienst zuständig. Dazu gehörte eine Abfragesprache, angelehnt an die IQF Abfragesprache der IBM, eine Abfrageschnittstelle und eine Reihe Komponente um die Abfragen zu bearbeiten. Die Komponente waren in SPL zu schreiben – eine abgemagerte PL/I Sprache mit Assembler Elementen.

Wir waren in unserem Team zu Dritt – ein langjähriger Assembler Programmierer, ein ehemaliger Telefunken Mitarbeiter und ich. Später kam noch einen Praktikanten von der TU München dazu. Zwischendurch gab es einen Austauschmitarbeiter von Philips aus Schweden. Damals war Philips und Bull Partner von Siemens in einem gemeinsamen europäischen IT-Konsortium - UniData. Ich wurde zum Teamleiter ernannt, wahrscheinlich wegen meiner englischen Sprachkenntnisse. Englisch war die Sprache des europäischen Konsortiums. So viel zum Hintergrund dieses Projektes, das zwei Jahre von Frühjahr 74 bis Frühjahr 76 gedauert hat. Am Ende dieser Zeit habe ich für das Jahrbuch der Frau Heilmann einen Artikel darüber geschrieben [Sned76].



Das IQS Projekt war das erste Mal für mich in der IT andere Menschen anzuleiten. Bei H.I.S. war ich selbst nur Projektmitarbeiter. In Göttingen hatten die Stadtangestellten mir nur zugearbeitet. Den überwiegenden Anteil der Arbeit habe ich selber geleistet. Mir war die Rolle als Teamleiter nicht ganz fremd, in der Armee bin ich Sargent gewesen und genoss eine Ausbildung als Unteroffizier. Ich wusste also schon wie man andere Menschen führen sollte. Ich entdeckte aber bald, dass es in der Softwareentwicklung ganz anders zugeht. Man kann nicht einfach befehlen, etwas zu tun, die Untergebenen müssen es von sich aus tun wollen. Die Kunst dabei ist sie dazu zu bringen, es tun zu wollen. Es beginnt damit, dass jeder einen eigenen Arbeitsbereich hat der zu seinen Fähigkeiten passt – nicht zu schwer und auch nicht zu leicht. Das allein ist keine einfache Aufgabe. Hinzu kommt, dass der Teamleiter die Schnittstellen zwischen den Bereichen wohl definiert und selber programmiert. Über die Schnittstellen steuert der Leiter die Teamarbeit. Damit stellt er den Rahmen bereit in dem die Anderen werken können. Ebenso wichtig war es, den Stand der Arbeit, bzw. den SPL Code, regelmäßig, mindestens einmal die Woche zu kontrollieren. Dies habe ich im Gespräch mit den Mitarbeitern getan. Jede zweite Woche hatten wir ein großes

Team-Meeting bei dem wir die bisherigen Ergebnisse zur Diskussion stellen und bei dem wir die nächsten Schritte geplant haben. Jedes Team-Mitglied müsste zu seinen Modulen ein Struktogramm und ein Datenflussdiagramm zeichnen und einen kommentierten Code liefern mit Datenerläuterungen und Prozedurköpfen. Jeder führte einen Projektordner mit seinem Code und seinen Dokumenten. Eigentlich hatten wir von der agilen Entwicklung vieles vorweggenommen – kurze Release-Intervalle, regelmäßige Berichte durch die Mitarbeiter, gegenseitige Kontrollen, usw. Das hat alles recht gut funktioniert und wir kamen schnell voran bis wir auf den Test stießen.

Mein Vorbild damals war das Chief-Programmer Teamkonzept von Harlan Mills [Mills72]. Ich habe mich selbst in der Rolle des Chefprogrammier gesehen. Ich sollte den Weg für die Anderen bahnen und kontrollieren, dass sie auf dem richtigen Weg auch blieben. Leider konnte ich nicht alles umsetzen was ich mir vorgenommen habe. Einen Assistenz-Programmierer, nämlich einen Architekten, habe ich gehabt. Das war der eine Schwede mit den tiefsten Kenntnissen von SPL. Ohne seinen Rat hätte ich manche Fehlentscheidung getroffen. Was fehlte war ein Tester im Team, um die abgelieferten Komponenten zu testen sowie ein Projektsekretär, jemand der die Dokumentation formatiert und aktualisiert, die Besprechungen protokolliert und Buch führt über das Projekt – die abgelieferten Ergebnisse, die offenen Fehlern, die anstehenden Aufgaben, usw. Vieles was heute unter Agile Entwicklung verstanden wird, haben wir praktiziert, aber wir hatten nicht das Personal um alle Rollen zu erfüllen. Die Rollen des Testers und des Projektsekretärs, bzw. Trackers, blieb aus und ich müsste zugleich Produkt-Owner, Tester und Leitprogrammierer spielen.

Da schon die ersten Change Requests gekommen sind, habe ich ein Tool entwickelt um sie zu registrieren, priorisieren und zuzuteilen. Darüber hinaus hatte ich den BS2000 Editor erweitert um die Änderungen in den Code zu platzieren und einen Trace-Pfad zu hinterlassen. Der schwedische Architekt hat mir damit geholfen. Im Rahmen dieses Projektes habe ich also schon begonnen mich mit der Softwarewartung zu befassen. Ich musste lernen, dass Software nie fertig ist.

Immerhin konnte das Produkt erfolgreich abgenommen werden und 1976 habe ich darüber einen Beitrag darüber in dem Integrata Handbuch der EDV veröffentlicht mit dem Titel „Planung, Organisation und Steuerung der Softwareherstellung und –wartung“. In diesem Beitrag ging ich auf die Rollen und Aufgaben eines Programmierteams ein.

- Chefprogrammierer = Product Owner und Leitprogrammierer
- Assistenzprogrammierer = Systemarchitekt
- Programmsekretär = Project Tracker

- Programmierer.

Was ich darin weggelassen habe war die Rolle des Testers. Ich habe bald gemerkt, dass die Programmierer sich schwer taten mit dem Test und die meiste Zeit mit Testen verbracht haben. Erst gegen Ende des Projektes begann ich einen Testrahmen zu entwickeln mit Testtreiber, Teststubs und Testmonitor, aber es war eigentlich zu spät. Die meisten Module waren schon in Produktion. Das war jedoch für mich eine Lehre für die Zukunft. Fortan würde ich den Test in den Vordergrund stellen.

Mein Beitrag zum Integrata Handbuch endete mit einem Plädoyer für mehr Disziplin in der Softwareentwicklung:

„Lange Zeit galt die Programmierung als exzentrische Kunst und der Programmierer als freischaffender Künstler. darin steckt immer noch ein Kern Wahrheit. die Ideenfindung beim Entwurf eines komplexen DV-Systems sowie die Lösung verknoteter Kodierungsprobleme ist sicherlich ein Kreativer Akt. Ohne kreative Programmierer wird keine Software-Abteilung Erfolg haben. Man braucht ständig neue Ideen um voranzukommen.

Andererseits braucht eine Software-Abteilung Disziplin. Ohne Disziplin wird die Kreativität der Programmierer vergeudet. Sie werden ihrem Hang zum Spiel nachgeben, ihre Aufgaben selber definieren und sich immer weiter von den wahren Bedürfnissen der Anwender entfernen. Die Programmierung wird damit zum Selbstzweck.

Um diesen Zustand zu verhindern, ist eine aktive und fähige technische Führung erforderlich, die in der Lage ist Entwicklungs- und Wartungsprojekte zu planen, zu organisieren und zu steuern. Kreativität genügt nicht. Sie muss durch konkrete Ziele und strenge Kontrollen kanalisiert werden. Das gewünschte Ergebnis soll eine Synthese von Kreativität und Disziplin sein. Dies ist der Schlüssel zum Erfolg in der Herstellung und Wartung komplexer Softwareprodukte…“ [Sned77].

2.5.2 Mein erstes Reengineering Projekt

Im Leben eines Menschen gibt es bestimmte Schlüsselereignisse, die sein Leben in die eine oder die andere Bahn lenken. Bei mir waren es drei die dicht aufeinander folgten. Die erste war eine neue für mich absolut fremde Aufgabe. Das IQS Projekt, über das im letzten Kapital berichtet wurde, war abgeschlossen. Ich sollte beginnen mit einem Projekt zur Sanierung der SESAM Datenbanksystemsteuerung, mein erstes Reengineering Projekt. Das Steuerungsmodul zu diesem quasi-relationalen Datenbanksystem, das aus der gleichen Quelle wie das spätere Datenbanksystem ADABAS

entsprang, war monolithisch und total verknotet, ein richtiger Spaghetti Code im wahrsten Sinne des Wortes. Er ist wie viele alte Softwaresysteme planlos durch Trial und Error entstanden. Meine Aufgabe war es, den Code schrittweise in eine neue, bessere Struktur zu überführen. Meine Erfahrung mit Assembler war zu diesem Zeitpunkt begrenzt, aber sie reichte aus, um den Code zu lesen und auseinander zu nehmen. Ich folgte dem klassischen Restrukturierungsansatz. Zuerst versuchte ich mir einen Überblick über die einzelnen Codebausteine und deren Abhängigkeiten zu verschaffen. Dafür habe ich verschiedene Strukturdiagramme verwendet – Baumdiagramme, Ablaufdiagramme und Datenflussdiagramme. Das Endresultat war ein Istmodell des Steuerungsmoduls. Ohne es explizit zu wissen, hatte ich mein erstes Reverse Engineering Projekt gemacht, in dem ich einen Plan aus dem Product gewonnen habe. All das hat schon drei Monate gedauert, die rund 16.000 Assembler Codezeilen nach zu dokumentieren. Ich musste alles manuell erstellen, denn Reverse Engineering Werkzeuge gab es damals keine. Erst recht nicht für die Assemblersprache. Zum Schluss war jeder Codeblock gekennzeichnet und einem Register zugeordnet.

Nachdem das Ist-Modell des Assembler Moduls endlich stand, begann ich ein Sollmodell zu konstruieren. Während das Ist-Modell bottom-up entstanden ist, sozusagen von den einzelnen Codezeilen über die prozeduralen Bausteine und Datenstrukturen hinauf bis zur gesamten Architektur, ging es mit dem Sollmodell top-down vom Konzept bis zum einzelnen Assembler Befehl vor. Am Anfang wurden die Hauptfunktionen des Steuerungsmoduls identifiziert und in einen Funktionsbaum dokumentiert. Der Steuerungsablauf wurde in einem Riesenkontrollflussgraph abgebildet. Daneben entstand ein grobes Datenflussdiagramm. Schließlich wurden die Datenblöcke als Entitäten in einem Entity/Relationship Diagramm dargestellt. Struktogramme, bzw. Nassi-Sheiderman Diagramme für die Modellierung der Ablaufstruktur sowie E/R Diagramme für die Modellierung der Datenstruktur waren gerade erfunden worden und es reizte mich, diese neuen Technologien anzuwenden. Ich war zu diesem Zeitpunkt auch dabei, ein Buch über strukturierte Programmierung zu schreiben. Ich hielt schon Seminare darüber und so lag es nahe, jene Strukturierungstechniken für die Nachdokumentation des Assembler Programmes anzuwenden. (siehe Sesam Struktogramm)

Nachdem das Sollmodell zur SESAM Steuerung stand, ging es über zur nächsten Phase – die Implementierung. Die eingegebenen Codeblöcke mussten aus der alten Spaghetti-Struktur herausgelöst und in die neue modulare, hierarchische Struktur versetzt werden. Dies stellte sich als eine besonders schwierige Aufgabe heraus. Nicht nur, dass es langwierig war, es war auch fehleranfällig. Ich musste einen Festrahmen für das neue Assemblerprogramm aufbauen und nach dem Anfügen eines jeden Codeblocks den Zusammenhang testen. Wie bei der IQS Entwicklung nahm der Test

den größten Teil der Zeit in Anspruch. Da ich alleine an dieser Aufgabe gearbeitet habe, wurde es nach einer Weile für mich sehr langweilig. Ich begann zu zweifeln, ob ich mit dieser Aufgabe je fertig sein würde. Zu Beginn des Jahres 1977 war ich noch Angestellter bei der Firma Siemens und hatte keine Absicht diesen Status zu verändern.

Um diese Zeit, im Frühjahr 1977 folgte das zweite große Ereignis – die Abwerbung. Ich lernte den Herrn H. kennen. H. war ein drahtiger, dynamischer Manager, so wie man sich einen Projektleiter vorstellt – lean and mean. Er raste von einem Termin zum anderen und sprach so schnell wie er zu entscheiden pflegte – also kein Bayer sondern ein „Zuagroasta“. H. hatte von mir gehört und wollte mich für sein neues Projekt in dem damaligen Hauptwerk gewinnen. Siemens hatte einen Großauftrag von der Bundesbahn erhalten. Sie sollte ein komplexes System für die Überwachung und Steuerung der Frachtzüge entwickeln – die integrierte Transportsteuerung – ITS. Daran sollten mehr als 100 Softwareentwickler beteiligt sein. Für die damalige Zeit ein riesiges Projekt. H. wusste von meinem Interesse in Qualitätssicherung und wollte mich für sein Testteam gewinnen.

Artikel über den Test der Datenbank Software bei Siemens 1977

Systematisches Programmtesten – ein mühsames Geschäft

Systematisches Programmtesten, ein mühsames Geschäft

Von Harry M. Sneed

Eine der größten Illusionen der Software-Entwicklung ist der Glaube, man könne ohne allzuviel Testaufwand auskommen. Aufgrund dieses Irrtums wird bei der Software-Projektplanung oft zu wenig Zeit für das Testen vorgesehen. Dies hat eine von zwei möglichen Konsequenzen für das Projekt: Entweder werden die Programme ungetestet freigegeben, um den Termin auf jeden Preis zu halten, oder der Fertigstellungstermin wird überzogen. Beide Konsequenzen fallen dem Benutzer zur Last, die eine früher, die andere später.

17:32

Online – Zeitschrift für Datenverarbeitung Sept. 1977

Sein Angebot kam mir gerade recht. Die Aufgabe erschien mir als eine interessante Herausforderung und als eine Möglichkeit, meine damalige Vision, ein automatisiertes Testwerkzeug zu bauen. Ich hatte von der mühseligen Restrukturierung des Assembler Codes schon genug und brannte darauf, eine neue aufregendere Aufgabe anzupacken. Es gab leider einen Haken. Mein Abteilungsleiter in der Datenbankentwicklung war nicht bereit, mich ziehen zu lassen. Er wollte mich bei sich behalten. Außerdem sollte ich das SESAM Reengineering Projekt zum Abschluss bringen. Mein Versetzungsantrag wurde abgelehnt. H. hat aber nicht locker gelassen. Er sah eine andere Möglichkeit vor. Ich sollte als Angestellter kündigen und bei ihm als freier Mitarbeiter einsteigen. Die Idee klang nicht schlecht. Ich hatte ohnehin schon meine Seminartätigkeit als freischaffender Referent, warum nicht auch als freischaffender Tester arbeiten?

Dagegen sprach, dass meine Frau gerade mit unserem ersten Kind schwanger war, einen Zustand, auf den wir schon lange gewartet hatten. Gerade jetzt zu diesem Zeitpunkt meinen sicheren Angestelltenstatus bei Siemens für ein Leben als Freiberufli-

cher in einem vorübergehenden Projekt aufzugeben, erschien mir gewagt zu sein. Nichtsdestotrotz habe ich mich entschieden das zu tun. Die Herausforderung ließ mich nicht locker und meine Frau war damit einverstanden. Vielleicht wäre es auch eine Möglichkeit, mehr zu verdienen und früher zu einem eigenen Heim zu kommen. Also heuerte ich bei H. für 50 DM die Stunde an. Das kam mir damals wie viel Geld vor, aber später stellte sich heraus, dass die anderen Freiberufler alle mehr verdienten. Ich war in dieser Hinsicht immer zu bescheiden. Es fehlt bei mir der Geschäftssinn. Mich interessierte nur die Aufgabe.

2.5.3 Das Integrierte Transport Steuerungssystem – I.T.S.

Das ITS Projekt fand in Baracken auf dem Gelände des Hauptwerkes in der Hoffmannstraße statt. In den regulären Gebäuden gab es keinen Platz mehr für so viele Softwareentwickler. Außerdem sollte das Projekt in zwei Jahren fertig sein. Danach sollten die Projektmitarbeiter wieder dorthin zurückkehren wo sie hergekommen sind. Viele kamen von anderen Siemens-Abteilungen, andere von den neuen Software-Häusern, die im Zuge der IT-Expansion wie Pilze aus dem Boden schossen. Für mich gab es keine Rückkehr, da ich gekündigt hatte. Darüber machte ich mir aber wenig Gedanken, es würde schon etwas Anderes geben. Dass es weit weg in einer anderen Welt sein sollte, konnte ich damals nicht ahnen.

Das I.T.S. Projekt war zu der Zeit das größte Softwareprojekt in der Bundesrepublik. H. erzählte gerne darüber wie er das Projekt schichtenweise, entsprechend der Systemarchitektur, organisiert hatte. Damals wurde in der Softwarewelt die These von Conway oft zitiert – Conway's Law, dass die Architektur der Software ein Spiegelbild der Organisation ist, welche die Software entwickelt [Conw68]. Es hieß, zeig mir ein Organigramm ihrer Projektorganisation und ich werde wissen wie ihre Software aufgestellt ist. H. wollte es umgekehrt machen. Die Organisation sollte nach der Softwarearchitektur ausgerichtet sein. Um H. herum gab es einige junge Systemberater von der TU-München, die später eigene Softwarefirmen gründen würden, u.a. die Herrn Dehnert und Maiborn. Sie haben sich das Schichtenmodell ausgedacht.

Das I.T.S. System soll in drei Schichten gegliedert sein:

- eine anwendernahe Schicht
- eine Umsetzungsschicht
- eine betriebssystemnahe Schicht.

Die anwendernahe Schicht entsprach dem, was man heute als Frontend bezeichnet. Sie umfasste die Bildschirmbedienung, die Plausibilitätsprüfung, die Aufbereitung der Eingabedaten und die Bereitstellung der Ausgaben. Für die Kommunikation mit

den Bahnbediensteten waren mehr als 3000 Triumph Adler Bildschirmgeräte vorgesehen. Sie standen schon in einem Lagerhaus in Nürnberg bereit. Für diese Schicht gab es ein Team von 20 bis 30 Entwicklern mit Kenntnissen der damaligen Online-Oberflächentechnik.

Die betriebssystemnahe Schicht war für die Verwaltung der Systemressourcen gedacht. Hier fanden die Datenbankzugriffe statt, aber nicht nur das. Diese Schicht war auch für die Speicherverwaltung und die Nachrichtenübermittlung zuständig. Heute würde man diese Schicht als Serviceschicht bezeichnen. Alles was das System an Ressourcen – Daten, Speicher, Kommunikationsanschlüsse usw. - brauchte, kam über die betriebssystemnahe Schicht. Diese Schicht war, wie die Bezeichnung impliziert, eng mit dem neuen Betriebssystem BS-2000 verknüpft. Sie wurde in BS-2000 Assembler von mehr als 30 Betriebssystemspezialisten implementiert. Da man der Performance der damaligen Datenbanksysteme nicht traute, wurde ein eigenes Dateiverwaltungssystem auf ISAM Basis gestrickt.

Die Umsetzungsschicht bildete den Kern des Systems. Hier war die eigentliche Geschäftslogik angesiedelt. Die größere Mannschaft mit rund 40 Entwicklern war für diese Schicht zuständig. Sie wurde mit der Systemprogrammierungssprache SPL eine Mischung aus PL/I und Assembler gebaut. Interessanterweise wurde diese Schicht wiederum in drei Schichten unterteilt:

- eine Steuerungsschicht
- eine Verarbeitungsschicht und
- eine Dienstleistungsschicht.

Die Steuerungsschicht erhielt die Eingabenachrichten von dem Frontend, der anwendernahen Schicht. Die Dienstleistungsschicht stellte die Aufträge in die betriebssystemnahe Schicht. Die Verwaltungsschicht setzte die Daten nach den von der Bahn vorgegebenen Geschäftsregeln um. Damals gab es allerding von Anfang an große Schwierigkeiten, denn die von der Bahn gelieferten Spezifikationen waren absolut unzulänglich. Sie bestanden aus wirren Prosatexten. Die Siemens Entwickler in München müssten immer wieder nach Frankfurt reisen um die Vorgaben mit den zuständigen Bahnspezialisten zu klären. Später in den 80er Jahren würde ich Jahre damit verbringen, den Bahnmitarbeitern beizubringen, wie man Anforderungen ordentlich spezifiziert.

Es gab noch ein weiteres Problem. Jede Schicht sollte nur mit der nächst darunterliegenden Schicht über Parameterschnittstellen kommunizieren. Die Schnittstellen waren sehr exakt als zusammenhängende Datenstrukturen spezifiziert, so wie heute

XML Schnittstellen. Das aufrufende Modul musste sämtliche Daten für die Übergabe erst sammeln und in die Schnittstellenstruktur einpacken ehe er sie an die nächste Schicht weitergeben konnte. Das gleiche passierte mit den Ergebnissen, die zurückkamen. Sie mussten aus der Schnittstelle ausgepackt und wieder an ihre ursprünglichen Orte verteilt werden. Diese Kapselungstechnik war theoretisch schon mal gut, passte aber überhaupt nicht zu der damaligen Rechnerkapazität. Der BS 2000 Rechner war voll ausgelastet damit, Daten nur von einem Ort zum anderen hin- und herzuschieben. Hinzu kam, dass das BS 2000 Betriebssystem damals in den Kinderschuhen steckte und keineswegs in der Lage war, ein so komplexes Anwendungssystem adäquat zu unterstützen. Nicht nur von der Performance her war es unzulänglich, auch was die Zuverlässigkeit anbetraf, ließ es zu wünschen übrig. Systemabbrüche waren an der Tagesordnung.

Um den Schnittstellenengpass zu umgehen sind die Entwickler dazu übergegangen einen globalen Datenbereich für die Kommunikation zwischen den Schichten zu nutzen. Die Entwickler der einen Schicht haben dort Daten an bestimmten vereinbarten Stellen abgelegt damit die Entwickler von der anderen Schicht sie abholen konnte. Meistens war das informal in den Kaffeepausen bilateral abgesprochen wo welches Bit stehen sollte. Zum Schluss wusste keiner mehr Bescheid wo was stand. Keiner hatte es dokumentiert. Da der globale Speicher beschränkt war, haben die Entwickler nicht selten die gleichen Speicherstellen für verschiedene Daten verwendet. Mit dieser Lösung haben die Entwickler den Teufel mit dem Bezelbub ausgetrieben. Der ursprüngliche Globalbereich war für lediglich 64 K Bytes vorgesehen. Am Ende war es auf 512 Kilo Bytes angewachsen und passte kaum noch in den Kernspeicher. Dieser Missbrauch der Common Daten führte zu unendlich vielen Fehlern und war auch ein Grund für das Scheitern des Projekts.

Also war dieses Projekt von Anfang an zum Scheitern verurteilt, aber es beschäftigte mehr als 100 Mitarbeiter für drei Jahre und kostete der Bahn wie die Zeitschrift „Der Spiegel“ später berichtete, mehr als 20 Millionen DM, eine stattliche Summe für die damalige Zeit. Die 3000 Triumph Adler Bildschirme sind nie aus dem Lagerhaus gekommen und wurden später verschrottet. Die Software wurde auf mehrere hundert Bänder abgezogen und archiviert. Dies war die größte Pleite in der bisherigen jungen deutschen IT-Industrie, obwohl die besten Köpfe des neuen Informatikfaches daran maßgeblich beteiligt waren oder vielleicht gerade deshalb. Eine Simulation mit einem Systemprototyp hätte die Engpässe in dem Nachrichtendurchlauf zum Vorschein gebracht und auch die Schwächen des Betriebssystems aufgezeigt. Aber das wollte keiner wirklich wissen. Jeder war zuversichtlich, es könnten alle Probleme mit den richtigen Leuten beseitigt werden. Die Schwächen der fachlichen Spezifikation hätte man

mit einfachen Review-Techniken feststellen können. Man brauchte sie nicht zu codieren, um sie erst beim Testen zu entdecken. Dennoch hatte das ITS Projekt auch seine positive Seite. Es war ein gigantisches Ausbildungsprojekt und die Mutter zahlreicher späterer erfolgreicher Softwarehäuser, wie Softlab und SD&M. Auch mein Start als Softwareunternehmer habe ich dem ITS Projekt zu verdanken. Aus dieser Perspektive hat das Projekt einem guten Zweck gedient. Die zuständigen Bahnmanager, die die Rechnung begleichen mussten, werden es anders gesehen haben. Vielleicht wurden sie versetzt oder frühzeitig pensioniert, aber so ist es in der IT-Industrie. Es müssen einzelne Menschen geopfert werden, damit die Technologie fortschreitet. Persönliche Niederlagen und Geldverlust darf man nicht so ernst nehmen. Wichtig ist, dass die Technologie voranschreitet.

Meine bescheidene Rolle in dem großen ITS Projekt war es, die Qualität der SPL Module in der Umsetzungsschicht zu sichern. Zu diesem Zweck wurde ich dem Qualitätssicherungsstab unter der Leitung von Dr. K., einem Wiener Wissenschaftler zugewiesen. Wie die meisten Wissenschaftler war Dr. K. in der Theorie der Softwarezuverlässigkeit und der Qualitätssicherung sehr bewandert. Er liebte es, über QS-Prozesse zu reden. Ich dagegen war inzwischen ein Anhänger der Automation geworden. Für mich hatten Methoden nur einen Sinn, wenn man sie mit Werkzeugen umsetzen konnte. Eine Methode oder Technik, die sich nicht per Programm umsetzen ließ, sondern nur der Stoff für ein Buch oder ein Seminar blieb, war für mich blanke Theorie. Man könnte sie zwar propagieren und eventuell das Interesse der Handelnden kurzzeitig gewinnen aber meistens haben sie nur gegähnt und sind ihrer Arbeit in der gewohnten Weise weiter nachgegangen. Das Einzige wofür man Softwareentwickler wirklich gewinnen kann, ist ein neues Gadget oder eine Sprache mit der sie herumexperimentieren können. Ich würde sogar behaupten, jeder echte Fortschritt in der Softwaretechnologie ist mit einem programmierten Werkzeug verbunden, sei es eine Programmiersprache, ein Datenbanksystem, ein Kommunikationsmittel oder eine Standardsoftware. Es dreht sich um Produkte. Meine Meinung war von Anfang an, dass Produkte die Prozesse bestimmen. Es macht keinen Sinn, Prozesse oder Methoden zu propagieren, solange die Produkte fehlen, die diese Visionen konkretisieren.

Also während Dr. K. Denkschriften verfasste und Vorträge hielt, setzte ich mich daran, ein Werkzeug zu entwickeln, um die SPL Module zu testen. Mit diesem Werkzeug sollte es möglich sein, Testdaten überall im Speicherbereich des Moduls zuzuweisen und die Testergebnisse des Moduls jederzeit kontrollieren zu können. Der Ablauf des Steuerflusses sollte beliebig anstoßbar und immer anhaltbar sein. Der Verlauf des Steuerflusses sollte dokumentiert und die Testüberdeckung gemessen werden. Ich war davon überzeugt, dass nur mit so einem Wunderwerkzeug die vielen ITS

Module in der vorgegebenen Zeit getestet werden können. Mit diesem Ziel setzte ich mich daran, das Wunderwerkzeug zu konzipieren.

Sehr bald stieß ich auf die ersten Schwierigkeiten. Wie sollte ich die Adressen der Datenfelder, vor allem die in dem globalen Datenbereich, bekommen um ihnen Werte zuzuweisen oder ihre Werte zu protokollieren? Da fiel mir ein, dass der SPL Computer eine Querverweisliste erstellte in dem die relativen Adressen sämtlicher Daten in Hexadezimal Format angegeben wurden. Ich musste nur die Anfangsadresse der Speicherbereiche finden und die relativen Adressen, bzw. die Absetzungen aus der Compiler-Listings, dazu addieren. Diese Anfangsadressen fand ich in der Linkliste, die vom Linkage Editor herauskam. Für die Parameter, die von außen kamen, konnte ich die Adressen selber bestimmen. Ich musste nur einen Speicherbereich für sie allokieren. Damit war das Problem theoretisch schon gelöst.

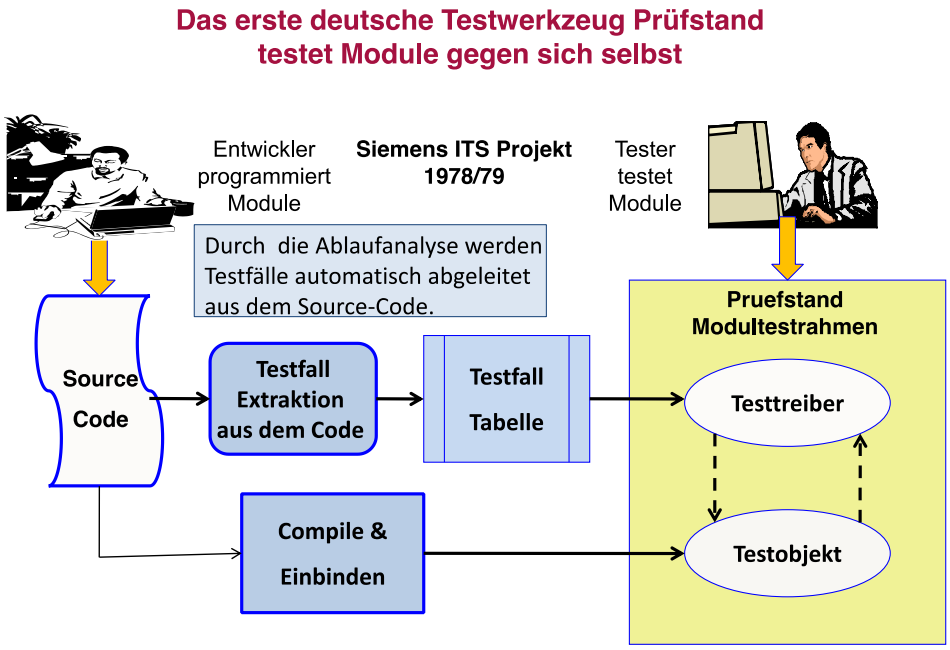
Die Praxis bestand darin, Programme zu schreiben, um diese Computerausgaben zu lesen und zu interpretieren. Da ich niemanden um mich herum hatte, der in der Lage war, derartige Aufgaben zu lösen, blieb mir nichts Anderes übrig, als diese Arbeit selber zu verrichten.

Dr. K. konnte mir nicht helfen. Er war so weit weg vom Programmieren wie Jupiter von der Erde. Ich hätte zu Herrn H. gehen können und um Programmierkapazität bitten. Aber er hätte bestimmt mit Recht ein Proof of Concept verlangt. Wie konnte er gutes Geld für eine Idee ausgeben, deren Nutzen noch gar nicht erwiesen war. Um mir diesen Bittgang zu ersparen, entschied ich, das Werkzeug nach dem Motto „Selbst ist der Mann“ alleine zu entwickeln. Das war der Anfang meiner Tätigkeit als Werkzeugbauer, eine Tätigkeit die ich bis an meinem Lebensende nachgehen würde.

2.6 Prüfstand - das erste deutsche Testwerkzeug

Der Software-Prüfstand war ein ehrgeiziges Projekt für eine einzige Person. Ich bekam Unterstützung nur von einem einzigen Siemens Mitarbeiter – Klaus Kirchhof – der mir half den Instrumentor zu schreiben. Er hatte schon für die Sprache FORTRAN einen Instrumentor entwickelt und veröffentlicht [Kirch77]. Jetzt konnte er ein ähnliches Tool für SPL entwickeln. Der Instrumentor ist ein Tool um den Steuerfluss zu überwachen und zu registrieren welche Zweige im Code durchlaufen werden. Da er keinen offiziellen Auftrag bekommen konnte, hatte er diese Arbeit in seiner freien Zeit verrichtet. Ich habe ebenfalls die meiste Arbeit an Prüfstand außerhalb der normalen Dienstzeit durchgeführt. Ich hatte schon im IQS Projekt begonnen ein Modultestwerkzeug zu konzipieren. Ich hatte es auch teilweise implementiert. Ich wollte das

Tool unbedingt fertigstellen und zum Einsatz bringen. Das I.T.S. Projekt gab mir die Gelegenheit dazu. So entstand das erste deutsche Testwerkzeug durch ein sogenanntes U-Boot Projekt.



Der Siemens Prüfstand bestand eigentlich aus zwei getrennten Komponenten:

- einem statischen Analysator und
- einem dynamischen Analysator [Sned78].

2.6.1 Prüfstand statischer Analysator

Der statische Analysator prüfte den Source Code, berichtete Abweichungen von den Codier-regeln und erstellte vier Entwurfsdokumente:

- ein Datenflussdiagramm
- ein Datenverzeichnis
- ein Baumdiagramm und
- ein Struktogramm.

Das Datenflussdiagramm war eher eine Eingabe/Ausgabe Tabelle. Es zeigte links die Eingabedaten und rechts die Ausgabedaten. In der Mitte war eine Liste der internen Prozeduren. Dies war ein Versuch, ein HIPO Diagramm nachzumachen. HIPO- hierarchische Input / Processing / Output – war damals die vorherrschende Analysetechnik. Es sollte dem Tester zeigen, welche Daten er zu generieren und welche er zu validieren hat.

Das Datenverzeichnis war eine Liste sämtlicher, von dem Modul verwendeten Daten mit Adresse, Typ und Länge. Diese Information über die Daten wurde aus den SPL Compiler Listen entnommen. Es war eine Voraussetzung für die Zuweisung sowie die Kontrolle der Daten zur Laufzeit. Dies war eigentlich meine eigentliche Erfindung. Soviel ich weiß, war ich der erste, der die Ergebnisse der Kompilierung zum Testen benutzte.

Das Baumdiagramm vermittelte den Tester eine Übersicht über die Aufrufhierarchie der internen Prozeduren. Ein SPL Modul bestand – wie auch ein PL/I Modul aus einer Hauptprozedur und mehreren Unterprozeduren. Der Hauptprozedur war der einzige Eingangspunkt. Sie empfing den externen Aufruf mit Parametern und leitete die Steuerung weiter an die internen Unterprozeduren. Die internen Prozeduren konnten sich gegenseitig aufrufen. Deshalb war es wichtig, diese Aufrufstruktur bildhaft festzuhalten, damit die Tester die Interaktionen zwischen den Prozeduren verfolgen konnten.

Das Struktogramm zeigte für die Hauptprozedur sowie für jede Unterprozedur eines Moduls die logische Ablaufstruktur. Jeder Entscheidungsknoten – ob IF Anweisung, Fallanweisung oder Wiederholungsanweisung – führte zu einer Aufteilung des Diagramms in weitere Spalten. Jede Spalte war ein Zweig. Die Aufeinander Reihung der Spalten ergab ein Pfad durch die Prozedur. Es war damit möglich, jeden Pfad von dem Eingang der Prozedur bis zum Ausgang zu verfolgen. Bald kam ich auf die Idee, mit Farbstiften die Pfade durch die Struktogramme mit verschiedenen Farben zu markieren. So gab es der Rote Pfad, der grüne Pfad, der blaue Pfad und so weiter. Auf diese Weise wurden die Ablaufpfade visualisiert und die Tester konnten sie im Test ansteuern und verfolgen. Deswegen musste ich später bei meinen Reisen nach Budapest immer neue Farbstiften mitbringen – im sozialistischen Ungarn gab es sowas nicht.

2.6.2 Prüfstand dynamischer Analysator

Der dynamische Analysator bestand aus drei Teilen:

- Ein Instrumentor als Preprozessor
- Ein Testrahmen als Prozessor und
- Ein Testmonitor als Postprozessor

Der Instrumentor hatte der Klaus Kirchhof geliefert. Es war ein Programm um Durchlaufzähler in den Source-Code einzubauen. In jeden Codezweig bzw. nach jeder Selektion und in jeder Schleife wurde ein Befehl eingebaut um die Anzahl Ausführungen des Zweiges hochzuzählen. Das hatte ich von den vielen Werkzeugen dieser Art in Amerika abgekuckt. Jeder Zweig war ein Eintrag in der Testüberdeckungstabelle, die in jedem Modul eingebaut war. Beim Testbeginn wurde die Tabelle auf Null gesetzt. Während des Tests wurden die Durchlaufzähler hochgezählt. Am Ende des Tests wurde die Tabelle abgespeichert und ausgedrückt. So wurde die Code-Überdeckung gemessen.

Der dynamische Analysator erstellte automatisch aus den von der statischen Analyse gewonnenen Informationen einen Testrahmen mit einem Treiber der das Testobjekt anstieß und Stubs für die Simulation sämtlicher Programmaufrufe, Supervisor-Aufrufe und Ein/ausgabe-Operationen. Damit konnten einzelne Module in einer künstlichen Umgebung unter Aufsicht ausgeführt werden. Der Testanalysator war er in der Lage, während und nach der Programmausführung über die Ablaufpfade und den Stand der Testdeckung bzw. den Anteil der durchlaufenen Zweige, zu berichten. Er generierte Zufallsdaten für die Rechenfelder und gab dem Tester die Möglichkeit im Dialogbetrieb die Parameterdaten zu versorgen. Bei jeder Ein/Ausgabe Operation, bzw. bei jedem Modulaufruf, wurde der Test angehalten damit der Tester die betroffenen Ausgaben prüfen und die betroffene Eingaben zuweisen konnte. Er notiert alle Programmausgaben und verglich sie mit den Soll-Ausgaben die der Tester bereits definiert hat. Daraus folgte ein Abweichungsbericht. Im Prinzip hatte der Tester Zugang zu sämtlichen Daten im Adressraum des Testobjektes. Die Adressen wurden aus den Compiler-Listen entnommen. Dies war eigentlich das Kernstück der Modultesttechnologie.

Der Testmonitor wertete die aufbewahrten Datenzustände sowie auch die Ablaufpfadtabellen aus. Die Datenzustände wurden gegen die vorgegebenen Assertionen geprüft. Die Ablauftabellen wurden analysiert um die Testüberdeckung zu dokumentieren. Codezweige die nicht erreicht wurden, wurden mit Sterne markiert. Am Ende erfolgte eine Überdeckungsstatistik. Ist-Ausgaben / Soll-Ausgaben und ausgeführten Zweige / alle Zweige waren die erste Testmetrik.

Der Prüfstand enthielt viele der Eigenschaften moderner Testwerkzeuge:

- Er trieb den Test ausgewählter Code-Bausteine
- Er simulierte die Betriebsumgebung der Module unter Test
- Er prüfte die Ist-daten gegen die Solldaten
- Er benutzte Assertions um Dateninhalte vorzugeben
- Er dokumentierte die durchlaufenen Ablaufpfade
- Er miss die Codeüberdeckung

Dieses Tool galt lange als das Modell für andere Testwerkzeuge. Auch in den USA wurde es bekannt und zitiert. Auf der ersten Software Maintenance Conference in Monterey, California, habe ich darüber berichtet [Sned83]. Heute würde wohl keiner auf die Idee kommen, ein eigenes Modultestwerkzeug zu bauen. Es gibt genug auf dem Markt.

2.7 Vom Entwickler zum Tester

In Dallas hatte ich mit dem amerikanischen Testexperte Ed Miller ausgemacht, dass er nach Deutschland kommen soll, um ein Testseminar zu halten. Mit der Schulungsfirma GES am Bodensee hatte ich schon eine gute Beziehung – seit 1974 hatte ich für sie öffentliche Seminare über Strukturierte Programmierung und Software Projektmanagement mit großem Erfolg durchgeführt. Die Seminare waren immer voll. Das Institut hat mir zu dieser Zeit auch geholfen mein Buch über strukturierte Programmierung herauszubringen. Ich nutzte nun meine Beziehung zum GES Training Institut um dieses Testseminar in die Wege zu leiten. Der Geschäftsführer des Instituts hatte ein mächtiges Vertriebsnetz aufgebaut um seine Schulungsveranstaltungen zu vertreiben und Schulungsmaterial wie mein Strukturierte Programmierungsbuch zu verteilen. Solche externen Seminare waren damals groß im Trend. Sie und die Konferenzen waren die bevorzugten Mittel, den Mitarbeitern und Managern neue Technologien zu vermitteln.

Ich sah zwei Möglichkeiten diese neue Technologie in Deutschland einzuführen. Die eine war über die öffentliche Schulung, die von Ausbildungsinstitute angeboten wurde, die andere war ein konkretes Projekt. Ich nahm beide Möglichkeiten wahr. Über die öffentlichen Seminare war es möglich, die Menschen in den DV-Projekten zu erreichen. Um deren Manager zu kontaktieren war es notwendig, auf den großen Fachkonferenzen aufzutreten. Das Control Data Institut hatte gerade begonnen solche Tagungen zu veranstalten. Die erste mit der Bezeichnung STRUCTO 76 hatte gerade im Herbst 1976 stattgefunden. Das Thema war die strukturierte Programmierung und ich selber hatte dazu einen Beitrag geleistet. Ich hatte eine gute Beziehung zu der Konferenzleiterin der CDI und konnte sie dafür gewinnen einen ähnlichen Kongress

zum Thema Testen gewinnen. Auf diesem Wege wollte ich auf die Bedeutung des Testens aufmerksam machen.

Gleichzeitig hatte ich die Möglichkeit, Dr. Ed Miller in das Siemens I.T.S. Projekt hineinzubringen. seit dem Sommer 1977 war ich in dem Projekt als Testmanager tätig, allerdings hatte ich niemandem zu managen. Ich war ganz alleine auf mich gestellt, denn Tester gab es keine. Keiner bei Siemens konnte sich damals vorstellen als Tester zu arbeiten. Ich verbrach meine Zeit damit Testkonzepte auszuarbeiten und abends Probetests durchzuführen. Da der Testrechner tagsüber von den Entwicklern ausgelastet wurde dürfte ich nur abends ab 1700 testen. Das Bewusstsein für die Bedeutung des Testens fehlte noch beim oberen Management. Wenn ich einen international bekannten Experten hineinbringe, könnte ich das Management eventuell beeinflussen, mehr in den Test der I.T.S Software zu investieren.

Der Auftakt zu Einführung der Testtechnologie hatte schon Ende 1977 stattgefunden. Ich habe in Schwarzwald für die GES das erste öffentliche Seminar zum Thema Softwaretest in Deutschland gehalten. Es ging hauptsachlich um die Instrumentierung von Code und die Messung der Codeüberdeckung aber auch um Testtreiber und Test Stubs. Das Interesse an dem Thema war groß. Mein erster Fachartikel über Testen erschien 1977 in der Zeitschrift für Datenverarbeitung. Weitere Veröffentlichungen zum Thema Test folgten. Es zeigte sich, dass die Fehleraufdeckung in der Praxis ein großes Problem war. Das Interesse an Werkzeugen um den Test zu unterstützen war daher besonders groß. Das hat mich ermuntert, die Prüfstandentwicklung voranzutreiben.

Im neuen Jahr 1978 überhäuften sich die Ereignisse. Im Februar veranstaltete die Infotech, eine Schulungsfirma in Maidenhead, G.B. die erste europäische Testkonferenz in London. Dr. Miller war als Keynote Speaker und Organisator eingeladen [Mill79]. Er hatte mich eingeladen ein Paper über das Prüfstand Werkzeug einzureichen. Es wurde angenommen und ich bin nach London gereist, um meinen Vortrag zu halten. In dem Vortrag habe ich erwähnt, dass wir in dem I.T.S. Projekt zwar gute Werkzeuge hatten, aber keine Tester um sie zu bedienen. In der Pause traf ich auf einen Ungarn, namens Gyuri. Er kam von einem Institut aus Budapest. Gyuri redete auf mich ein, ich soll nach Budapest kommen und die Leute dort in das Projekt einbeziehen Ich sagte, ich würde mir das überlegen. Zu diesem Zeitpunkt war ich damit beschäftigt, einen Termin für Dr. Miller bei der Siemens Projektleitung in München zu vereinbaren, was mir auch endlich gelungen ist. Dr. Miller sollte nach der Tagung nach München kommen und sich mit dem Technologiemanagement treffen. Zunächst ging es nur um eine Beratung in Sache Software Qualitätssicherung im I.T.S Projekt.

Gleichzeitig habe ich mit der GES ausgemacht, dass dort Dr. Miller für sie ein Seminar über Testtechnologie hält. Zu diesem Zweck soll er im Frühjahr nach Deutschland zurückkehren. Sein Projekt – das RXPV-Testpaket für die Missile Defense System wurde gerade von dem U.S. Verteidigungsministerium auf Eis gelegt. Die Amerikaner waren im Begriff sich mit den Russen über die atomare Abrüstung zu einigen – das sogenannte SALT Abkommen und sie wollten ihre Investitionen in die Missile Verteidigung reduzieren. Also hatte Dr. Miller freie Kapazität, um nach Deutschland und anderswo die Botschaft der Software Qualitätssicherung hinauszubringen. Wir verblieben dabei, dass er bald zurückkommt um Siemens zu beraten und öffentliche Seminare zu halten.

Testing Techniques Newsletter

Volume 1 Number 3 December 1978

CONTENTS….

P.O. Box 2432 • San Francisco • California 94126 • Telephone (415) 957-1441 TN-517

Noch im Monat Februar traf ein weiteres Ereignis zu. Auf einer Faschingsfeier in Neuperlach traf ich eine Ungarin, die bereit war mit mir bei einer geschäftlichen Beziehung in Budapest zu helfen. Sie ist mit ihrer kleinen Tochter über Nigeria nach Deutschland ausgewandert und bekam aufgrund ihrer deutschen Herkunft die Staatsbürgerschaft. Damit konnte sie ohne weiteres nach Ungarn ein- und ausreisen, was sonst nicht möglich gewesen wäre. Außer Ungarisch und Deutsch sprach sie auch ein

ausgezeichnetes Englisch und suchte nach einem Job. Das habe ich mir gemerkt und als die Sache mit Ungarn aktuell wurde, habe ich sie sofort kontaktiert. Der Weg nach einer neuen Zukunft in Ungarn zeichnete sich auf. Jetzt brachte mich ein weiteres Ereignis näher dazu.

2.8 Das Caroline Testprojekt der Firma Spardat in Wien

Im März 1978 bekam ich ein Testprojekt für das Spardat Rechenzentrum in Wien. Ein Teilnehmer meiner Seminare für die GES war ein gewisser Herr Hrib aus Wien. Herr Hrib war EDV-Trainer an der HTL-Schule und veranstaltete Seminare für Anwenderbetriebe in Wien. Herr Hrib wurde durch meine Seminare für die GES inspiriert mich nach Wien zu holen um dort ebenfalls solche Seminare zu halten. Der erste Kunde war das Spardat Rechenzentrum Ich sollte dort ein Seminar über Software Testen halten. Der Spardat hatte gerade ein großes Projekt für den Online-Banking angelegt. Es war das erste Projekt dieser Art in Österreich. In jeder Zweigstelle sollte ein Arbeitsplatzrechner stehen, der mit dem zentralen Buchungssystem in Wien verbunden ist. Die Server-Software in der Zentrale wurde von der Spardat selbst entwickelt. Die Software für die Client-Rechner wurde von Philips, dem Lieferanten der Rechner bereitgestellt. Die Datenkommunikationssoftware wurde von einem dänischen Softwarehaus entwickelt. Die zuständigen Manager der Spardat hatten erkannt, dass das Hauptproblem in der Integration und dem Test des Gesamtsystems liegen würde. Sie suchten nach Rat. Trotz meiner vielseitigen Tätigkeit bei Siemens in München nahm ich die Gelegenheit wahr und kam das erste Mal beruflich nach Wien.

Auf dem Seminar wurde vereinbart, dass ich für die Spardat gleich in das Caroline Projekt als Tester einsteigen sollte. Damit wurde ich möglicherweise der erste Software-Tester in Österreich. Der Test fand nachts in einer Zweigstelle in Stockerau statt. Herr Hrib hat für mich ein Zimmer in den 3. Bezirk unweit von der Spardat in der Unteren Zollamt Gasse besorgt. Also musste ich abends mit der S-Bahn nach Stockerau hinausfahren um meine Testfälle durchzuführen. Getestet wurde nachts, weil der große IBM Rechner in der Spardat tagsüber mit der Produktion ausgelastet war – sowie bei der Siemens. Vor 0100 Uhr nachts war ich nie zurück im Hotel.

Der Spardat Test war mein erster großer Systemtest. Anders als bei Siemens bekam ich den Code nicht zu sehen. Für mich war es eine echte Black-Box. Die 88 Testfälle, die ich spezifizierte, entnahm ich aus der Bedienungsanleitung für das Endgerät. Ich verlangte 700 Schilling, bzw. DM100, pro ausgeführten Testfall plus 1400 Schilling pro aufgedeckten Fehler. Die Testfälle waren bei mir als Formblätter in einem Leitz-Ordner abgeheftet. Ich machte eine Risikoanalyse und versuchte die Schnittstellen des Systems auszureizen. Zum Beispiel, im Falle eines Ausfalls der

Verbindung zum Server sollte es möglich sein an dem Endgerät weitere Buchungen zu betätigen. Allerdings war der Puffer für diese Buchungen beschränkt. Sollte die Verbindung zum Server zu lange ausbleiben war vorauszusehen, dass der Puffer überläuft und so war es auch. Ich musste diese und ähnliche Grenzfälle testen. Zum Schluss habe ich vier schwere Fehler – wie der Arbeitsplatz-Pufferüberlauf - und neun leichte Fehler, wie die Verwechselung der Systemmeldungen die an den Endbenutzer gemeldet werden. Der Caroline Systemtest dauerte nicht länger als sechs Wochen in denen ich mehrere Abende von Stockerau aus getestet habe. Tagsüber habe ich den Test bei der Spardat vorbereitet. Es gab also keine Zeit Wien kennen zu lernen. Dazu sollte ich Jahrzehnte später genug Gelegenheit haben. Bis Mai 1978 war die Spardat mit dem Testergebnis zufrieden und ich war frei nach Budapest zu gehen.

3 Das Budapester Testlabor

3.1 Zum Ursprung des Testlabors

Die Idee eine separate Testmannschaft zu bilden geht zurück auf die Arbeitsteilung in dem amerikanischen Ballistic Missile Defense Projekt. In dem Projekt hatte es getrennte Teams für die einzelnen Projektphasen gegeben:

- ein Team für die Anforderungsspezifikation,
- ein Team für den System Design
- ein Team für die Programmierung und
- ein Team für den Test [Alfo77].

Jedes Team gehörte zu einem anderen Auftragnehmer des Verteidigungsministeriums und war an einem anderen Ort in den U.S.A. Das Test-Team gehörte zur General Research Corporation und saß in Santa Barbara, California. Dr. Ed Miller war dort der Teamberater [Mill77].

Das Test Team bekam die Anforderungsspezifikation von dem Anforderungs-Team der TRW in Alabama und der Code von dem Los Alamos Labor in New Mexico. Die Testfälle wurden zum einen aus den Anforderungssprache- RSL – und zum anderen aus dem Code – FORTRAN – entnommen. das Ziel war, alle Pfade durch den Code zu testen. Der Requirements-Test wurde als Black-Box Test bezeichnet, der Code-Test als White-Box Test. In beiden Tests wurde die Testüberdeckung gemessen bzw. der Prozentsatz der durchlaufenen Knoten und Kanten. Beim Anforderungstest wurde mindestens 95% Testüberdeckung und beim Code-Test mindestens 85% angestrebt. Fehler wurden durch den Abgleich der spezifizierten Sollergebnisse mit den tatsächlichen Ist-Ergebnisse des Codes aufgedeckt. Die Soll-Ergebnisse wurden mit eingebauten Assertionen definiert. Die Ablaufpfade im Code wurden außerdem mit den spezifizierten Pfaden in den Anforderungsnetzen abgeglichen um Abweichungen im Ablauf festzustellen.

In diesem ersten größeren Testprojekt Mitte der 70er Jahre finden wir also viele Eigenschaften der heutigen Testtechnologie- Assertions, Soll/Ist Datenabgleich, Soll/Ist Ablaufabgleich und Testüberdeckungsmessung. Das Wesentliche war die totale Trennung des Tests von der Entwicklung. Das Testen wurde zur eigenständi-

gen, ingenieursmäßigem Tätigkeit und die Tester wurden zu Testingenieuren mit eigenen Methoden und eigenen Werkzeugen [Rama75].

Ich wollte dieses Modell auf das Siemens I.T.S-Projekt übertragen. Das Werkzeug für die Assertions-Prüfung und die Testüberdeckungsmessung hatte ich schon – den Prüfstand. Die Tester sollen von den ungarischen Instituten kommen. In dem SZKI Institut stand der Testrechner. Als Testberater hatte ich Dr. Miller zur Seite. Der Code wurde vom Siemens geliefert. Was mir fehlte, waren die Testdaten und die Anforderungsspezifikation. Künstliche Testdaten konnten wir selber aus den Datenbeschreibungen im Code ableiten, auch wenn es umständlich und arbeitsintensiv war. Was wir nicht selber machen konnten, war die Anforderungen zu spezifizieren. Ein Test sollte immer ein Test gegen etwas sein. Wir hatten nichts, wogegen wir hätten testen können. Als Notlösung habe ich ein Tool entwickelt, um Struktogramme aus dem Code abzuleiten. Diese dienten als Basis für unsere Testfalldefinition. Mir war aber klar, dass dies kein echter Test war. Wir haben den Code praktisch gegen sich selbst getestet. Da wir aber keine weitere Dokumentation bekamen, blieb mir nichts Anderes über. Später sollte ich diese Notlösung bereuen.

Dies wirft eine immer wiederkehrende moralische Frage der Software Entwicklung auf. Ist es besser einen fragwürdigen Auftrag anzunehmen, obwohl man weiß, dass er nur unzulänglich erfüllt werden kann, als den Auftrag gleich aus diesem Grund abzulehnen? In diesem Fall habe ich gewusst, dass ein Modultest allein nicht ausreichend würde um die Qualität der Software zu sichern. Dennoch ließ ich den Kunden in dem Glauben, wir würden den Löwenanteil der Fehler auf dieser Weise finden. Wenn er das gewusst hätte, hätten wir den Auftrag nicht bekommen und es wäre nie zu dem Testlabor gekommen. So aber bekam ich die Gelegenheit, wenigstens die Hälfte der Fehler zu finden und die Testtechnologie einen guten Schritt voranzubringen. Dieser war auch der Grundstein für meine spätere Toolentwicklung und der Anfang meiner langjährigen Kooperation mit den Ungarn. Hätte ich vom Anfang an auf die Grenzen dieses Verfahrens hingewiesen, wäre das alles nicht zustande gekommen.

Viele Jahre später wurde ich mit der gleichen Frage als Expertenzeuge vor dem Gericht konfrontiert in dem Fall Debis gegen Volvo. Debis hatte einen Auftrag von Volvo bekommen die In-Flight-Überwachungssoftware für die Düsenmotoren am Flugzeug zu entwickeln. Die Anforderungsspezifikation war völlig unzulänglich und konnte niemals als Grundlage für eine seriöse Software Entwicklung dienen. Deshalb wurde Debis von der Schuld an dem Scheitern des Projektes freigesprochen. Aber als der Anwalt von Volvo mich fragte, warum Debis den Auftrag überhaupt angenom-

men hat, wenn die Anforderungsdokumentation so schlecht war, konnte ich keine Antwort geben. Sie waren in der gleichen Situation wie ich damals mit Siemens. Sie brauchte das Projekt.

3.2 Meine Vision eines Software Testlabors für das I.T.S Projekt

Die Idee einer unabhängigen Testinstanz ist eigentlich aus dem I.T.S. Projekt hervorgegangen. Die Siemens Projektleitung hatte entschieden parallel zur Entwicklungsabteilung eine separate Qualitätssicherungsgruppe zu installieren. Zu dem Zeitpunkt war Qualitätssicherung in aller Munde, es gab schon die ersten Bücher zu diesem Thema, u.a. von Siemens Mitarbeiter[Siem77], und das Siemens Projektmanagement war entschlossen sie in dem neuen Großprojekt zu praktizieren. Ein Qualitätsmanager, mein Vorgesetzter, Dr. K aus Wien, war schon im Amt als ich zum Projekt zugestoßen bin. Es war allerdings nicht klar was unter Qualitätssicherung zu verstehen war. Es gab verschiedene Ansätze die aus der amerikanischen Literatur übernommen wurde, Ansätze wie Reviews, Walkthroughs, Inspektionen, Testen, usw. [McCa77], nur wusste keiner so recht wie sie einzuführen wären. Meine Veröffentlichungen über Testen waren innerhalb Siemens inzwischen bekannt und ebenfalls meine Vorträge zu diesem Thema. Deshalb hat der Entwicklungsleiter, der Herr H., mich damals gedrängt bei ihm als Testmanager einzusteigen.

Schon während meiner Amerikareise habe ich, inspiriert von den Testvorträgen auf dem NCC Kongress in Dallas [NCCD77], begonnen Gedanken zu machen wie ich den Test der I.T.S. Software am besten organisieren konnte. Ich stellte mir ein separates Testprojekt vor das neben dem Entwicklungsprojekt ablaufen sollte. Die Entwickler würden ihre Module an die Tester übergeben und diese würden die Fehler an die Entwickler zurückmelden. Ich war bereits dabei die Werkzeuge dafür zu entwickeln. Was mir fehlte waren die Menschen dazu. Herr H. hatte genug Probleme gehabt Entwickler für das I.T.S. aufzutreiben. Von Testern war keine Rede. Dann folgte die Begegnung mit dem Lazar, Gyüry in London und eine neue Perspektive eröffnete sich.

3.3 Ausflug in eine fremde Welt

Nach Abschluss des Caroline Testprojektes im Mai 1978 bin ich von Wien aus mit der Bahn nach Budapest gefahren. Damals dauerte diese 220 km Bahnfahrt mit dem Orientexpress fast vier Stunden. Am Keleti Bahnhof hat mein Partner von der Londoner Konferenz auf mich gewartet. Er war sehr bedacht, mir die positiven Seiten von Budapest zu zeigen. Er brachte mich zum Essen in den Matyas Keller, ein Lokal an

der Elisabethbrücke, der fast nur von Ausländern besucht war. Es gab die übliche Zigeunermusik und das üppige Essen und anschließend brachte er mich zu dem Institutshotel. Das Szamok Institut war an südwestlichen Rand von Buda in einem Vorort namens Kelenföld direkt vor dem alten Bahnhof situiert. Der Bahnhof war ein Relikt aus der K&K Zeit. Das Institutsgebäude war ein krasser Gegensatz dazu, ein typischer Ostblockbau mit Fertigbetonteilen und viel Aluminium. Es wurde von der UNESCO finanziert als DV-Ausbildungszentrum für die osteuropäischen Länder gedacht. Unten waren das Rechenzentrum und die Schulungsräume, oben die Einzelzimmer mit Balkon für die Kursteilnehmer, die dort übernachteten. Da es an Kursteilnehmer fehlte, wurden die Zimmer auch an Touristen angeboten und da es an Einkommen fehlte wurde ein Software-Entwicklungslabor eingerichtet. Dieses Labor sollte Entwicklungsleistungen für andere Industriebetriebe bringen. Im obersten Stock war ein Bar für Kursteilnehmer und Touristen wo man auch Sandwiches bekommen konnte. Die erste Zeit in Budapest wurde ich dort untergebracht.

Am nächsten Morgen lernte ich die Institutsleitung von SZAMOK kennen. Der Gewerkschaftsleiter und die kaufmännische Leiterin waren immer in Begleitung des Institutsdirektors. Hier lernte ich auch den Leiter der Softwareabteilung, Miklos Rabar kennen. Obwohl seine Abteilung nur den Auftrag hatte für ungarische Industriebetriebe zu arbeiten, war er nicht abgeneigt auch Divisen aus dem Ausland zu verdienen. Mit ihm sollte ich den Kooperationsvertrag abschließen. Da wir aber für den Test der I.T.S. Software einen Siemens Rechner brauchten, hatte der Abteilungsleiter gleich einen Termin mit einem anderen staatlichen Institut in der Innenstadt SZKI ausgemacht. Dort sollte ich den Chef der Siemens Exportabteilung – Győző Kovács – kennenlernen. Seine Abteilung war ermächtigt Geschäfte mit dem Ausland abzuwickeln. Sie hatte schon mehrere Entwickler bei Siemens in München platziert. Da es dem SZKI an Divisen fehlte waren diese Entwickler die Bezahlung für den Siemens Rechner den sie betrieben. Einige von ihnen hatte ich schon in meiner Zeit als Datenbankentwickler bei Siemens kennengelernt, u.a. Frau Nyary, die spätere SofSpec Projektleiterin.

Das erste Treffen fand auf dem Budapester Messegeländer statt. Es war gerade Messezeit und SZKI war dort um ihre Rechenleistungen anzubieten. Herr Kovacs, ein jovialer kleiner Mann mit Sakko und einer Fliege saß in einem Konferenzraum, umgeben von seinen Unterabteilungsleitern und kaufmännischen Angestellten. Ich saß mutterseelenallein in meiner alten Lederjacke den fremden Funktionären gegenüber und sollte mit denen verhandeln. In erster Stelle ging es um die Rechnernutzung, das Personal war von untergeordneter Bedeutung. Rechenzeit war zu dieser Zeit in Ungarn ein kostbares Gut – eine Rechenstunde kostete mehr als 100 Personenstunden.

Das hat mich zuerst schockiert aber ich behielt die Fassung. Überhaupt war ich dort in einer sehr schwierigen Situation. Ich gehörte nicht zur Siemens, vertrat keine Firma und besaß kein Renommee. Eigentlich war ich ein Niemand. Außer meinem Werkvertrag mit Siemens und einem Kooperationsvertrag mit der Firma von Ed Miller – Software Research Associates – hatte ich nichts in der Hand. Aber ich hatte eine Vision und war entschlossen sie durchzusetzen. Da ich wusste, dass Siemens niemals nach Stunden bezahlen würde, habe ich die Diskussion über Rechen- und Personenstunden gleich abgebogen und bin auf die Leistung zu sprechen gekommen. Siemens wollte, dass ihre Softwaremodule getestet und deren Fehler aufgedeckt werden. Außerdem wollte der Siemens Projektleiter wissen, ob er der Software trauen konnte, ob er sie an die Bahn mit gutem Gewissen übergeben konnte. Wer heute die Prüfung zum Certified Tester ablegen will, muss die Frage nach dem Ziel des Testens beantworten. Die Antwort steht in dem Buch von Spillner und Linz „Basiswissen Softwaretest“ [Spil05]. Das Ziel des Testens ist demnach:

- Fehler zu finden und
- vertrauen zu bilden.

Es kam also darauf an, diese beiden Kriterien zu erfüllen. Ich habe das peinliche Gespräch über Stundensätze und Rechnerkosten vermieden und bin gleich auf die Leistung zu sprechen gekommen. Bei mir ging es nur um die Leistung „only the results count“. Das war im Falle vom Test – die Überdeckung der Software mit Testfällen und die Aufdeckung von Fehlern. Die Bezahlung würde sich nach dieser Leistung richten. Ich wollte auch auslotsen, wie hungrig meine Gegenüber waren.

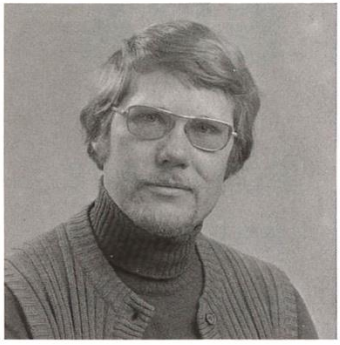
Meine ungarischen Gesprächspartner waren zunächst verwirrt. Über den Wert einer Softwareleistung hatten sie bestimmt nie nachgedacht. Sie hatten bis dahin nur über Personentage und Rechenstunden gehandelt. Über die Leistung die dabei herauskommen sollte zu sprechen war für sie fremd. Sie baten um Zeit um sich die Sache zu überlegen. Ich sollte ein zweites Mal wiederkommen.

Das zweite Mal war das Treffen bei dem SzKI Büro am Martinelli-Platz in der Pester Innenstadt. Dieses Mal waren auch Techniker dabei. Sie haben zwar nicht alles verstanden, aber Herr Kovacs hatte irgendwie Vertrauen zu mir gefasst. Er war bereit, sich auf das Projekt einzulassen. Was ihm dazu bewegt hat, weiß ich bis heute nicht. Vielleicht witterte er ein noch größeres Geschäft dahinter. Jedenfalls haben wir uns bald geeinigt. Die beiden Institute sollten 75% der Bezahlung von Siemens erhalten. Diese 75/25 % Aufteilung des Einkommens blieb erhalten, auch für die Folgeprojekte bis zum Ende unserer Zusammenarbeit. Ich musste meine Partner in die Verantwortung einbeziehen und dazu musste ich ihnen einen großen Anteil an das Einkommen

anbieten, vor allem wenn völlig unsicher ist wie groß das Einkommen sein würde. Das erste Testprojekt war mit hohen Risiken verbunden. Keiner wusste wie viele Testfälle wir brauchten und wie viele Fehler wir finden würden. Kritiker haben später gemeint, ich bin den Ungarn bei dieser Aufteilung viel zu weit entgegengekommen aber ich empfand sie als gerecht und so blieb es.

Zielsetzung

Software Research Associates of Germany ist eine Organisation mit dem Ziel, die Qualität der Software-Entwicklung im deutschsprachigen Raum zu verbessern. Zur Realisierung dieses Zieles bietet sie eine Reihe von Seminaren und Workshops zum Thema Software-Technologie, Beratung in der Planung und Einführung neuer Methoden, Werkzeuge zur Unterstützung der Software-Entwicklung und die Qualitäts-



kontrolle von Software-Produkten in ihrem Budapester Testlabor an.

Bei der Entwicklung von Software-Werkzeugen, sowie dem Test und der Bewertung fremder Software arbeitet SRA Germany mit zwei ungarischen Computerinstituten – das Koordinationsinstitut für Rechentechnik (SZKI) und das Internationale Institut für Computer-Ausbildung (SZAMOK) zusammen. Dort in Budapest werden eigene Werkzeuge für den deutschen Markt entwickelt und fremde Software-Produkte geprüft. Bei der Ausbildung und Beratung arbeitet SRA, Germany, mit ihrer Schwesterfirma SRA, San Francisco, zusammen.

Der Leiter des SRA-Testlabors – Harry M. Sneed – ist seit vielen Jahren als Lehrer und Forscher auf dem Gebiet der Software-Entwicklung tätig. SRA legt großen Wert auf ihre Beziehungen zu der wissenschaftlichen Welt sowohl in den USA als auch in Deutschland und investiert einen beträchtlichen Teil ihres Einkommens in die eigene Forschung. Wer von SRA bedient wird, kann also getrost sein, daß er von den neuesten Entwicklungen in der Software-Welt profitiert.

SOFTWARE RESEARCH

3.4 Die Vision wird Wirklichkeit

Mit dem Siemens ITS Projektleiter habe ich vereinbart, dass er DM 75,- pro Testfall und DM 150,- pro aufgedeckten Programmfehler bezahlen. Für ihn war diese Art von Bezahlung auch neu und er war nicht sicher, was er für sein Geld bekommt. Das Ganze erschien ihm als Forschungsexperiment zu sein. Um ihm etwas mehr Sicherheit zu geben, habe ich in das Angebot hineingebracht, dass wir mindestens 85% Zweigüberdeckung zusichern und dass wir alle Testfälle automatisch protokollieren. Die Testprotokolle und die Fehlermeldungen sollten die Basis für die monatliche Bezahlung sein. Es war nicht einfach, die Siemens Projektleitung von dem Nutzen des ausgelagerten Tests zu überzeugen. Hierbei erwies sich die Unterstützung von Dr. Miller als sehr hilfreich. Er ist nach München gekommen und hat an den Verhandlungen teilgenommen. Damals galt er als der weltgrößte Experte auf dem Gebiet des Testens und das hat die Siemens Manager beeindruckt. Nach einigen Diskussionen mit der Einkaufsabteilung wurde der Auftrag schließlich genehmigt. Mit Dr. Miller habe ich vereinbart, dass das Projekt unter dem Namen seiner Firma laufen sollte – die Software Research Associates. Damit hatte auch er etwas davon, zwar nicht finanziell aber für seinen wissenschaftlichen Ruf. Er war beteiligt an dem ersten ausgelagerten, Festpreistestprojekt. Der Vertrag, der zustande kam war also zwischen Siemens und SRA. Der Vertrag mit den Ungarn war zwischen mir persönlich und dem ungarischen Außenhandelsfirma Metrimpex, die die beiden Institute im Ausland vertrat. Damals war es dem ungarischen Firmen nicht erlaubt im Ausland direkte Geschäfte zu machen, sie dürften nur indirekt über das Außenhandelsministerium Divisen verdienen. Diese hatte dafür einen Anteil des Einkommens bei sich behalten.

Das Ganze war jedenfalls äußerst kompliziert. Ich stand zwischen dem ungarischen Außenhandelsministerium auf der einen und dem Siemens Entwicklungsabteilung auf der anderen Seite, und handelte teils im Namen der SRA und teils im eigenen Namen. Dazu gab es drei verschiedene Verträge. Dies war überhaupt meine erste Erfahrung mit solchen Verträgen und ich machte alles ohne Rechtsberatung. Um mich einzuarbeiten habe ich dasselbe getan was ich tat um eine fremde Programmiersprache zu lernen. Ich studierte Beispiele von Softwareverträgen aus der amerikanischen Literatur und machte mir den Stil zu Eigenem.

In dieser Situation viel mir die Begegnung mit Gabriella Molnar ein. Sie hatte früher bei einer ungarischen Außenhandelsfirma gearbeitet und beherrsche neben ihrer Muttersprache - Ungarisch – auch Deutsch und Englisch. Ich suchte sie auf und bat ihr an für mich als freie Mitarbeiterin zu arbeiten. Da ich damals noch kein Büro hatte, war sie einverstanden in ihrer Wohnung ein Zimmer als Büro einzurichten. Sie sollte meine Kurse organisieren, Unterlagen erstellen, meine Fachartikel schreiben und die Abrechnungen für die Ungarn machen. Ich hatte großes Glück die richtige

Person für das Projekt zu finden. Frau Molnar brachte alles was ich damals brauchte. Sie blieb meine rechte Hand und wurde später meine Prokuristin.

Dienstleistungen

Die Dienstleistungen der SRA teilen sich in vier Kategorien:

- Ausbildung,
- Beratung,
- Entwicklung und
- Kontrolle.

Ausbildung

Dieser Bereich umfaßt eine Reihe von Seminaren zum Thema Software-Technologie. Sie behandeln alle Phasen der Software-Entwicklung von der Spezifikation bis zur Wartung. Dazu gehören die Fachseminare:

- Software-Entwurfsmethodik,
- Strukturierte Programmiertechnik,
- Systematisches Testverfahren

und das Management-Seminar:

- Software-Projektmanagement.

Außerdem bietet SRA zwei Workshops an:

- Software-Entwicklung und
- Software-Qualitätssicherung.

Die Seminare werden zum größten Teil von Herrn Sneed selbst durchgeführt. Sie werden entweder in München oder bei Kunden im Hause durchgeführt. Die Workshops finden dagegen im SZAMOK-Institut in Budapest statt. In dem dortigen Schulungszentrum haben die Teilnehmer Gelegenheit mit den SRA-Werkzeugen zur Entwicklung und Kontrolle von Programmen direkt zu arbeiten.

SOFTWARE RESEARCH

Beratung

Dieser Bereich beinhaltet persönliche und schriftliche Beratung. Bei der persönlichen Beratung kommt ein SRA-Mitarbeiter zum Kunden ins Haus und bespricht mit ihm die Lösung seiner Software-technischen Probleme. Bei der schriftlichen Beratung werden kundenspezifische Konventionen für die Software-Entwicklung, z. B. Entwurfsverfahren, Codierrichtlinien oder Testkriterien ausgearbeitet und präsentiert.

Entwicklung

Dieser Bereich schließt

- Entwurf,
- Implementierung,
- Installation und
- Wartung

von Software-Hilfsmitteln ein, insbesondere Werkzeuge für die Unterstützung und Kontrolle der kundeneigenen Software-Produktion.

Die Hilfsmittel werden zusammen mit dem Kunden entwickelt, d. h. die SRA-Mitarbeiter arbeiten zum Teil beim Kunden und zum Teil in ihrem eigenen Rechenzentrum in Budapest. Dabei wird besonderer Wert auf die Know-how-Übergabe an den jeweiligen Kunden gelegt. Nach Fertigstellung des Produkts kann SRA den Kunden im Rahmen eines Wartungsvertrages solange unterstützen, als der Kunde es für nötig hält, d. h. ein SRA-Mitarbeiter bleibt beim Kunden, um ihm bei der Handhabe des Produkts zu helfen.

SOFTWARE RESEARCH

Endlich war es so weit, dass wir mit dem Testen der I.T.S. Software beginnen konnten. Im Juli 1978 habe ich das erste Band mit I.T.S Modulen nach Budapest geflogen. Es gab am Anfang Probleme mit dem Zoll aber das hat sich bald erledigt. Ich wurde am Flughafen bevorzugt behandelt und durfte ohne Kontrolle durchgehen. Bei der SzKI warteten die ersten zwei Tester auf mich, Maria Majoros von dem Szamok und Tibor Jobbagy von der SzKI. Die beiden Institute haben die Aufgaben und auch

die Einnahmen 50/50 aufgeteilt. Ich hatte fortan mit zwei Partner zu tun, was sich für mich nicht negativ auswirkte. Da die beiden Institute Rivalen waren, konnte ich sie nach dem Prinzip „Divide et imperium“ gegeneinander ausspielen. Dieses Dreiecksverhältnis wehrte in unserer Zusammenarbeit bis zum Ende des sozialistischen Systems im Jahre 1990.

Es kamen später weitere Tester hinzu, aber die 50/50 Aufteilung zwischen den beiden Instituten blieb mehr oder weniger erhalten. Es ging weiter so für ein ganzes Jahr – ich bekam zweiwöchentlich ein Band mit den Source-Modulen und flog damit nach Budapest. Die Tester haben die Sourcen auf den Siemens Rechner bei der SzKI eingespielt und begannen mit dem Testprozess, der zwar am Angang etwas holprig war aber im Laufe der Zeit immer glatter lief.

3.5 Der Modultestprozess mit Prüfstand

Das Testverfahren vollzog sich in vier Phasen:

- Entwurfsanalyse,
- statische Programmanalyse,
- dynamische Programmanalyse und
- Testauswertung.

3.5.1 Modulentwurfsanalyse

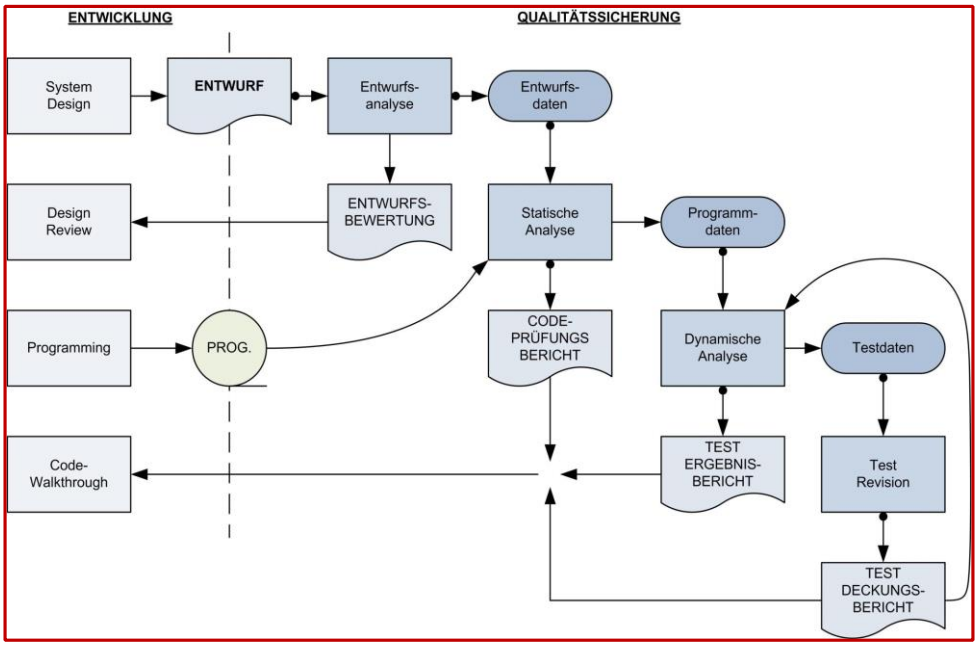
Die Modulentwurfsanalyse begann mit einer automatisierten Nachdokumentation der einzelnen Moduln durch das Prüfstand Werkzeug. Bei der Nachdokumentation wurden vier Diagrammtypen für jedes Modul generiert:

- ein Datenflussdiagramm
- ein Datenverzeichnis
- ein Baumdiagramm und
- ein Struktogramm.

Das Datenflussdiagramm war eher eine Eingabe/Ausgabe Tabelle. Es zeigte links die Eingabedaten und rechts die Ausgabedaten. In der Mitte war eine Liste der internen Prozeduren. Dies war ein Versuch, ein HIPO Diagramm nachzumachen. HIPO- hierarchische Input / Processing / Output – war damals die vorherrschende Analysetechnik. Es sollte dem Tester zeigen, welche Daten er zu generieren und welche er zu validieren hat.

Das Datenverzeichnis war eine Liste sämtlicher, von dem Modul verwendeten Daten mit Adresse, Typ und Länge. Diese Information über die Daten wurde aus den

SPL Compiler Listen entnommen. Es war eine Voraussetzung für die Zuweisung sowie die Kontrolle der Daten zur Laufzeit. Darauf basierten die Testdatenzuweisung sowie auch die Testergebnisprüfung. Es war die Landkarte des Testobjektes.



Das Baumdiagramm vermittelte den Tester eine Übersicht über die Aufrufhierarchie der internen Prozeduren. Ein SPL Modul bestand – wie auch ein PL/I Modul aus einer Hauptprozedur und mehreren Unterprozeduren. Der Hauptprozedur war der einzige Eingangspunkt. Sie empfing den externen Aufruf mit Parametern und leitete die Steuerung weiter an die internen Unterprozeduren. Die internen Prozeduren konnten sich gegenseitig aufrufen. Deshalb war es wichtig, diese Aufrufstruktur bildhaft festzuhalten, damit die Tester die Interaktionen zwischen den Prozeduren verfolgen konnten.

Das Struktogramm zeigte für die Hauptprozedur sowie für jede Unterprozedur eines Moduls die logische Ablaufstruktur. Es war damals das Hauptinstrument um strukturierte Programmlogik darzustellen und war in Siemens weitverbreitet [NaSh73]. Jeder Entscheidungsknoten – ob IF Anweisung, Fallanweisung oder Wie-

derholungsanweisung – führte zu einer Aufteilung des Diagramms in weitere Spalten. Jede Spalte war ein Zweig. Die Aufeinander Reihung der Spalten ergab ein Pfad durch die Prozedur. Es war damit möglich, jeden Pfad von dem Eingang der Prozedur bis zum Ausgang zu verfolgen. Bald kam ich auf die Idee, mit Farbstiften die Pfade durch die Struktogramme mit verschiedenen Farben zu markieren. So gab es der Rote Pfad, der grüne Pfad, der blaue Pfad und so weiter. Auf diese Weise wurden die Ablaufpfade visualisiert und die Tester konnten sie im Test ansteuern und verfolgen. Deswegen musste ich bei meinen Reisen nach Budapest immer neue Farbstiften mitbringen – im sozialistischen Ungarn gab es sowas nicht. Später gegen Ende des Projektes haben wir diese Testpfadanalyse automatisiert.

3.5.2 Statische Analyse der Module

Nach der Dokumentation der Module folgte die Definition der Testfälle. Da wir auch nach Testfall bezahlt wurden, musste jeder Testfall dokumentiert sein. Dies geschah auf zweierlei Weise:

- statisch, durch die Angabe der Steuerungsvariabel und deren zugewiesenen Werte,
- dynamisch, durch den Trace des Ablaufs durch den Code

Die statische Programmanalyse hatte das Ziel, die statische Qualität der Programmstrukturen und den Code zu bewerten. Ausschlaggebend für die statische Qualität war die Einhaltung der Coderegel, die ich mit dem Projektqualitätsmanager, dem Dr. K., vereinbart hatte. Ein Großteil dieser Regel wurde von dem Prüfstand Werkzeug automatisch geprüft. Die verbleibenden Regeln wurden manuell von den Testern kontrolliert. Unsere Aufgabe bestand darin einen Bericht über die Codemängel, bzw. über die Regelverletzungen, zu erstellen und zusammen mit dem Bericht über die Programmfehler einzureichen. Die statische Qualitätsnote war gleich dem Konformitätsmaß, als Prozentsatz der Codeanweisungen ohne Mängel. Damit hatten wir begonnen 1978 technische Schulden zu messen, ein Thema das 30 Jahre später groß auf den Markt kommen sollte [KrNo12].

Bei der Analyse der Datenstrukturen ging es darum, die Anzahl der Ein- und Ausgabe-Variablen in jedem Modul zu ermitteln. Es war auch nötig zu wissen, woher die Daten kamen und wohin sie gingen, d.h. wir mussten die Art der Datenschnittstelle feststellen, ob

- Parameterliste,
- statisches COMMON Feld,
- based Feld oder

- Datei.

Bei Parametern musste der Tester ferner zwischen Argumenten von aufrufenden Modul und Ergebnissen vom aufgerufenen Modul unterscheiden. dargestellt wurde die Datenstruktur in Form eines Datogramms, ein von mir erfundenen Abbildung der Datenflüsse im Modultest [Sned78]. (siehe Abb. 3.?).

Anhand des Datogramms zeigte sich der Komplexitätsgrad des Moduls, bezogen auf seine Datenflüsse. Ein Modul dürfte in der Regel nicht mehr als insgesamt 10 Schnittstellen haben. Die Schnittstellen zur Peripherie, z.B. Nachrichten oder Dateien dürfen nicht mehr als 3 sein. Außerdem sollte kein Modul mehr als 100 Ein-/Ausgabe Variabel verarbeiten, wobei lokale Variabel nicht mitgezählt wurden.

Die Analyse der Ablaufstruktur diente im ersten Range dazu, Ablaufpfade zu erkennen. Wir haben dazu die aus dem Code generierten Struktogramme verwendet. Jeder Pfad war ein einmaliger Weg durch den gerichteten Graphen. Um den Weg zu gehen mussten die Weichenstellen richtig gesetzt werden. Die Weichenstellungen waren die Entscheidungspunkte. Entschieden wird aufgrund der Bedingungsoperanden die als Prädikate bezeichnet sind. Um die Ablaufpfade anzusteuern, musste der Tester nicht nur wissen wo sie anfangen und wo sie enden, er musste auch wissen welche Prädikate er setzen musste um auf dem gewünschten Pfad zu bleiben. Am Anfang wurden die Pfade und deren Prädikate manuell aus den Struktogrammen entnommen und in einer Testfalltabelle dokumentiert. Später ist es einer Doktorandin des Szamok Instituts gelungen ein Tool dafür zu bauen, so dass wir die Testfälle automatisch mit Testdaten aus dem Code ableiten konnten. Das war ein großer Durchbruch, aber er kam leider gegen Ende des Testprojektes. Der Hauptaufwand im Projekt blieb die Definition und Dokumentation der Modultestfälle.

3.5.3 Dynamische Analyse der Module

Nach der Definition der Testfälle folgte der Test selbst. Der Test der I.T.S. Module verlief interaktiv. Die Tester saßen in der SzKI an einem Siemens-BS2000 Bildschirm und testeten ein Modul nach dem anderen. Bei jedem neuen Testfall wurden sie vom Tool befragt, welche Eingabedaten sie setzen wollen. Dazu erschien eine Tabelle der Daten am Bildschirm, die aus dem Datenverzeichnis der statischen Analyse entnommen wurde. Die Tester wählten ihre Eingangsdaten aus und wiesen ihnen Testwerte zu. Die Testwerte entnahmen sie aus den Testfällen, die sie vorher spezifiziert hatten. Anschließend haben sie den Test gestartet. Bei jedem Kontrollwechsel, bzw. bei jedem Aufruf einer internen Prozedur sowie Aufruf eines anderen Moduls wurde der Ablauf angehalten. Die Tester bekamen eine Liste der veränderten Daten-

felder und konnten sie anzeigen lassen. Nach Bedarf konnten sie auch die Eingabedaten sehen und sie mit neuen Werten überschreiben. Dadurch konnten sie den Testablauf verändern.

Während die Tester den Test über die Benutzeroberfläche gesteuert haben, wurden intern vom Prüfstand alle durchlaufenen Zweige registriert und alle veränderten Datenfelder protokolliert. Dadurch wurde die Tabelle der aufgeführten Zweige ständig aktualisiert und das Protokoll der aktuellen Datenzustände immer weitergeführt. Als der Test endlich zu Ende war, entweder

- weil er den Modulausgang erreichte,
- weil der Tester ihn abbrach oder
- weil ein Programmfehler aufgetreten ist,

wurden diese beiden Tabellen ausgedruckt, die eine als Ablaufpfadbericht, die andere als Testergebnisbericht.

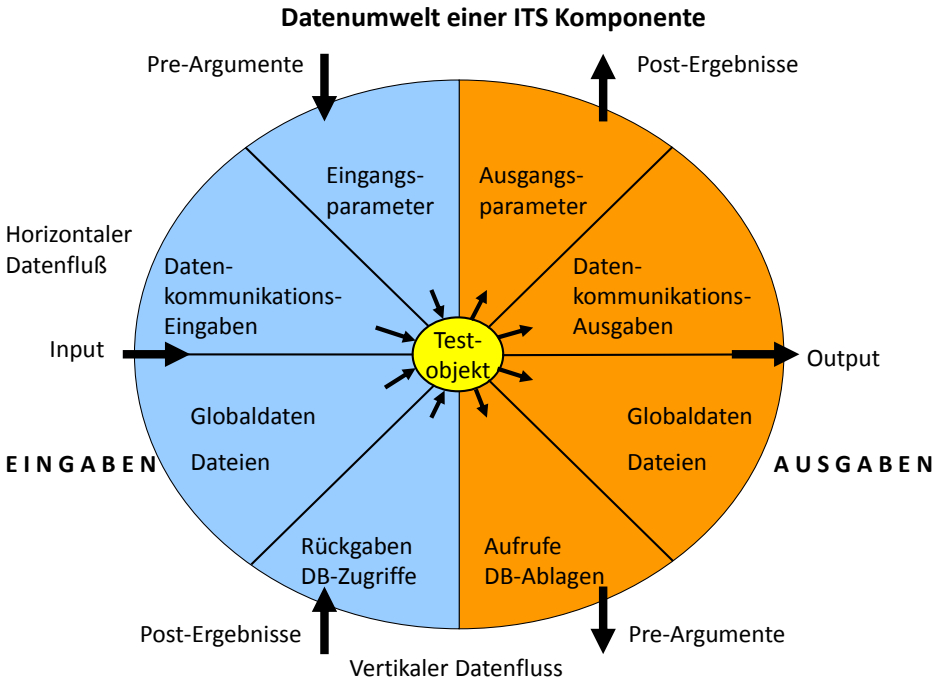
3.5.4 Testauswertung

Der letzte Schritt in dem automatisierten Modultest war die Testauswertung. Das Werkzeug fasste die Ablaufpfade zusammen in einem endgültigen Pfadbericht, aus dem ein Testüberdeckungsbericht erzeugt wurde. Dieser Bericht war für uns sehr wichtig, weil wir vertraglich verpflichtet waren, mindestens 85% Zweigüberdeckung zu erreichen. Der Ablaufpfadbericht diente dazu die ausgeführten Testfälle zu dokumentieren, ein Pfad durch das Testobjekt entsprach einem Testfall. Wir müssten die Testfälle dokumentieren um die Bezahlung nach Testfall zu erhalten. Der Testergebnisbericht war wichtig, um die Soll/Ist Abweichungen festzuhalten, auf Grund dessen die Fehler berichtet wurden. Die Fehlerberichte waren die wichtigsten Testergebnisse und wurden extra bezahlt.

3.6 Bilanz des I.T.S. Modultestprojektes

Schon bis Ende 1978 hat sich das Konzept des Testlabors bewährt. In den ersten sechs Monaten wurden 128 Moduln mit 60881 Anweisungen geprüft. Die 4378 Ablaufzweige wurden bis zu 89,7% abgedeckt. dazu waren 1544 Testfälle erforderlich. Aufgedeckt wurden 190 bestätigte Programmfehler und 1396 Entwurfs- und Codiermängel. Die meisten Fehler lagen in der Ablauflogik, gefolgt durch Fehler in den Datenschnittstellen. Es gab nur wenige Programmabbruchsfehler, weil die meisten Module durch die Entwickler selbst schon oberflächlich getestet wurden [Sned79]. (siehe Abb.3.?)

Sowohl die Programmfehler als auch die Entwurfs- und Codemängel wurden mit den Entwicklern in einer kooperativen Atmosphäre durchgesprochen und Korrekturmaßnahmen eingeleitet. Die Entwickler waren zuerst etwas skeptisch, aber im Laufe der Zeit akzeptierten sie die Kontrolle als eine nützliche Dienstleistung, die ihnen zusätzliche Sicherheit gab. Die Wirkung der Kontrolle war also insgesamt positiv und die Kosten vertretbar. Sie blieben unter DM 4,- pro Anweisung. Dies war weniger als 10% der Entwicklungskosten. Diese Erfahrungen führten zu dem Schluss, dass ein solches Software-Testlabor durchaus seine Berechtigung hat auch wenn nicht alle Fehler gefunden werden.



Im Herbst 1978 kamen zwei weitere Tester hinzu, das Testteam zu stärken. Die Produktion beschleunigte sich und bis Ende 1978 hatten wir 128 Module mit mehr als 60.000 Anweisungen und 4378 Ablaufzweige. Wir hatten 1544 Testfälle getestet, 192 bestätigte Programmfehler gemeldet und 1396 Codemängel aufgedeckt. Damit war die Siemens Projektleitung zufrieden und die Institutsleitung auch. Aus dieser Sicht war das Projekt ein Erfolg. Im Dezember 1978 ist die Testteam-Leiterin – Maria Majoros – nach Florida geflogen, um dort auf den ersten IEEE Testworkshop in Fort

Lauderdale über unser Testprojekt zu berichten. Die Amerikaner zeigten sich beeindruckt – so etwas, wie ein kommerzielles Testlabor hat es dort noch nicht gegeben. Unser Paper erschien in dem Workshop Proceedings und dieses Testprojekt wurde berühmt. Wie es sich später herausstellte, zu berühmt. Es fand in einem kommunistischen Land statt.

Anfang 1979 bekamen wir auch Verstärkung aus Amerika. Ed Miller schickte uns einen Doktoranden von der Universität California. Er sollte helfen, den Testprozess zu optimieren und die Tester zu schulen. Eigentlich kam er für beides zu spät. Der Testprozess war schon eingefahren und lief so gut, wie die damaligen Umstände es erlaubten. Die Tester waren bereits erfahren, waren aber offen für Verbesserungsvorschläge. Der Hauptbeitrag des Doktoranden war unser Projekt durch seine Veröffentlichungen noch bekannter zu machen [Budd80]. Später ist er einer der ersten Professoren für Testtechnologie in Amerika geworden.

Statistik des Testlabors im ersten Halbjahr

Comp	Modules	Stmts	Test cases	Branches	Coverage	Design & Code Defis	Program Errors
A	6	4029				138	5
B	37	7588	232	603	91%	130	22
K	71	40735	1064	2843	87.5%	868	143
N	6	2847	101	140	94%	110	14
S	8	5682	147	792	95%	150	6
Total	128	60881	1544	4378	91.9%	1396	190

Der Testbetrieb lief weiter zur Zufriedenheit beider Seiten. Bis Juli 1979 hatten wir weitere 120 Module getestet und 168 Fehler gemeldet. Das Einkommen blieb

unter den Erwartungen, aber es reichte für die Institute. Die große Enttäuschung kam im Sommer 1979 als die ersten Module in den Integrationstest kamen. Es sind etliche Integrationsfehler aufgetreten. Ich hatte das Siemens Projektmanagement schon gewarnt, dass ein bestandener UnitTest noch lange nicht heißt, dass die Software korrekt ist. Da wir keinerlei Spezifikation der Anforderungen hatten, mussten wir den Code gegen sich selbst testen. Probleme, die durch das Zusammenwirken der Module zustande kamen, konnten wir nicht testen, auch nicht Probleme, die in der Interaktion mit dem Endbenutzer auftraten. Von den Fehlern, die insgesamt im Test aufgedeckt wurden, haben wir nur 51% in dem Modultest erkannt. Der Rest wurde erst im Integrationstest bzw. im Systemtest durch die Siemens Integratoren gefunden. Die I.T.S. Projektleitung war etwas enttäuscht, sie hatte mehr von dem Modultest erwartet. Das war der erste Rückschlag, der zweite sollte bald folgen.

3.7 Rausschmiss von Siemens

Im Frühjahr 1979 habe ich ein Testseminar für Control Data in Frankfurt gehalten. Dabei habe ich das Testprojekt in Budapest geschildert. Was ich nicht ahnen konnte, es waren zwei Teilnehmer vom Schweizer Geheimdienst dabei. Sie haben mitbekommen, dass die I.T.S. Software in Ungarn getestet wurde und wollten mehr darüber erfahren, was sie auch erreicht haben. Kurz danach wurde ich zur Projektleitung gerufen. Siemens war in Verhandlung mit dem Schweizer Verteidigungsministerium über den Verkauf des I.T.S. Systems. Nachdem die Schweizer erfahren hatten, dass das Source Code des Systems durch ein sozialistisches Land gegangen ist, haben sie die Verhandlungen abgebrochen. Der Siemens Projektleiter hat sofort einen aufs Dach bekommen- dafür, dass er überhaupt so etwas angeleiert hat. Seine erste Reaktion war mich herauszuschmeißen. Ich musste meine Eintrittskarte abgeben und das Testprojekt wurde eingestellt.

Dies war zum Glück nur seine erste Reaktion. Beim weiteren Überlegen hat Herr H. sich das doch anders überlegt. In dem I.T.S. Projekt dürfte ich nicht weiterarbeiten, aber meine Testwerkzeuge waren zu wertvoll als das Siemens ganz auf mich verzichtet. Er vermittelte mich weiter an den Leiter der BS2000 Betriebssystementwicklung – einen Herrn D. Dort gab es bereits eine kleine Testgruppe aber sie war total überfordert. Ich sollte dort das Prüfstand Werkzeug einführen und die Gruppe mit ungarischen Testern unterstützen. Das Problem hier war, dass die Betriebssystemsoftware in Assembler geschrieben war. Siemens hatte das Betriebssystem von der amerikanischen Firma RCA übernommen einschließlich einiger amerikanischen Entwickler. Da die Siemens Entwickler mit dem Code noch nicht so vertraut waren, gab es natürlich zu Beginn der Weiterentwicklung durch Siemens jede Menge Software Fehler. Mit

dem Prüfstand Tool konnten sie den Code besser durchleuchten. Da einige I.T.S. Module auch in Assembler verfasst waren, hatten wir das Tool schon in Richtung Assembler erweitert. Jetzt bekam ich zwei erfahrene Assembler Programmierer von der SzKI zugeteilt. Sie sollten vor Ort in München arbeiten und die Siemens Tester beim Modultest unterstützen. Dafür wurden wir jetzt Stundenweise bezahlt. Ich bekam auch ein Honorar als Testberater.

Herr H. hat mir auch geholfen bei der Firma Quelle in Nürnberg einzusteigen. Quelle war dabei ihre Anwendungssoftware in einer Makro-Assembler Sprache zu entwickeln und hatte Probleme die Module zu testen. Der Quelle IT-Leiter war ein alter Studienfreund von Herrn H. und interessierte sich sehr für den Prüfstand. Er wollte so ein Werkzeug für seine Entwickler haben. Ich machte dort eine Präsentation und sie waren sofort bereit die Toolentwicklung zu finanzieren, wollten allerdings einen festen Preis dafür haben. Wir sind einig geworden, dass sie in Stufen bezahlen und im Herbst 1979 begann das QSTS - Quelle Software Testsystem – Projekt in Nürnberg. Zwei Mitarbeiterinnen von der Szamok und eine von der SzKI waren für das Projekt vorgesehen. Die Hälfte der Zeit sollten sie in Nürnberg arbeiten und dort an dem IBM Rechner testen.

Es tat dem Herrn H. scheinbar leid dass er mich entlassen musste, denn neben der Vermittlung an die Siemens Betriebssystementwicklung und der an die Quelle AG, vermittelte mich auch in ein anderes Siemens Projekt. Die Telefonbranche war dabei, ein digitalisiertes Telefonsystem – das EWSD – elektronische Wahlsystem für Deutschland einzuführen. Ein Teil der Software und die Programmiersprache hatten sie von der schwedischen Telekom übernommen. Die Siemens Leute – alle umgeschulte Telekommunikationsingenieure - sollte die Software anpassen und weiterentwickeln. Sie war in der Telekommunikationssprache CHILL – eine Erweiterung von PASCAL – geschrieben. Ich sollte die CHILL Module statisch analysieren und in einen Modultestrahmen testen. Es war wieder eine große Herausforderung. Ich musste mich in eine neue Programmiersprache und in ein fremdes Anwendungsgebiet einarbeiten. Auch dieses Projekt begann im Herbst 1979. Es gab 26 Regeln gegen die ich den CHILL Code checken sollte. Die Nachrichtentechniker haben zu jedem Modul ein Struktogramm gezeichnet und zwar mit Tinte. Ich habe die Ablauflogik des Codes mit der in dem Struktogramm verglichen und Abweichungen notiert. Das Ganze lief unter der Rubrik „Code Inspektion“ [Faga76]. Auf meiner Amerikareise in 1977 hatte ich das IBM Federal Systems Labor in Washington D.C. besucht und habe mir zeigen wie sie dort die Programme und Dokumente kontrollieren. Jetzt hatte ich die Gelegenheit diese Technik selber anzuwenden. Da ich allein war und alles manuell ma-

chen musste, war diese Arbeit sehr zeitaufwendig. Es war in dieser Zeit als ich begonnen habe 60 Stunden der Woche zu arbeiten.

3.8 Probleme mit dem Staatssicherheitsdienst in Ungarn

Ungefähr zu gleichen Zeit, als ich in Deutschland von Siemens einen Dampfer erfuhr, bekam ich in Ungarn von der anderen Seite eine Watsche. Eine Mitarbeiterin der SzKI, die an dem Quelle-Testprojekt beteiligt war, wollte mich mit einem Reporterfreund von ihr zusammenbringen. Um Publizität für mein Vorhaben in Budapest zu gewinnen, und weil mir die junge Frau nicht unsympathisch war, habe ich mich darauf eingelassen. Auch das war ein entscheidender Fehler, den ich bald zu bereuen bekam.

Wir trafen uns, der Reporter, die Kollegin und ich eines Winterabends in dem berühmten Café Hungaria. Beim Kaffee und Kuchen habe ich über meine ehrgeizigen Pläne in englischer Sprache erzählt und von den bürokratischen Hindernissen, die mir seitens der staatlichen Institute in den Weg gelegt wurden. Das Haupthindernis für mich war das mangelnde technische Verständnis und das Unvermögen des mittleren Managements. Gerade im mittleren Management gab es immer wieder Personen, die ihre Position aufspielten, wichtige Vorhaben, die sie gar nicht verstanden, zu verhindern. Das gleiche hätte ich über fast jeden größeren deutschen Betrieb sagen könne. Es viel ein Begriff, einer der wenigen ungarischen Begriffe die ich damals kannte – Tisztviselő – der Beamte. das Mittelmanagement der ungarischen Institute bestand nur aus „Tisztviselő“, die von der Sache keine Ahnung hatten.

Damit lieferte ich dem Reporter genau das, was er hören wollte. Er suchte nämlich Stoff bzw. die Meinung eines ausländischen Experten um den schwerfälligen Beamtenapparat in den staatlichen Betrieben zu diskreditieren. Was ich nicht wissen könnte, war dieser Reporter auch in einer Dissidentenbewegung aktiv und stand schon unter Beobachtung. Sein Artikel mit dem Titel „Mein Kapital steckt in den Köpfen meiner Mitarbeiter“ erschien zwei Wochen später in der ungarischen Wirtschaftswochenzeitschrift [HVG80]. Ich bekam eine Ausgabe und merkte sofort, obwohl ich nicht alles verstand, dass dies nicht das war, was ich erzählte. Der Reporter hatte meine Aussagen über die Außenhandelsgesellschaft und die beiden Partnerinstitute in eine allgemeine Kritik des Regimes umgewandelt. Meine Mitarbeiter in den Instituten waren begeistert und haben mich gelobt, es dauerte aber nicht lange und ich wurde auf die Polizeistation zitiert. Dort wurde mir von einem Polizeibeamten in Zivil in gebrochenem Deutsch mitgeteilt, dass ich eine schwere Sünde begangen habe, mich mit so einem polizeilich bekannten Dissidenten einzulassen und dass ich nie wieder mit Reportern sprechen darf. Sonst würden sie sich gezwungen sehen, mich des Lan-

des zu verweisen und nie wieder einreisen zu lassen. Eingeschüchtert wie ich war, habe ich mein Wort gegeben und habe mich lange Zeit nicht mit ungarischen Reportern eingelassen. Erst 1985, beim Anlass einer Konferenz für Softorg-Benutzer aus Westdeutschland habe ich mich mit der lokalen Presse wieder eingelassen. Inzwischen hatte sich in Ungarn vieles geändert.

HETI VG **INTERJÚ**

VÁLASZOL A SZÁMÍTÁSTECHNIKUS-MANAGER

„Az én tőkém magyar munkatársaim fejében van"

Harry Sneed nem tesz mást, minthogy megpróbál magának valamit „összeadni" a világgazdaság nagy különbségeiből: az USA és Nyugat-Európa közötti technológiai különbségből, illetve a Nyugat- és Kelet-Európa közötti bérszínvonal-különbségből. A tulajdonában lévő Software Research Associates nevű müncheni cég rajta és titkárnőjén kívül 10 magyar agyat foglalkoztat, amelyeknek „tulajdonosai" egyébként két magyar intézmény, a SZÁMOK (Nemzetközi Számítástechnikai Oktató és Tájékoztató Központ) és az SZKI (Számítástechnikai Koordinációs Intézet) alkalmazottai. A cég technológiáját az NSZK-ban élő, 42 éves programozási szakember szülőházojából, az USA-ból hozta magával. Harry Sneed a marylandi egyetemen tanult számítástechnikát. Európába áttelepülve, négy évet dolgozott a Volkswagen számítóközpontjában és ugyanennyit a Siemensnél. 1977-től önálló tanácsadó, 1978 tavaszán pedig saját tőkével alapított önálló céget.

HVG: Hogyan jött létre a Software Research Associates, és hogyan került Ön kapcsolatba magyar számítástechnikusokkal?

H. S.: A két dolog lényegében egyszerre történt, s természetesen a véletlen műve. 1978 februárjában Londonban egy konferencián előadást tartottam egy tesztelési, azaz a számítógépprogramok ellenőrzését célzó módszerről, amellyel addig csak elméletben foglalkoztam. Mellettem egy magyar számítástechnikus ült, aki megkérdezte, nem lenne-e kedvem módszeremet a gyakorlatban is alkalmazni intézetével, a SZÁMOK-kal együttműködve. Márciusban már Budapesten tárgyaltam a részletekről, néhány héttel később saját tőkémmel megalapítottam a Software Research Associates (SRA) müncheni testvérvállalatát és aláírtam egy szerződést a Metrimpex magyar külkereskedelmi vállalattal, amelynek értelmében az SZKI és a SZÁMOK öt-öt rendszerprogramozója nekem dolgozik, munkaidejük felét saját munkahelyükön, másik felét pedig az én NSZK-beli megbízóimnál töltve.

Az SRA egyébként három független testvérvállalatból áll. Az egyik San Franciscóban van, vezetője dr. Miller, aki az általunk is alkalmazott tesztelési eljárás szellemi atyja, és mindössze hét embert, valamint egy miniszámítógépet foglalkoztat. Tokióban van a másik testvérvállalat 200 munkatárssal és egy közepes méretű IBM számítógéppel. Végül itt vagyok én az SRA nyugatnémet—amerikai—magyar testvérvállalatával — legalábbis én ezt a megjelölést használom a legszívesebben — 10 magyar munkatárssal, egy adminisztrátorral és egy kis müncheni irodával, valamint egy IBM és egy Siemens számítógéppel a SZÁMOK-nál, illetve az SZKI-nél.

HVG: Olvasóink között kisebbségben vannak azok, akik pontosan tudják, hogy mit is jelent a programok tesztelése, vagyis, hogy mit csinál az Ön cége.

H. S.: A tesztelés tulajdonképpen nem más, mint a számítógépprogramok minőségi ellenőrzése. Ha például egy vállalat számítógépet használ raktárkészlet-nyilvántartásra, könyvelésre vagy bármilyen más feladatra, akkor maga viszonylag könnyen elkészítheti, vagy megvásárolhatja ezeket az úgynevezett felhasználói programokat. Annak megállapítása azonban, hogy a programok valóban jól működnek-e, vagy hogy programozási szempontból az adott feladat legoptimálisabb megoldásai-e, már egészen speciális számítástechnikai feladat. Ezzel természetesen a programokat készítő nagy software-házak is foglalkoznak, de mi vagyunk Európában az első független — azaz csak ellenőrzéssel foglalkozó — tesztlaboratórium.

Az általunk alkalmazott, több évi kutatómunkával kifejlesztett tesztelési eljárás pillanatnyilag a legkorszerűbb a világon, jobb mint amit az IBM kínál. Nem véletlen, hogy az IBM is megbízott minket egy tesztelési rendszer kidolgozásával.

HVG: Milyen egyéb megbízásokat kap az SRA?

H. S.: Először a Siemens számára dolgoztunk ki egy tesztelési rendszert, az úgynevezett Prüfstandot — ez ma a legismertebb referenciánk. A rendszert a Siemens el akarta adni a svájci hadseregnek is, de amikor ez utóbbi tudomására jutott, hogy a munka részben Magyarországon, magyar közreműködőkkel készült, nemcsak hogy elállt a szerződéstől, de olyan patáliát csapott, hogy a Siemens legfelső vezetői is elbizonytalanodtak: valóban nem olyan technológia kiszivárgásáról van-e itt szó, amelynek nem szabadna Kelet-Európába jutnia. Ezek után mindenesetre hónapokig be sem tehettem a lábam a Siemens Művek területére. Azóta napirendre tértek a dolog fölött, és jelenleg is dolgozunk a Prüfstand tökéletesítésén. Emellett készítünk egy tesztelési rendszert a Quelle áruházi konszern számára, valamint a Kienzle nyugatnémet kisszámítógépgyártó cégnek a gépeihez adott programok tesztelésére. Megbízást kaptunk a Német Szövetségi Postától is ellenőrző programok készítésére.

HVG: Ön rendszeresen ingázik München, illetve más nyugatnémet városok és Budapest között. Szinte az aktatáskájában hordja az irodáját. Emellett megbízói nem is veszik jónéven, hogy magyarokkal dolgoztat. Mindezeket a hátrányokat feltehetően igen komoly előnyök egyenlítik ki.

H. S.: Szakmámbeliek az NSZK-ban is gyakran szegezik nekem a kérdést: miért nem alkalmazok nyugatnémet számítástechnikusokat, mint bármelyik más hasonló profilú cég — úgynevezett software-ház — az NSZK-ban. A válasz kézenfekvő: olcsóbb havonta egyszer Budapestre jönni, mint Münchenben egy irodát fenntartani. Ez utóbbi havonta háromezer márkámba kerülne, nem is szólva a számítógép-kölcsönzésről, ami évente legalább 200 ezer márka lenne. Ugyanakkor havonta egyszer mindössze 750 márkáért repülök ide és vissza, és ráadásul Pesten jóval olcsóbban élek, mint az NSZK-ban. Munkatársaim saját munkahelyük számítógépén dolgoznak, a SZÁMOK-nál IBM, az SZKI-nél Siemens gépen, éppen azon a két típuson, amelyeket megbízóink döntő többsége is használ. Egy úgynevezett rendszerprogramozóért — ők végzik a magasan kvalifikált programozói munkát — 60 márka órabért fizetek a Metrimpexnek, ugyanennek a munkának az árfolyama az NSZK-ban körülbelül 100 márka. A számítógép mellett töltött órákért további 17,5 márka géphasználati díjban egyeztünk meg.

Ilyen feltételekkel valamivel jobb versenyhelyzetben vagyok nyugatnémet konkurrenseimnél, amire viszont az említett hátrányos megkülönböztetés miatt szükségem is van. Az üzlet természetesen a Metrimpexnek is előnyös. A magyar programozók munkájából származó bevételeim 75 százalékát fizetem ki magyar partneremnek, ez évente körülbelül 1 millió márka. A bevétel fennmaradó 25 százalékát teljes egészében fejlesztésre, azaz olyan tesztelési rendszerek kidolgozására fordítom — mint például az IBM gépekre készülők —, amelyekért egyelőre egy vasat sem kapok, de elkészítésükkel mintegy elébe vágunk a megrendelői igényeknek.

Der Artikel in der Weltwirtschaftswoche hat mir eine Lehre erteilt, eine Lehre die mir auch in der freien westlichen Welt zu Gute kam. Vertreter der Presse kann man nicht trauen. Meistens haben sie eine vorgefasste Meinung zu einer Sache und suchen

nur Stoff um diese Meinung zu untermauern. Sie können die Worte immer so zu Recht zu schneiden, dass man sagt, was die Reporter sagen wollen. Man muss also vorsichtig mit ihnen umgehen und darauf bestehen, ihren Artikel vor der Veröffentlichung vorzulegen. Dies gilt auch für die Computer Fachpresse. Auf der einen Seite braucht man sie, um sein Vorhaben publik zu machen, auf der anderen Seite muss man aufpassen, dass sie die Sache richtig darstellen. Sonst wird man von ihnen eher geschädigt als begünstigt.

3.9 Das unrühmliche Ende des I.T.S. Projektes

Das I.T.S. Projekt war für mich zu Ende. Stattdessen hatte ich Dank Herrn H. drei andere Projekte:

- der Test der BS2000 Kernmodule für die Siemens Betriebssystementwicklung,
- die Entwicklung eines Modultestsystems für die Quelle Anwendungsentwicklung und
- die Prüfung der EWSD Telekommunikationsmodule für die Siemens Nachrichtentechnik.

Mit dem I.T.S. Projekt war es letztendlich für mich Glück da herauszukommen, denn Ende 1979 wurde das Projekt abgebrochen. Es hat sich herausgestellt, dass der Siemens BS2000 Rechner viel zu schwach war um die Rechenlast zu tragen. Nicht nur das, die dreischichtige Software Architektur war viel zu umständlich. Es dauerte ewig, bis eine Transaktion von dem Endbenutzer bis zur Datenzugriffsschale ankam. Die Idee hinter dem Schichtenmodell war im Prinzip richtig, aber zu früh für die damalige Rechenarchitektur. I.T.S. war das erste gescheiterte Großprojekt im öffentlichen Bereich. Es sollten andere folgen. Laut einem Bericht des Spiegels musste 20 Million DM von der Bahn abgeschrieben werden. Es sollte nicht das letzte Pannenprojekt sein. Im Verhältnis zum späteren FISCUS Projekt war dieser Verlust ehe Peanuts. Durch I.T.S und dem parallel laufende START Projekt für die Reisebranche ist eine deutsche Softwareindustrie entstanden, die es bisher nicht gegeben hat. Viele namhafte deutsche Softwarefirmen haben ihren Anfang in diesen beiden Projekten gehabt, unter anderem SoftLab und SD&M. Dem I.T.S. Projekt habe ich auch meine Karriere als Software Testexperte zu verdanken.

4 Die SoftOrg Toolentwicklung in Ungarn

4.1 Der Software Lebenszyklus zu Beginn der 80er Jahren

Die Erfahrung mit den I.T.S. Test hat gezeigt, dass ein Test ohne Orakel nicht ausreichend ist. Er bestätigt lediglich, dass ein Programm oder ein System ablauffähig ist und Ergebnisse produziert. Ob diese Ergebnisse richtig sind oder ob die Software so abläuft wie sie ablaufen sollte, kann niemand wissen. Nur im Bezug zu einem Orakel lassen sich das Verhalten eines Systems und die Gültigkeit seiner Ausgaben beurteilen [Howd87]. Deswegen wurden nur knapp die Hälften aller Fehler in der I.T.S. Software durch den Modultest in Budapest aufgedeckt. Die restlichen 49% der Fehler konnten nicht erkannt werden, weil es keine Spezifikation der getesteten Komponente gab. Wenn die Spezifikation als Orakel fehlt, kann nur ein Applikationsexperte beurteilen, ob die Software unter Test sich richtig verhält, Fremde Tester können nur erahnen, was richtig und was falsch ist.

Es gab mehrere Gründe für das Scheitern des I.T.S. Projektes, aber einer der Hauptgründe war das Fehler einer vollständigen Anforderungsdokumentation. Keiner wusste, was das System leisten sollte. Jeder hatte seine eigene Interpretation. So kam es, dass am Ende keiner mit dem Ergebnis zufrieden war. Die Ziele waren unzulänglich definiert. Als ich mich gezwungen sah die Werkzeugentwicklung für die ungarische Außenhandels-Gesellschaft weiterzuführen, wollte ich dieses Versäumnis nicht wiederholen. Um die Voraussetzungen für weitere Testprojekte zu schaffen, musste als erstes ein Tool für die Spezifikation der Anforderungen geschaffen werden. Die damit erstellte Spezifikation bzw., Blaupause sollte nicht nur als Leitlinie für die Entwicklung, sondern auch als Vorgabe für den Test dienen. Das Ziel war – wie in dem amerikanischen Ballistic Missile Defense System – die Programme gegen die Spezifikation zu testen. In dem BMD Projekt war die RSL – Requirement Specification Language – die BaseLine für die Implementation und das Orakel für den Test [Alfo77]. Die Testtechnologie war in den 70er Jahren hauptsächlich auf den Code fokussiert. Code-Testüberdeckung und Testdatenmutation standen im Mittelpunkt der Testforschung [Huang75]. Eine Veröffentlichung von Bill Howden mit dem Titel „Functional Testing“ brache eine neue Sicht auf die Testproblematik. Howden führte den Begriff „Testorakel“ ein [Howd80]. Functional Testing setzte auf ein Orakel, nämlich auf die funktionale Spezifikation. Ich habe mir vorgenommen, diese Theorie

in die Praxis umzusetzen. Das Tool SoftSpec sollte als Grundstein für eine allumfassende Software Entwicklungsumgebung dienen.

Noch während des I.T.S. Projektes habe ich erkannt, dass große Software Systeme in einem wohldefinierten Prozess entstehen müssen und dass dieser Prozess durch automatisierte Werkzeuge unterstützt werden muss. Im Mittelpunkt der Entwicklung sollte eine große Datenbank stehen, in der alle Schnittstellenbeschreibungen einschließlich Benutzeroberflächen, Berichte, Import/Export Dateien und APIs zusammengefasst sind. Die Tools sollten um diese große Projektbibliothek herum gebaut werden. Über die Tools wird die Bibliothek bevölkert und aus der Bibliothek werden weitere Software-Artefakten generiert, z.B. Programme und Testprozeduren. Die Entstehung und Fortentwicklung eines Software Produktes wurde damals gemäß dem Wasserfallmodell in Phasen aufgeteilt. Am Anfang kam die Analysephase in der das Problem analysiert und die fachliche Lösung mit echten Datenbanken, Schnittstellenmustern und Programmentwurfsdokumenten umgesetzt wird. Erst nach dem die Systemarchitektur steht und alle Programmbausteine spezifiziert sind, beginnt die dritte Phase der Programmierung. Dort sollten soweit wie möglich die Programme aus den Entwurfsmodellen automatisch abgeleitet werden [McClu89].

Die Rückseite des Lebenszyklusmodells mit den Testphasen habe ich aufgrund meiner Erfahrung mit Testen vom Anfang an vorgesehen. Später wurde das gleiche Modell von der IABG übernommen und in das V-Modell umgetauft [Bröh95]. Als erstes sollte der generierte oder manuell erstellte Code geprüft und nachgebessert werden. Diese war die Bausteinreparatur- und Anpassungsphase. Ich nannte sie die Nachbearbeitungsphase. Darauf folgte eine Modultestphase in der alle Codemodule durch einen strengen Filter passieren müssen. Hier sollte sie von Codierfehler bereinigt werden. In der letzten Phase wurden die Codebausteine integriert und gegen die Spezifikation getestet. Testfälle sollten aus den Funktionsspezifikationen, Funktionsbäumen und Entscheidungstabellen abgeleitet während Testdaten aus dem Datenmodell und den Schnittstellenbeschreibungen generiert werden.

Nach Abschluss der Integrationsphase wird die fertige Software als Ganzes oder als Teile an die Produktion ausgeliefert. Dann beginnt der Zyklus wieder von vorne – die Spezifikation wird ergänzt und erweitert, der Entwurf überarbeitet und neuer Code generiert. Ich habe also schon damals 1980 die Softwareentwicklung als iterativen Zyklus gesehen [Sned80]. Was ich nicht vorgesehen habe, war wie die alten Systemartefakte fortschreiben und mit den neuen Systemartefakten integriert werden sollten. Dieses konzeptionelle Versäumnis blieb bis zum Ende das Hauptmanko der

automatisierten Softwareentwicklung und sollte mir in späteren Projekten zum Verhängnis werden.

Über allen Software Entwicklungsphasen hinweg, bzw. in der Mitte des Entwicklungszyklus habe ich eine Prozesssteuerung vorgesehen- das Projektmanagement. Das Projektmanagement war keiner der genannten Phasen zuzuordnen. Es lief parallel zu den Phasen. Ein Projekt habe ich damals als einen Durchgang durch den Lebenszyklus definiert. Das Projektmanagement steuert den Durchgang durch die Phasen mit der Auswahl der zu erfüllenden Funktionen, der Definition der Qualitätsziele, der Festlegung der Termine und der Zuteilung der Ressourcen. Es bestimmt also das *was*, das *wie*, das *warum* und das *womit*? Mit der zentralen Produktbibliothek, bzw. Repository, hatte das Projektmanagement Zugriff zum aktuellen Stand aller Softwarebestandteile und konnte ihren Zustand verfolgen. Es war immer auf dem aktuellen Stand und nicht auf Berichte der Mitarbeiter angewiesen. Nur so dachte ich konnte ein großes Projekt erfolgreich gesteuert werden. Für mich war Transparenz das wichtigste Kriterium. Leider hat diese Managementkomponente in dem BMD Lebenszyklusmodell gefehlt, so dass ich in dieser Hinsicht kein richtiges Vorbild hatte. Sonst hätte ich diese Komponente früher konzipiert. Mit der SoftMan Entwicklung begann ich erst 1983. Da die Funktionalität des Tools mehrfach umdefiniert wurde, dauerte es bis 1986 bis die erste Version betriebsbereit war [Sned87].

Das SoftOrg Vorgehensmodell hatte ich inzwischen in diversen Artikeln über Software Entwicklung und Qualitätssicherung bis zum Jahr 1980 skizziert und im Jahre 1980 erfolgte das Buch „Software-Entwicklungsmethodik“ im Rudolf Müller Verlag, Köln [Sned80]. Dieses dritte Buch schrieb ich bruchweise zwischen Projekten, Seminaren und Beratungstermine unter den Druck des monatlichen Überlebenskampfs mit all seinen Rückschlägen und der erzwungenen Gründung einer neuen Firma – Die SES. Es ist ein Wunder, dass es je erschien, aber ich war besessen mit der Idee einer allumfassenden Software-Entwicklungsumgebung für die deutschen Anwenderbetriebe. Ich träumte sogar davon, die Kluft zwischen der deutschen und der amerikanischen Softwaretechnologie zu schließen. In meinem Newsletter propagierte ich immer wieder dieses Ziel. Zumindest auf diesem Gebiet sollte Deutschland gleichwertig sein. Allerdings war das SoftOrg Konzept nur ein erster Schritt in einer langen Reise, die noch 10 Jahre dauern sollte.

SOFTWARE ENGINEERING NEWSLETTER

SOFTING

ZUR FÖRDERUNG DER DEUTSCHEN SOFTWARE-TECHNOLOGIE

Heft 1 Juli 1980

INHALTSVERZEICHNIS

Software Engineering Services GmbH
Quiddestraße 78, 8000 München 83, Tel.(089)670 72 58

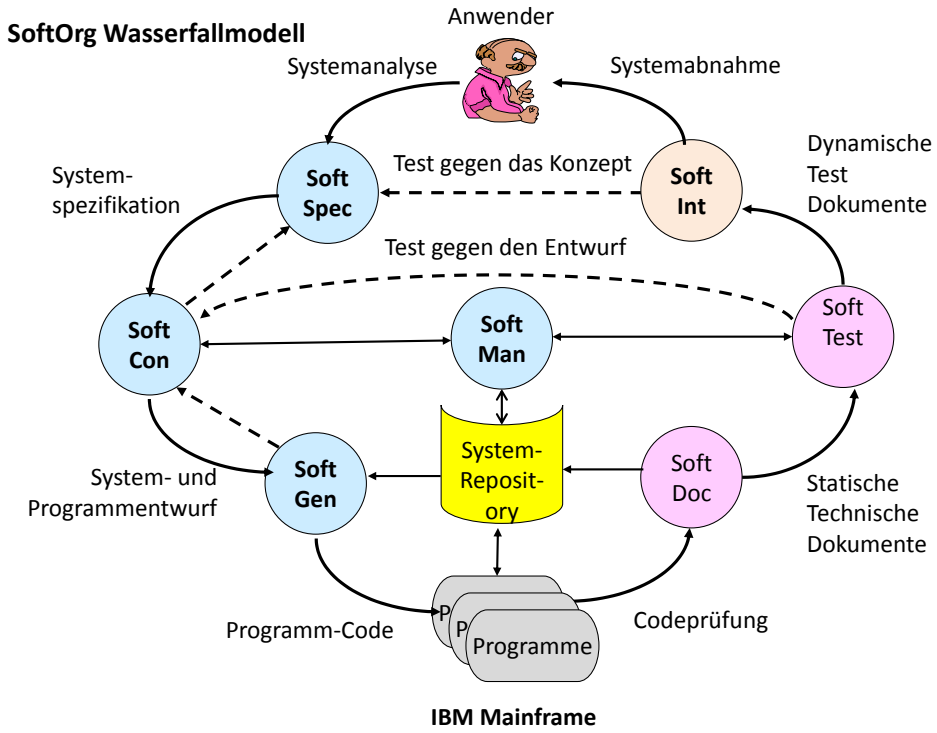
4.2 Das SoftOrg Projekt wird aufgestellt

Es war vom Anfang an meine Absicht, die Werkzeugentwicklung auf die beiden Institute SzKI und SzAMOK aufzuteilen. In dem SzKI mit seinem Siemens BS2000 Rechner sollten die Frontend-Tools entwickelt werden, weil dazu ein Dialogbetrieb erforderlich war. Mit dem Siemens Time Sharing System konnte man interaktiv arbeiten. Dort sollte die Spezifikation der Anforderungen und den Entwurf der Implementierung stattfinden. Später kann die Projektplanung- und Steuerung hinzu. In dem SzAMOK mit seinem IBM Rechner sollten die Qualitätssicherungswerkzeuge, die im Batchbetrieb laufen sollten, gebaut werden. Auf dem IBM Rechner gab es damals kein eingebauter Dialogbetrieb. Anwender mussten entweder mit CICS oder mit IMS-DC ihre Online Applikationen implementieren. SPF/TSO war noch nicht verfügbar. Demzufolge habe ich die Testwerkzeuge als Batchprogramme konzipiert, so wie wir es in dem Quelle Testprojekt zu dieser Zeit erfolgreich praktiziert haben. Die Quelle Software System war neben dem Prüfstand mein Prototyp für die Entwicklung der Testwerkzeuge.

Beachtungswert ist, dass ich zu diesem frühen Zeitpunkt dem Test die gleiche Bedeutung wie der Entwicklung beigemessen habe. Das folgte aus meiner Erfahrung bei Siemens, wo ich gelernt habe, dass der Test nahezu 50% des Projektaufwandes beansprucht. Alles, was neu entwickelt wird – jede Codezeile und jedes Datenelement – muss auf seine Richtigkeit geprüft werden. Es ist davon auszugehen, dass Software fehlerhaft ist, solange sie durch den Test nicht bestätigt wird. In dieser Hinsicht war ich meiner Zeit weit voraus. Ich wusste schon von der Bedeutung des Softwaretests. Ich der Industrie setzte sich diese Erkenntnis nur langsam durch [BoMe82].

Die Teilung des SoftOrg Projektes zwischen den beiden Instituten brachte einige Nachteile mit sich, vor allem, was die Schnittstellen zwischen den Tools anbetraf. Wir kamen nur einmal im Jahr in einem sozialistischen Betriebsferienheim für einige Tage zusammen, um die Zusammenarbeit zu besprechen. Das war immer recht unterhaltsam. Jedes Team trug über den Stand ihres Produktes vor und ich gab die Richtung für die kommende Zeit vor. Später habe ich meine beiden Söhne mitgenommen und sie konnten sich dort austoben. Abgesehen von diesen jährlichen Treffen haben sich die Mitarbeiter der beiden Institute sich nur bei den Kunden in Deutschland getroffen. Beim Bertelsmann und BMW sowie später beim Thyssen-Stahl saßen sie im selben Raum zusammen und konnten sich austauschen. Ansonsten behielt ich die Zügel in der Hand und leitete das Projekt über schriftliche Spezifikationen.

Die Teilung des Projektes auf die beiden Recheninstitute war auch aus der Not geboren. Keins der Institute wäre allein bereit mir so viel Mitarbeiter bereitzustellen. Somit war das Risiko geteilt. Ich hatte auch den zusätzlichen Vorteil, dass ich die beiden konkurrierenden Staatsbetriebe gegen einander ausspielen konnte. Falls ein Institut für eine Aufgabe zu viel verlangte, konnte ich mit dem anderen Institut drohen. Ich konnte die Konkurrenzsituation für mich ausnutzen. Projektmanagement heißt schließlich Menschen dazu zu bewegen, etwas zu tun, was sie sonst nicht tun würden. Im Westen wurde das Lockmittel Geld dazu benutzt. Im Osten zog diese Karte nicht. Man musste auf andere Mittel zurückgreifen.



Während der SoftOrg Entwicklung versuchte ich mindestens jede zweite Woche für 2 bis 3 Tage in Budapest zu sein. Die Pförtner der beiden Institute kannten mich gut und sprachen mich immer höflich mit Sneed elvtárs – Genosse Sneed – an. Sie wussten, Genosse Sneed war dort um ihnen zu helfen, den Sozialismus aufzubauen. Ich hatte ein Zimmer unterhalb des Budaer Burg gehabt und ging abends am Donauufer joggen oder im Burgviertel spazieren. Die Tage dort waren für mich eine Erholung von den stressvollen Projekttagen in Deutschland. Die Abende verbrachte ich

mit Ungarisch-stunden und morgens schrieb ich Artikel für unser Newsletter und Papers für die Konferenzen. Dieses Pendeln zwischen zwei Welten war recht interessant aber sehr anstrengend. Für die Flüge mit den alten russischen Maschinen der ungarischen Fluggesellschaft hatte ich Sonderbedingungen aber sie waren manchmal aufregender als es mir lieb war, vor allem, wenn ein Motor ausfiel. Jedenfalls war es für mich eine aufregende Zeit.

4.3 Die Entwicklungswerkzeuge – von den Anforderungen zum Code

Zur Vorderseite des Software Lebenszyklus gehörten die Werkzeuge SoftSpec, SoftCon und später SoftGen. Sie entsprachen den drei Stufen von der Analyse bis zum lauffähigen Programm.

4.3.1 SoftSpec für die Analyse und Dokumentation der Anforderungen

Das SoftSpec Modell für die Erfassung der Anforderungen stand schon zu Beginn der SoftOrg Entwicklung im Jahre 1980. Für das fachliche Modell ist man frei von irgendwelchen rechnertechnischen Einschränkungen. Ich konnte das Modell so gestalten, wie ich es für richtig hielt. Ich legte großen Wert auf die Symmetrie, also teilte ich es in zwei Seiten – Daten und Funktionen – mit jeweils zwei Granularitätsstufen – grob und fein. Das ergab vier Quadrate. In dem groben Datenquadrat wurden die Datenentitäten bzw. Objekttypen festgelegt. In dem feinen Datenquadrat wurden die Datenelemente bzw. Attribute definiert. Jedes Elementardatum hatte einen Typ, eine Bedeutung und mehrere Zustände die in einer Assertion festgehalten wurden. Es hat zwei verschiedene Objekttypen gegeben – interne Objekte entsprechend den Data-Stores in der strukturierten Analysemethode von Yourdan und DeMarco [DeMa78], sowie Schnittstellenobjekte, entsprechend den Interfaces in Structured Design von Constantine [Cons74]. Dazu gehörten Benutzeroberflächen und Listen sowie auch Nachrichten und Import/Export Dateien. Die internen Objekte wurden als Datenbäume, die Schnittstellen als Belege dargestellt. Die Funktionsstruktur wurde im Sinne der Jackson Methode aus der Datenstruktur abgeleitet [Jack75].

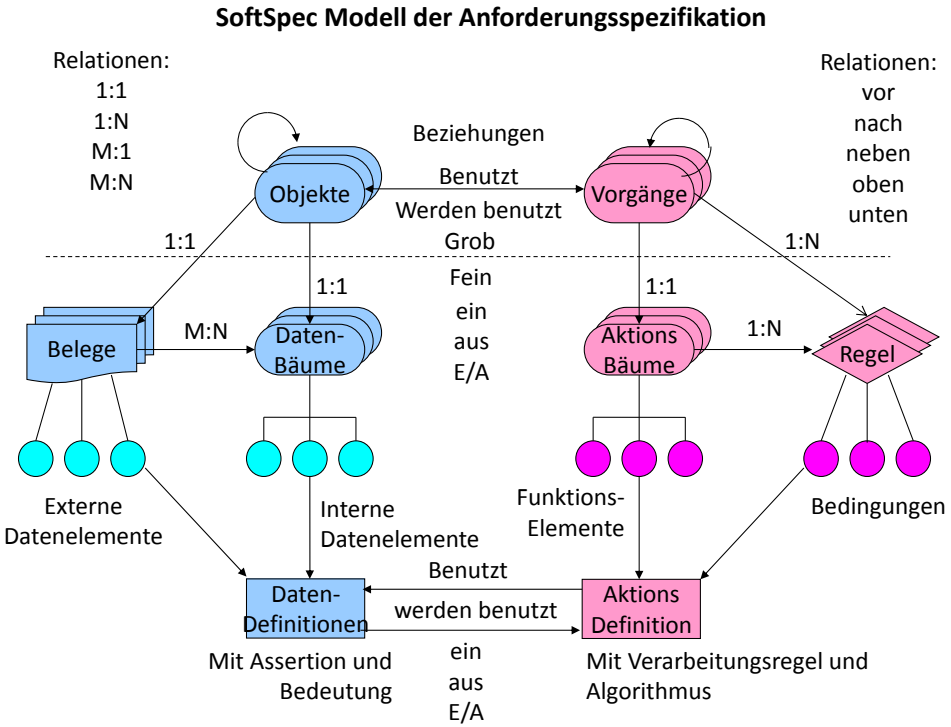
In dem groben Funktionsquadrat gab es die Vorgänge. Heute würde man sie als Anwendungsfälle bezeichnen. Vorgänge sind Hauptfunktionen bzw. Schritte in einem Geschäftsprozess. Sie beinhalten mehrere Elementarfunktionen. Für die Vorgänge gab es zwei alternative Darstellungsweisen – als Funktionsbaum und als Entschei-

dungstabelle. Die Entscheidungstabelle diente dazu die Funktionen mit den Bedingungen zu verbinden.

In dem feinen Funktionsquadrat gab es die elementaren Funktionen bzw. Operationen. Jede Elementarfunktion konnte bedingt oder unbedingt sein. Wenn sie bedingt war, musste sie eine Regel haben. Die elementaren Funktionen wurden in Funktionsbäumen, die Regel in Entscheidungstabellen dargestellt.

Neben den vertikalen Beziehungen zwischen grob und fein, bzw. zwischen über- und untergeordneten Funktionen und Daten gab es horizontale Beziehungen zwischen Vorgängen und Objekten sowie zwischen Funktionen und Daten. Vorgänge erzeugen, verändern und löschen Objekte, Funktionen verarbeiten und befragen Datenelemente. Diese Querbeziehungen wurden in groben und feinen Datenflussdiagrammen abgebildet.

Das SoftSpec Modell fußte theoretisch auf dem Entity/Relation Modell von Chen und eignete sich praktisch um fast jede Applikation darzustellen [Chen76]. Neben dem Prüfstand und dem Nachfolgeprodukt SofTest war es das Beste, was ich je ausgedacht habe. Das Tool wurde von Erika Nyary und ihrem Team auf hervorragender Weise implementiert. Ein vergleichbares System zur Erfassung, Speicherung und Prüfung betrieblicher Anforderungen gab es zu dieser Zeit nur im fernen Amerika und dort nur im Verteidigungsbereich. Jeder Datenname und Funktionsname, den der Anwender benutzte, wurde gegen das bestehende Namensverzeichnis geprüft ob er bereits existierte. Wenn nicht, musste der Anwender ihn zentral für das ganze System definieren. So wurde die Konsistenz der Namensvergebung gewährleistet und die Verfolgbarkeit gleich eingebaut. Die Konsistenz und Vollständigkeit der Spezifikation wurde in Echtzeit fortwährend kontrolliert. Das Anforderungsmodell durfte gar nicht erst inkonsistent werden [Nyar83].



Die schwächen von SoftSpec ergaben sich aus den Einschränkungen der damaligen Mainframe Rechner. Es gab keine graphischen Benutzeroberflächen. Die Arbeit mit TSO/SPF Masken war umständlich. Das SzKI Team nahm heraus was die Technologie hergab. SoftSpec war eben ein Kind seiner Zeit, zugeschnitten auf die Entwicklung auf dem Mainframe. Als die PC-Revolution aufkam, wurde sie abgehängt. Ende der 80er Jahren haben wir das Modell auf dem PC mit Windows implementiert aber es war zu spät. Es fehlte ein adäquates Datenbanksystem auf dem PC um die Spezifikation-Repository zu verwalten und wir hatten nicht mehr die Ressourcen eins selber zu bauen.

4.3.2 SoftCon für den Entwurf der Systemarchitektur

Aufgrund meiner Erfahrung mit den verschiedenen Betriebssystemen und Datenbanksystemen bei den deutschen Hochschulen wusste ich, es würde nicht leicht sein das fachliche Anwendungsmodell in ein DV-technisches Modell umzusetzen. Es gab auch zu der Zeit Anfang der 80er Jahren zu viele unterschiedliche Zielumgebungen. Allein in der IBM-Welt gab es drei Betriebssysteme, drei Datenbanksysteme und

zwei Telekommunikationssysteme. Hinzu kamen die DB/TP Systeme von Drittanbieter, wie die Software AG, ADR und Collinet. Diese Systeme widersprachen sich oft in ihrer Grundtheorie. In der IMS Umgebung der IBM waren die Anwenderprogramme Hauptprogramme und die IMS-Kommunikationsprogramme die Unterprogramme. In der CICS-Umgebung der IBM war es gerade umgekehrt. Die Kommunikationsprogramme steuern den Dialog mit dem Endanwender und riefen die Anwenderprogramme auf. Das Siemens BS-2000 Betriebssystem war ein Time-Sharing System und regelte den Dialog mit den Endbenutzern auf einer ganz anderen Art und Weise. Das IMS-Datenbanksystem war hierarchisch und griff auf die Datenobjekte bzw. Segmente nur über die übergeordnete Segmente-Parents. Andere Datenbanksysteme wie ADR-Online und IDMS waren netzartige Datenbanksysteme nach dem CODASYL Form und griffen auf die Daten über vordefinierten Zugriffspfade mit Owners und Members. Dann gab es ADABAS mit seiner tubulären Datenspeicherung, ein Vorgriff auf die relationalen Datenbanksysteme, die erst später folgten.

Es wäre unmöglich gewesen sämtliche Betriebsumgebungen auf dem Markt abzudecken. Es blieb nur übrig sich auf einer oder vielleicht zwei zu beschränken. Ich konnte jedoch nicht wissen, welche das sein sollte. Es hing davon ab, welche Anwender SoftSpec einsetzen würden. Es wäre für uns günstiger gewesen, wenn wir auf dem Siemens Rechner hätten bleiben können, aber es kam anders. Unsere ersten Kunden auf dem deutschen Markt hatten alle IBM Rechner entweder mit CICS oder mit IMS. So wurden wir gezwungen in jene Richtung zu gehen. Es blieb uns nichts Anderes übrig, wenn wir auf dem Markt Fuß fassen wollten.

Dem Tool SoftCon fällte die Aufgabe zu, das fachliche Modell in eine DV-technische Architektur umzusetzen. Dafür waren vier nebenläufige Transformationsschienen vorgesehen:

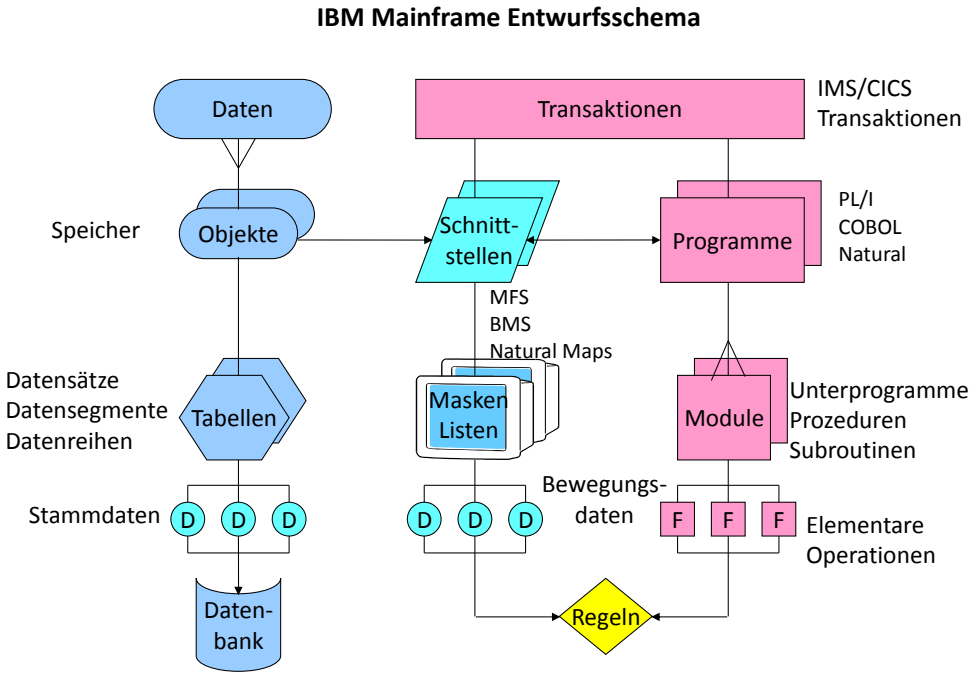
- die Datenbankgenerierung,
- die Maskengenerierung,
- die Programmgenerierung und
- die Testgenerierung.

Das Ergebnis der Generierung sollte ein strukturierter Systementwurf im Sinne von Yourdan und Constantine [Your79] und ausgedrückt in einer Entwurfssprache sein: PDL – Program Design Language – sowie die in meinem Vorbild, dem BMD Entwicklungssystem [Dave77]. Daraus soll dann der endgültige Code generiert werden. Ich wollte es offenhalten, welcher Datenbanktyp, welcher Maskentyp und welche Programmiersprache letztendlich dabei herauskommen. Die Konzeption dieser Sprache hat mich viele Jahre beschäftigt. Sie musste einerseits alles aus dem SoftSpec

Fachmodell abbilden und anderseits alle technische Implementierungspfade abdecken – der Pfad zu IMS sowie der Pfad zu DB2 und der Pfad zu ADABAS, der Pfad zu IMS-DC sowie zu CICS und Natural Benutzertransaktionen, und der Pfad zur jeweiligem Programmtyp, ob COBOL, PL/I oder Natural. Natural hatte sich inzwischen als 4GL-Programmiersprache immer mehr ausgebreitet, vor allem in Banken und Behörden [Mart85]. Ich habe mit einem Sprachkern angefangen und diesen im Laufe der Jahre immer weiter ausgebaut. Der SoftCon Entwicklung begann mit einem neuen Team unter der Leitung von Laszlo Fehervari im Jahre 1982, zwei Jahre nach dem Start der SoftSpec Entwicklung. Es hat so lange gedauert, um die Entwurfssprache endgültig zu spezifizieren. Inzwischen war SoftSpec bei drei Kunden in Deutschland in Einsatz und der Druck nahm zu, aus den bereits erarbeiteten Fachmodellen lauffähige Systeme zu generieren. Die Anwender haben es erwartet.

Eigentlich hätte man die von SoftSpec erzeugte Anforderungsdokumentation als Vorgabe für eine manuelle Implementation sowie als Grundlage für den Systemtest verwenden können. Es wäre nicht unbedingt notwendig gewesen, daraus Programme automatisch zu generieren. Die ersten Anwender bestanden jedoch darauf. Sie meinten, sie hätten schon zu viel in die fachliche Lösung investiert um sie einfach dabei zu lassen. Der eingeschlagene Weg sollte weiter bis zum lauffähigen System führen. Ich wollte auch die Lücke bis zum Code irgendwie schließen, anderseits war die Lücke viel zu breit um sie mit einem Sprung zu überwinden. Deshalb die SoftCon Entwurfssprache zwischen der Anforderungssprache und der Programmiersprache. Heute hat man dafür UML aber zu dem Zeitpunkt gab es noch keine allgemein anerkannte Entwurfssprache.

Diese Entwurfssprache bestand aus mehreren Untersprachen – eine Datenentwurfssprache zur Beschreibung der physikalischen Datenbankstrukturen mit Schlüssel, Suchbegriffe, Sichten und Relationen aber unabhängig von dem darunter liegenden Datenbanksystem, so etwas wie ein universales Datenbankschema. Sie sollte in jede gängige Datenbanksprache übersetzbar sein. Eine weitere Entwurfssprache – die Oberflächenentwurfssprache – diente zur Beschreibung der Benutzerschnittstellen. Dazu zählten die Bildschirmmasken und die Listenausdrucke. Diese Sprache gab an, an welchen Stellen die Anzeigefelder und Empfangsfelder zu platzieren sind, wie lang sie sind und von welchem Typ: daraus sollte es möglich sein CICS-, IMSDC- und Natural Masken zu generieren sowie auch die Listenformatbeschreibungen in COBOL, PL/I und Natural.



Die Entwurfssprache für die Programmlogik war die komplexeste, denn daraus sollten compilierbare, ausführbare COBOL, PL/I oder Natural Programme generiert werden. Die Meta-Anweisungen der Programmentwurfssprache mussten so formuliert sein, dass daraus Anweisungen aller Zielsprachen ableitbar sein sollten. Es gab schon zu diesem Zeitpunkt Metasprachen mit denen es möglich war, andere Sprachen zu beschreiben und ich habe mich damit befasst, doch die Umsetzung im Detail war schwierig. Ich hätte mich auf eine Sprache konzentrieren sollen, wie z.B. COBOL. So wäre es für mich einfacher gewesen. Dann hätte ich aber Kunden mit anderen Sprachen nicht gewinnen können. Die Vielfalt der IT-Welt war schon immer das Haupthindernis auf dem Wege zu einer automatisierten Softwareentwicklung. Selbst IBM ist mit ihrem AD-Cycle Projekt über dieses Hindernis gestolpert [IBM89]. PL/I und Assembler waren damals in der IBM Mainframe Welt sehr verbreitet und Natural war im Kommen. Wenn ich diese Anwender als Kunden gewinnen wollte, mussten meine Werkzeuge ihre Sprache sprechen. Demzufolge sah ich mich gezwungen, eine Metasprache einzuführen, die als Drehscheibe zwischen der fahlich-orientierten Spezifikationssprache und der jeweiligen Programmiersprache dienen könnte.

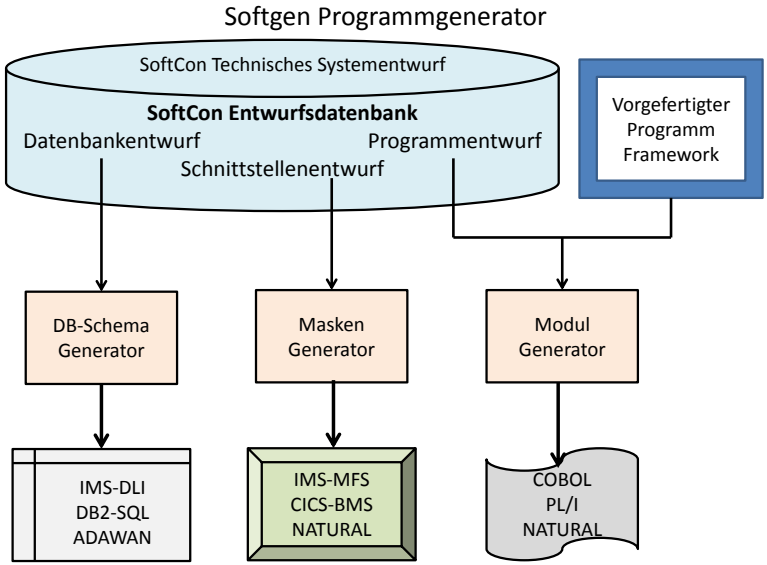
In SoftCon war es vorgesehen, die Entwurfssprache online zu editieren, d.h. der aus SoftSpec generierten Pseudocode Texte konnten korrigiert und ergänzt werden, ehe sie in die Ziel-Sourcetexte umgesetzt wurden. Wenn er wollte, konnte der Anwender die Detaillogik gänzlich in der Entwurfssprache erfassen, was später einige Anwender wie Krupp, Thyssen und die Bundesbahn auch gemacht haben. Damit wurde jedoch die fachliche Spezifikation von der technischen Implementierung abgehängt. Sie diente nur noch als Absichtserklärung der Fachabteilung. Es war nicht mehr möglich die Funktionen im Code darauf zurückzuverfolgen.

Als positiv an SoftCon zu bewerten war die umfangreiche Qualitätssicherung. Die einzelnen Entwürfe der Datenbanken, der Benutzeroberflächen, der Berichte und der Programmlogik wurden automatisch auf Vollständigkeit, Konsistenz und Konformität geprüft und Abweichungen gemeldet. Darüber hinaus wurden die Entwurfsentitäten gegen die Spezifikationsentitäten in der SoftSpec Datenbank quer geprüft. Diese Prüfung sollte sichern, dass die technische Implementierung mit den fachlichen Vorgaben übereinstimmte. Nicht implementierte Anforderungen wurden auf dieser Weise aufgedeckt und angezeigt. Diese umfassende Qualitätssicherung in SoftCon war gedacht um zu verhindern dass die Fehler und Versäumnisse der Analyse in den Code hineingetragen werden. Eigentlich waren wir mit SoftSpec und SoftCon unserer Zeit weit voraus, zu weit. Ich hatte mich nach den Visionen des amerikanischen Marktes gerichtet [Mart85a].

4.3.3 SoftGen für die Generierung der Programme

Das Tool SoftGen war das letzte Glied in der Entwicklungskette. Damit wurde auf die Entwurfsdatenbank zugegriffen um Source Code für den Zielrechner zu produzieren. Eigentlich bestand SoftGen aus vielen Einzelwerkzeugen, eins für jede Zielsprache. Es gab ein Tool um IMS, DB2 und ADABAS Datenbankschemen interaktiven aus dem Datenentwurf zu generieren, ein Tool um IMS, CICS und Natural Masken zu generieren, ein Tool um COBOL, PL/I und Natural Source Module zu generieren und sogar ein Tool um Testskripte aus den Datenbeschreibungen zu generieren. Im Gegensatz zu SoftSpec und SoftCon – die große interaktive, zusammenhängende Online-Programme waren – war SoftGen eine Sammlung unabhängiger Batchläufe. Unter TSO-SPF bekam der Benutzer ein großes Menü der Generierungsmöglichkeiten präsentiert und konnte auswählen, was er generieren wollte. Demnach wurde ein Batch-Job gestartet, der auf die betroffenen Tabellen in der SoftCon Datenbank zugriffen hat und den gewünschten Source-Code Dateien erzeugte. Das Ganze war total modular aufgebaut. Man konnte die Einzelmodule austauschen und statt COBOL, PL/I Sourcen oder statt IMS, CICS Masken generieren. Später, für Thyssenstahl haben wir

sogar Tandem-COBOL und Tandem-Masken sowie ADABAS Datenbanken generiert. Für BMW mussten wir IMS Masken und IMS-DBDs sowie PL/I Code generieren. Für Bertelsmann haben wir CICS-Masken, ADABAS Datenanken und COBOL Programme generiert. Für die Bundesbahn müssten wir CICS-Masken, DB2 Datenbanktabellen und COBOL Programme generieren. Dies zeigt, wie unterschiedlich die Anforderungen waren allein auf dem IBM-Mainframe. Diese Diversität war die Ursache vieler Probleme in der SoftOrg Entwicklung. Natürlich gab es auch andere Probleme, wie die Entfernung der ungarischen Entwickler von der deutschen IT-Welt und der beschränkte Zugang zu IBM Mainframe Rechnern. Aber, ohne die Vielfalt der Zielumgebungen wäre alles viel einfacher gewesen.



Ursprünglich war es meine Absicht, SoftGen in der SzKI zu entwickeln zusammen mit den anderen Entwicklungswerkzeugen. Ich meinte, die Programmgenerierung musste im Dialogbetrieb stattfinden. Als ich später die Aufgabenstellung besser verstand, habe ich gesehen, dass die vielen unabhängigen Generatoren am besten als Batch-Jobs zu realisieren waren. Den ersten Prototyp für die Generierung von CICS/COBOL Programme habe ich selber in PL/I programmiert. Das war mein einziger Programmierbeitrag zu den SoftOrg Entwicklung. Ich bräuchte dringend einen Generator für die Deutsche Bank und wollte nicht warten, bis die SzKI-Leute dazu kommen. Außerdem hatten die Institute begonnen Vorauszahlungen für neue Werkzeuge zu verlangen. Das war im Jahre 1984. Ein Jahr später habe ich das SzAMOK

Team von Herrn Jandrasics beauftragt, das Produkt neben SoftDoc - das Programmanalysewerkzeug - zu entwickeln. Die allumfassende SoftGen Version kam erst 1986 zum Einsatz, sechs Jahre nach dem Beginn der SoftOrg Entwicklung. Wie auch die anderen SzAMOK Tools, wurde es in PL/I implementiert, so wie ich mit dem Prototyp begonnen hatte. Inzwischen gab es etliche Programmgeneratoren auf dem Markt die uns als Vorbild dienen konnten. Wir hielten uns mehr oder weniger an der damaligen Generierungstechnik [Rice81].

SoftGen wurde so konstruiert, dass die einzelnen Generatoren unabhängig voneinander blieben. Somit konnten sie leicht ausgetauscht werden. Dies ist auch geschehen als wir Code für völlig unterschiedliche Sprachen in unterschiedlichen Umgebungen erzeugen mussten. Das Konzept der Verteilung hat sich bewährt. Es hat eine Weile gedauert, aber am Ende ist es gelungen COBOL-CICS, COBOL-IMS, Tandem-COBOL, PL/I-CICS, PL/I-IMS und Natural/Adabas Programme vollständig zu generieren. Dies war eine unserer größten Leistungen. Leider kam es zu spät für einige Kunden, die inzwischen schon abgesprungen waren.

4.4 Die Testwerkzeuge – vom Code zur Abnahme

Auf der Rückseite des Lebenszyklus waren – wie auf der Vorderseite – drei Werkzeuge geplant

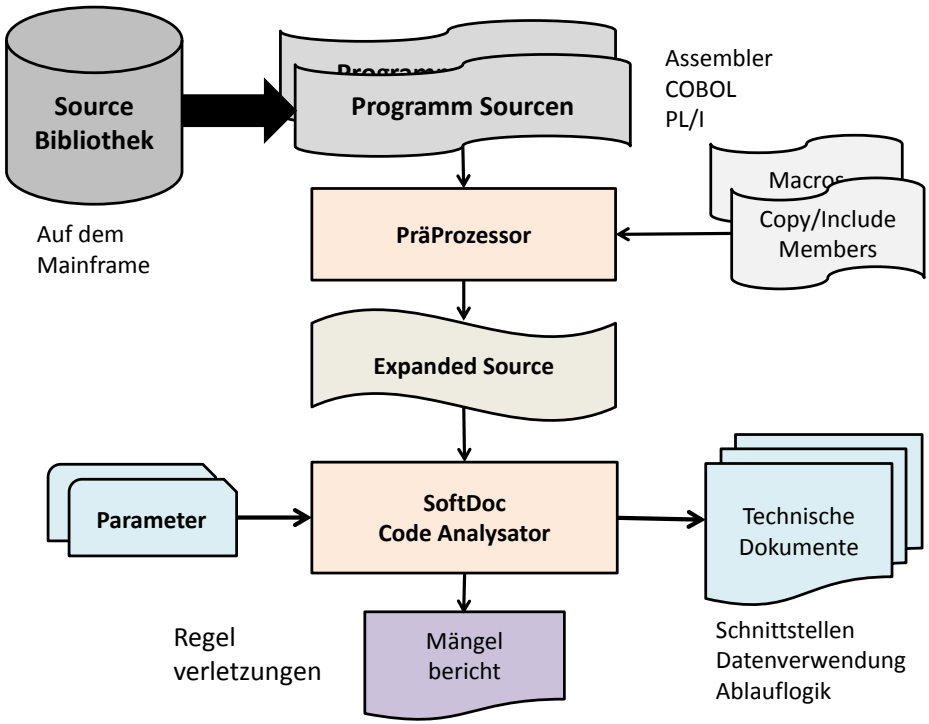
- SoftDoc für die statische Analyse des Codes
- SoftGen für die dynamische Analyse und
- SoftInt für die Integration der fertigen Komponente

Einer meiner Grundsätze war, dass es möglich sein sollte jedes Werkzeug unabhängig von den anderen zu benutzen. Es gab zwar einen Datenaustausch zwischen den Tools aber er war keine Vorbedingung für deren Nutzung. Falls ein Tool die gleichen Funktionen innehatte wie ein anderes Werkzeug z.B. das Parsing des Zielcodes, wurde die Funktion in allen beiden Tools eingebaut. D.h. sie wurde dupliziert. Das war Absicht, damit kein Tool von anderen abhängig war. Dieses Prinzip hat sich mehrfach bewährt. So hatte jedes Team die volle Verantwortung für sein Produkt. Es gab kein Common Code.

4.4.1 SoftDoc für die Prüfung und Nachdokumentation des Codes

In dem SzAMOK Institut gab es also neben dem Team für SoftGen drei weitere Teams, eins für jedes Testwerkzeug. Das Software Team unter der Leitung von G. Jandrasics übernahm den statischen Analysator von Prüfstand und reimplementierte

ihn in PL/I. Sie konnten damit Source Code in den drei IBM Sprachen – Assembler, COBOL und PL/I. – analysieren.

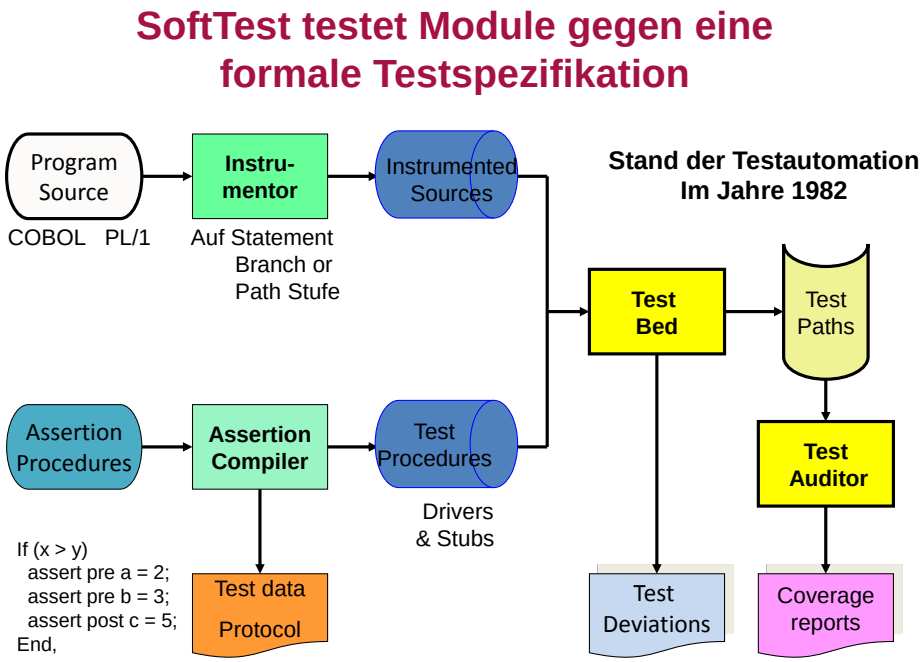


Aus jeder der drei Sprachen wurde eine gemeinsame Zwischensprache geschaffen. Die Zwischensprache wurde in mehrere Tabellen aufgeteilt, eine für die Darstellung der Programmarchitektur, eine für die Spezifikation der Schnittstellen, eine für die Abbildung der Datenstrukturen, eine für die Beschreibung der Datenflüsse und eine für die Wiedergabe der Entscheidungslogik in Form von Entscheidungsbäumen. Bei der Codeanalyse wurde der Code gleichzeitig gegen eine Reihe Codierregel geprüft und die Regelverletzungen protokolliert. Außerdem wurden einige Eigenschaften – wie Ablaufkomplexität, Datenkomplexität und Datenflusskomplexität sowie Modularität, Wiederverwendbarkeit und Portabilität – gemessen Ich habe mich dabei an die Werkzeuge von Tom McCabe und Mike Hennel angelehnt. Unser Ansatz zur Codeanalyse haben wir zunächst auf der COMPSAC Konferenz in Chicago 1982 vorgestellt. Später habe ich zusammen mit Andras Merey, dem damaligen SoftDoc Projektleiter über die statische Analyse mit SoftDoc für die IEEE Transactions on S.E. verfasst. Er wurde angenommen und im Frühjahr 1985 veröffentlicht [SnMe85].

Was die Anzahl der Anwender anbetrifft, war SoftDoc unser erfolgreichstes Produkt. Wir kamen auf mehr als 30 Kunden, darunter viele PL/I Anwender wie BMW, die Allianz Versicherung, die Volksfürsorge, die TÜV, die CA in Wien und die Schering in Berlin. Sie benutzten das Tool vor allem um ihren Code zu prüfen und zu messen. Die Dokumentation haben sie mehr als Beigabe angesehen. Sie entsprach nicht gerade ihrer Anforderungen.

4.4.2 SoftTest für den Test der Programme

Mit SoftTest hatten wir nicht so viel Erfolg, obwohl es das anspruchsvollere Produkt war. Das war auch das Problem – es war zu anspruchsvoll. SoftDoc musste nur angestoßen werden, es lief dann voll automatisch ab. In Falle von SoftTest mussten die Anwender ein Testskript in Prädikatenkalkül erster Ordnung schreiben um die Eingaben zu generieren und die Ergebnisse testfallweise zu validieren [Majo83]. Bei Kunden mit Akademikern in der Qualitätssicherung wie bei der Allianz, der TÜV, Höchst und der BASF kam es gut an. Sie waren fasziniert von der Möglichkeit, den Datenbereich der einzelnen Module zu manipulieren, die Ergebnisse zu kontrollieren und die Testüberdeckung zu messen. Es gab aber zu diesem Zeitpunkt nur wenige Akademiker in der Datenverarbeitung. Für die gewöhnlichen Entwickler war das Ganze einfach zu kompliziert. Sie mussten im Voraus festlegen, welche Ausgaben auf welche Eingaben folgen, d.h die Beziehungen zwischen den Variablen exakt definieren. Damit waren die Durchschnittprogrammierer überfordert. Folge dessen kamen wir nie über 10 Anwender hinaus, obwohl SofTest einen technologischen Durchbruch für die damalige Zeit war. Es gab weltweit kein anderes Testwerkzeug, das so viele Funktionen hatte. Die Skriptsprache mit pre- und post-Assertionen diente zugleich als Modulspezifikation für die Entwicklung. Der Code konnte gegen diese Spezifikation getestet werden. Ich hatte aus dem I.T.S. Projekt gelernt nie ohne Orakel zu testen. Das Orakel war bei SofTest die Testskripte. Leider waren nur wenige Anwender soweit diesen spezifikationsgetriebenen Testansatz ausreichend zu würdigen.



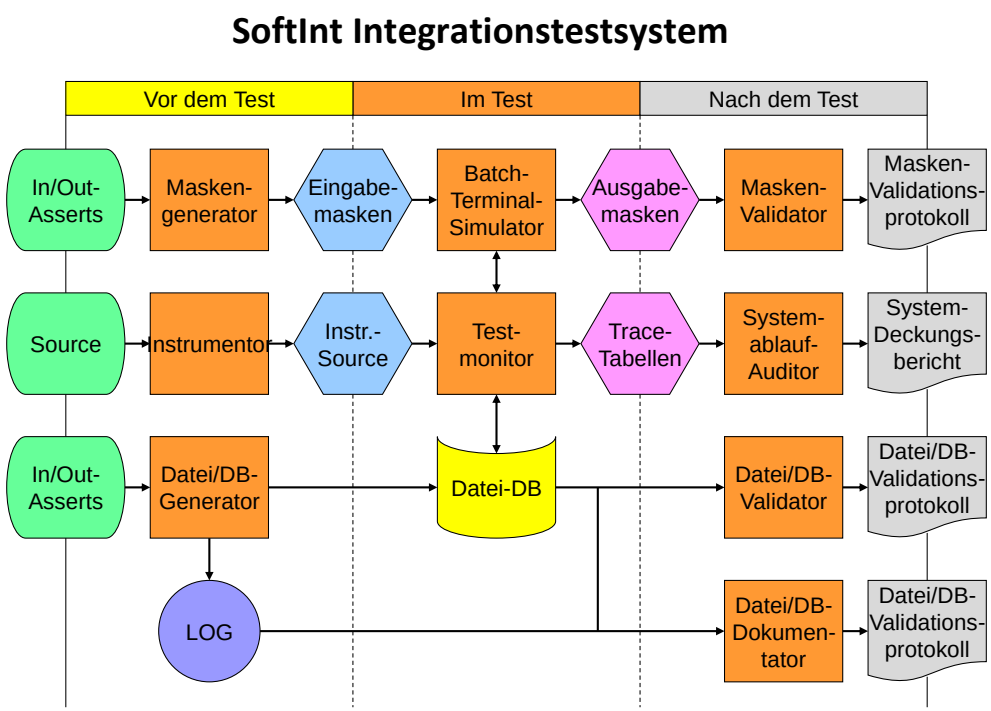
4.4.3 SoftInt für den Test des Gesamtsystems

Bei der Entwicklung von SoftInt bin ich vorsichtiger geworden. Ich bin nicht so akademisch vorgegangen. Der Integrationstest sollte möglichst einfach sein. SoftInt bestand im Prinzip aus drei gekoppelten Werkzeugen entsprechend den drei Schritten im Testprozess:

- erstens sollten Testdaten generiert werden – Datenbanken, Dateien und Nachrichten. Dafür brauchten wir einen Generator
- zweitens sollten einzelne Programmkomponente angestoßen werden und ihre Ausführung überwacht werden, dafür brauchten wir einen Testmonitor
- drittens sollten die Ergebnisse des Tests – Datenbanken, Dateien, Listen und Nachrichten validiert werden. Dafür brauchten wir einen Validator.

SoftInt war eigentliche eine Nachahmung des Integrationssystems das wir früher für MBB entwickelt hatten, nur das SoftInt mehr allgemeiner Natur war [Erdo85]. Das Problem dabei war wiederum die Vielzahl der Datenarten. Bei MBB gab es nur

VSAM und DL/I. Das Standardprodukt müsste vielmehr Varianten abdecke. Es gab sequentielle Dateien, ISAM-Dateien, VSAM-Dateien, IMS-Datenbanken, DL/I-Datenbanken, IDMS, SESAM, ADABAS- und DB2-Datenbanken. Für jede dieser Datenbankarten mussten wir stellvertretende Sätze mit unterschiedlichen Wertkombinationen erzeugen – Äquivalenzklassen, Grenzwerte und Wertebereiche. Das Testdatenskript wurde aus dem Datenbankschema, bzw. aus der Dateibeschreibung, generiert und es wurden repräsentative Werte zugewiesen, aber der Tester musste die echten Testwerte am Ende zuweisen. Der Generator hat sie mutiert und vervielfältigt.



Das gleiche Problem stellte sich auf der Validierungsseite. Er wurden unterschiedliche Datenobjektarten verglichen mit Wertebereichen, die sie haben sollten. Passte ein Ist-Wert der tatsächlichen Ausgabe nicht zu dem entsprechenden Sollwert in der Ausgabespezifikation, galt der Datensatz als ungültig. Das Tool musste Millionen von Sätzen lesen und vergleichen und das in einer vertretbaren Zeit. Demnach war die Performanz ein strenges Kriterium.

Was der Testmonitor anbetrifft, war es der Zweck der Testüberwachung festzustellen, wie die Programme unter Test sich verhalten. Dazu mussten die Programm-Sourcen vor dem Test instrumentiert und nach dem Test die Ergebnisse der Instrumentierung auswertet werden. Die Ablaufpfade wurden in Trace-Files festgehalten. Es ging beim Integrationstest darum, jede IO-Operation, jeder Datenbankzugriff, jede Listenausgabe sowie jede Bildschirm Ein- und Ausgabe zu registrieren und dies zu protokollieren, damit der Tester sie nachträglich kontrollieren konnte. So wurde auch die Testüberdeckung gemessen. Es kam darauf an, jeden Modulaufruf, jeden Datenbankzugriff und jede Eingabe/Ausgabe Operation mindestens einmal auszuführen.

SoftInt war auch das erste Testwerkzeug um Datenüberdeckung zu messen. Ich habe zusammen mit Frau Erdös ein PL/I Programm geschrieben um die Daten vor und nach dem Test miteinander zu vergleichen um festzustellen welche Daten sich verändert hatten. Es wurden Zugänge, Löschungen und Veränderungen gezählt. Am Ende kam heraus wie die Daten verwendet werden, als Eingabe, als Ausgabe oder als Beides. Der Bericht darüber war der einzige Beitrag von mir der je von einem ACM Testworkshop angenommen wurde [Sned86]. Erst später im Jahre 1990 sollte in San Francisco vor 400 Teilnehmer der StarWest Konferenz darüber vortragen.

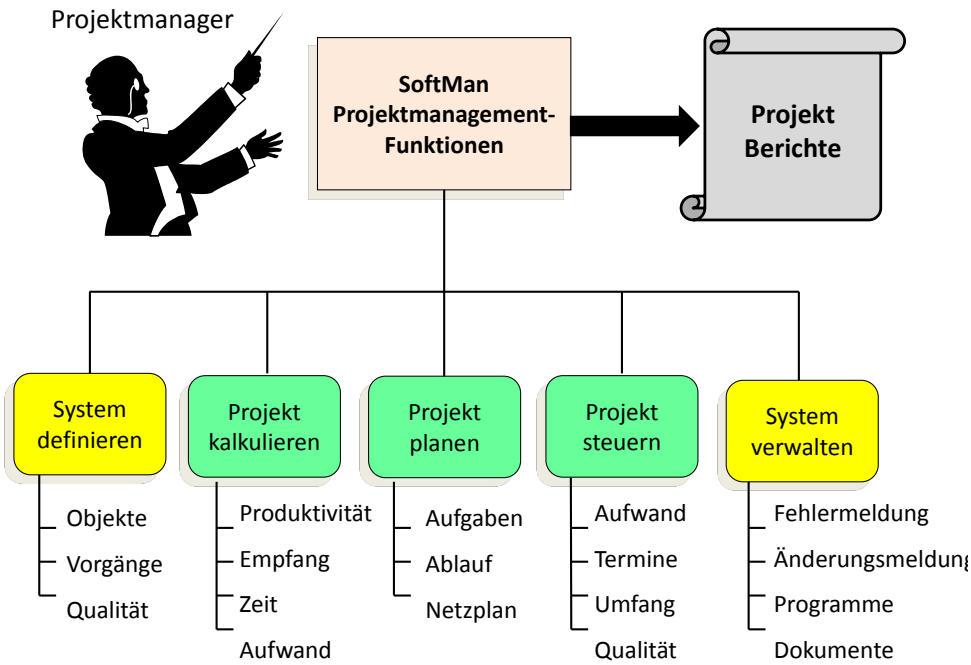
Zur Entwicklung dieses Werkzeuges brauchten die Entwickler gute Datenbankkenntnisse. Ich habe dazu ein neues Team von drei Entwickler in der SZAMOK aufgebaut unter der Führung von Frau Erdös von dem SofTest Team. Als Frau Majoros, Leiterin des SofTest Teams in Deutschland um politisches Asyl bat, musste Frau Erdös auch dieses Team übernehmen. Wir hatten schon Dr. Merey vom SoftDoc Team auf dieser Weise an BMW in München verloren. Natürlich hat die SoftOrg Entwicklung unter diesem Personalverlust gelitten. Es wurde immer schwieriger Schlüsselpersonen zu ersetzen. Das hat mich veranlasst immer weniger zu delegieren und immer mehr selbst zu machen. Von dem SzAMOK Institut sind insgesamt fünf Entwickler in Deutschland geblieben. von dem SZKI Institut aber keine. Die SzKI Leute waren politisch zuverlässiger.

4.5 Das übergeordnete Lebenszyklus Management Werkzeug

Unter anderem aus dieser Überlegung heraus habe ich entschieden, das allerletzte Werkzeug – SoftMan von der SZKI entwickeln zu lassen. Mit diesem Projektmanagement Tool wollte ich die SoftOrg Projekte definieren, schätzen, planen, überwachen und verwalten. Dafür waren mehrere Benutzeroberflächen und eine große Projektdatenbank mit vielen Tabellen erforderlich – eine Tabelle für jeden Projekttyp. Zu

den Objekttypen zählten die Produktbestandteile, die Projektaktivitäten und –aufgaben, die Schätzgrößen und die Wartungsaufträge – Fehlermeldungen und Änderungsanträge [Boeh96].

Die Produktbestandteile waren auf der Funktionsseite die Anwendungen, die Prozesse und die Vorgänge. Auf der Datenseite waren sie die Datenbereiche, die Datenbanken und die einzelnen Datenobjektarten – Sätze, Tabellen und Segmente. Es gab noch eine Kommunikationsseite mit Kommunikationsbereich, Schnittstellen und Nachrichtenarten. Die Aufgabe der Produktdefinition war alle diese Entitätentypen und ihre Beziehungen zueinander zu erfassen und zu modellieren. Es sollte möglich sein bis auf Vorgangsebene, Objektebene und Nachrichtenebene das geplante System zu modellieren und es aufgrund des Produktumfangs einigermaßen präzise schätzen zu können.



Die Schätzgrößen waren die Risiken: die Einflussfaktoren, die Produktivitäten und die Schätzregel. Der Benutzer konnte jedes Risiko identifizieren und gewichten, sowie auch jeden Einflußfaktor. Für die Produktivitätsmessung gab es eine Produktivi-

tätstabelle für jeden Größenmaß – für Code-Zeilen, Anweisungen, Function-Points, Data-Points und Anwendungsfälle in der das Verhältnis von dem Größenmaß zu dem erforderlichen Aufwand in Personentagen festgelegt war. Dieses Verhältnis wurde entweder aus der Literatur entnommen oder aus dem eigenen Erfahrungsschatz. Für jede Schätzmethode gab es einen Schätzalgorithmus um den Aufwand und die Zeitdauer zu errechnen. Der Anwender konnte die geeignete Schätzmethode aus einem Menu auswählen.

Die Projektelemente waren die Projekte, die Phasen und die Ausgaben sowie die Produkte, die Teilprodukte und die Ergebnisse. Hinzu kamen die Ressourcen – die Hardware, die Software und die Menschen. Zwischen diesen Objekten gab es jede Menge Beziehungen, z.B. ein Projekt liefert ein Produkt, eine Aufgabe erzeugt ein Ergebnis, eine Aufgabe benutzt eine Ressource. Die Objekte mit ihren Attributen waren in der einen Tabelle, die Beziehungen mit Quellobjekt und Zielobjekt in einer anderen. Aus diesen beiden Tabellen war es möglich Projektablaufdiagramme, Netzpläne und Ganttdiagramme zu generieren.

Die Wartungsaufträge waren die Fehlermeldungen, die Änderungsanträge und die Erweiterungswünsche. Diese Objekte enthielten feste Attribute wie Fehlerart, Schwäre, Priorität, Terminwunsch und Nutzen sowie einen freien Textteil. Der Benutzer konnte sie verbunden mit bestimmten Vorgängen, Datenobjekten, Komponenten und Schnittstellen. Diese wurde unterstützt von tabularen Masken in TSO/ISPF auf dem Mainframe Rechner.

Es kamen zahlreiche Berichte hinzu, die der Benutzer beliebig ausdrucken konnte, z.B. Planungsberichte und Statusberichte zu den Terminen, den Kosten und dem Projektfortschritt. Geplant waren Schnittstellen zu den anderen Werkzeugen, um sowohl den Fertigstellungsgrad als auch die Qualität des Endproduktes zu verfolgen. Von SoftSpec wurde übergeben, welche Spezifikationsobjekte fertig waren mit welcher Qualität. Von SoftCon kam die Information über den Stand der Entwurfsobjekte – quantitativ und qualitativ. Von SoftDoc kamen Nachrichten über die statische Qualität der Programme und von SofTest kamen Nachrichten über deren dynamischen Qualität. SoftMan hat alle diese Daten aggregiert und in einzelnen Projektberichten zusammengefasst. Der Projektleiter bekam auf dieser Weise einen umfassenden Überblick über den Stand seines Projektes.

Leider setzte ich mit der Entwicklung von SoftMan zu spät an. Für die ersten Kunden – Bertelsmann, BMW, Deutsche Bank und Württ-Feuer –waren die Projekte schon im Gange. Es machte keinen Sinn sie nochmals zu planen. Der erste Einsatz

von SoftMan erfolge bei Thyssen-Stahl. Dort wurde es verwendet, um das Schienen-Kontrollsystem zu planen und zu verfolgen. Die Prozesse wurden modelliert und den Aufwand für deren Realisierung geschätzt. Der dortige Projektleiter – Dr. A. – war begeistert. Ein solches Projektmanagement Tool mit so vielen Informationen hat es bei Thyssen bisher nicht gegeben. Es folgten Einsätze bei der Bundesbahn und dem Bremer Lagerhaus. Die Reaktion der Benutzer war überall positiv. [Sned83].

4.6 Die unvollendete Vision

In einem Buch von Mitarbeitern der Gesellschaft für Mathematik und Datenverarbeitung – GMD aus dem Jahr 1985 wurde SoftOrg angepriesen als die fortschrittlichste Software Produktionsumgebung auf dem deutschen DV-Markt [Haus85]. Sie würde die meisten Entwicklungsaktivitäten abdecken. Leider wurde die SoftOrg Toolentwicklung nie vollendet. Der Systemwechsel in Osteuropa kam dazwischen. Die Frage stellt sich allerdings ob ein so umfassendes Werkzeugsystem in einer so dynamischen Umgebung wie Softwareentwicklung überhaupt je vollendet werden kann. Es ist alles im Fluss – die technische Umgebung, die Anforderungen der Anwender, die Entwicklungsmittel. Entwurfsmethoden und Programmiersprachen kommen und gehen. Mal heißt es die Funktion steht in der Mitte des Universums – was geschieht. Später heißt es das Objekt steht im Mittelpunkt – das Was. Man stieg von Maschinensprachen auf prozeduralen Sprachen, auf 4GL Sprachen, auf objektorientierten Sprachen und wieder auf prozessorientierten Sprachen. Am Anfang gab es nur Dateien, dann hierarchische Datenbanken, dann netzartige Datenbanken, dann relationale Datenbanken, dann objekt-relationalen Datenbanken, Die Evolution der IT-Entwicklung schreitet stets voran im Sinne von Lenin – zwei Schritte vorwärts, ein Schritt zurück. Manchmal hat man den Eindruck, es dreht sich im Kreis. Beim Versuch diese Welt abzubilden und zu automatisieren läuft man ständig hierher. Kaum ist der eine technologische Stand einigermaßen unterstützt dann kommt der Nächste. Der Entwickler von Tools wird immer von dem nächsten Technologieschub abgehängt.

Im Jahre 1988 stand ich, was die SoftOrg Entwicklung anbetraf, kurz vor der Vollendung meiner Vision. Alle Einzelwerkzeuge waren weitgehend abgeschlossen und miteinander integriert. Ich hatte etwas erreicht, was keiner vor mir in Europa erreicht hatte – eine vollintegrierte Software Entwicklungsumgebung mit Test. Ich hatte auch alle Werkzeuge sorgfältig dokumentiert und darüber berichtet. Sogar die Universität California in Berkeley hatte von uns gehört und um eine Installation gebeten. Herr Jandrasics und Frau Nyary sind mit ihren Teams hingeflogen und haben dort die Werkzeuge an dem Universitätsserver installiert. Es schien, als ob mir der große Durchbruch kurz davorstand. Doch statt der Durchbruch folgte der Zusammenbruch.

Die Entwicklung in einem sozialistischen Land erwies sich wie vormals beim Testlabor am Ende als Fehlentscheidung. Das Eis, auf dem ich gebaut habe, schmolz einfach dahin.

5 Softwareentwicklung im Deutschland der 80er Jahre

5.1 Kundenspezifische Testwerkzeugentwicklung

Als die SES im Sommer 1980 gegründet wurde, hatte sie lediglich zwei Kunden für den Test – Quelle in Nürnberg und Siemens in München. Schon während des I.T.S. Projektes hatte sich die Quelle IT-Leitung Siemens besucht und Interesse für den Prüfstand gezeigt. Der Entwicklungsleiter dort wollte so etwas Ähnliches für seine IBM Umgebung haben. Sie wusste jedoch, dass das Werkzeug nicht 1:1 auf ihre IBM Anlage übertragbar wäre. Zum einen war es größtenteils in der SPL Sprache von Siemens implementiert, zum zweiten lief es im BS2000 Dialogmodus und zum dritten war ursprünglich für den Test von BS2000 Module gedacht. Bei der IBM hat es zu diesem Zeitpunkt kein solches Time-Sharing System gegeben. TSO war noch nicht freigegeben. IBM Anwender mussten ihre Online Applikationen mit zusätzlichen Transaction Monitoring Systemen wie CICS oder IMS betreiben. Diese Systeme waren aber für die Software Entwicklung kaum geeignet. Für Quelle musste ich mir einen anderen Ansatz einfallen lassen.

5.1.1 Das Quelle IBM-Assembler Testsystem

Die damaligen IBM Betriebssysteme – OS und DOS – waren auf die Batchverarbeitung ausgerichtet. Sie konnten viele einzelne Batchprozesse nebeneinander ausführen aber nicht mehrere Benutzer gleichzeitig im Dialogbetrieb bedienen. Die Ergebnisse der Batchläufe wurden aufbewahrt und dem Benutzer auf Anfrage angezeigt. Entwickler haben ihre Programme im Batchmodus editiert und kompiliert. Nachdem die Kompilierung gelungen ist wurden die einzelnen Module zusammengelinkt und als Run Unit ausgeführt. Dazwischen bekam der Benutzer in den Jobprotokollen zu sehen, was geschehen ist. Es war nicht möglich in ein laufendes Programm, bzw. Run Unit, einzugreifen, wie dies im Prüfstand im Time Sharing Modus der Fall war um die Datenzustände zu verändern. Die Datenzustände mussten vor dem Test definiert und nach dem Test kontrolliert werden. Dazwischen konnte der Job nicht unterbrochen werden.

Zu Beginn habe ich zusammen mit dem Chefentwickler von Quelle eine Assemblerähnliche Sprache definiert um Testdatenwerte zu setzen und Testergebnisse zu prüfen. Maria Majoros und Erzebeth Tomka von SZAMOK haben für diese Testspra-

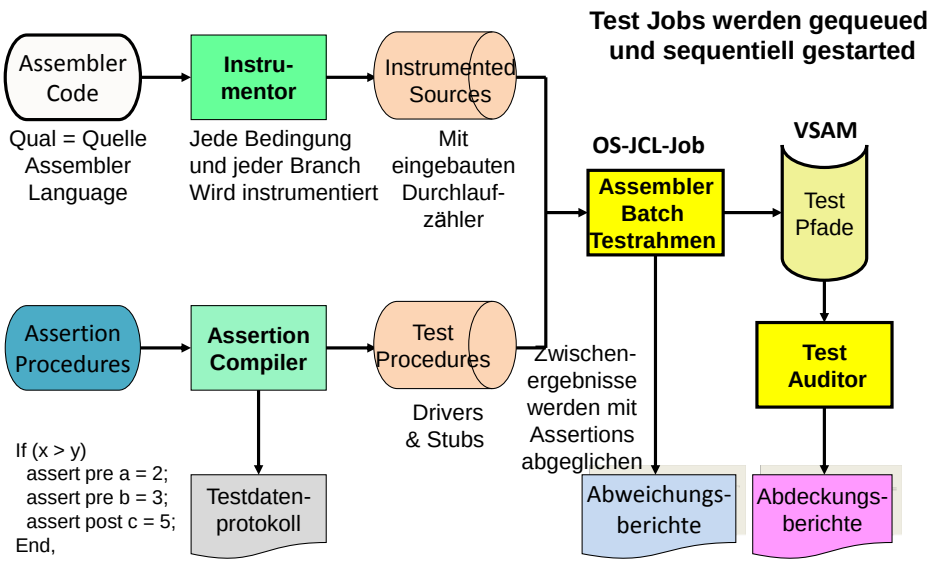
che einen Compiler entwickelt. Sie haben auch einen Instrumentor für die Quelle Assembler Sprache – QUAL – sowie einen Testmonitor für die Auswertung der Testüberdeckung gebaut. Parallel dazu habe ich einen eigenen Statischen Analysator konzipiert, der von einem neuen SZAMOK Team unter der Leitung von Gabor Jandrasics in Assembler implementiert wurde. Quelle hatte eine strenge Programmierkonvention gehabt. Der Leiter der Programmierer war einen Mann mit dem ich gut ausgekommen bin, ein Reserve Major der Bundeswehr. Er führte eine strenge Regie und legte großen Wert auf Disziplin. Die Programmierregel sollten genau kontrolliert werden und wer sie verletzte, sollte zur Rechenschaft gezogen werden. Die Programme sollten nicht nur richtig funktionieren, sondern auch richtig gestaltet sein.

Die Sammlung einzelner Batchprogramme:

- Statischer Analysator
- Instrumentor
- Assertion Compiler
- Testtreiber und
- Testmonitor

haben wir als Quelle Software Testsystem – QSTS – bezeichnet [Sned79].

Quelle Software Test System testet Assembler Module gegen eine Testspezifikation im Batchbetrieb



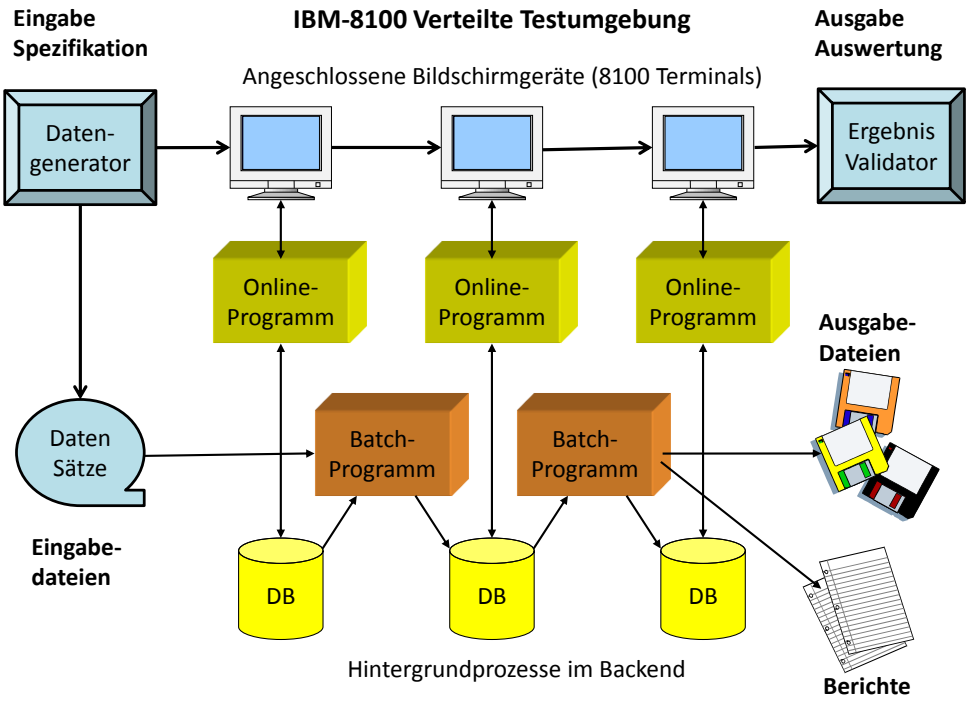
Problematisch war in diesem Projekt, das bis auf den Entwicklungsleiter – Major S. – keiner der Entwickler Englisch sprach. In dem Deutschland von damals war dies nicht üblich. Vom SZAMOK Team sprach nur Frau Tomka Deutsch und sie ist mitten im Projekt wegen eines Visumproblems ausgefallen. Sie musste durch eine deutschsprachige Programmiererin von der SzKI ersetzt werden, genau jene junge Frau die mich mit den Dissidenten zusammengebracht hat – Marta Máté. Sie konnte nicht nur gut Deutsch, sondern auch gut Assembler. Außerdem war sie blond und blauäugig, so dass sie von den Franken dort in Nürnberg gut auskam. Bei Quelle habe ich erst gelernt wie wichtig die persönlichen Beziehungen zum Kunde sind. Bis dahin war ich nur auf die Technik fokussiert. Jetzt sollte ich lernen mit Kunden umzugehen. Der Test des Testsystems durch die Quelle Mitarbeiter hat noch lange gedauert und es war nur den guten persönlichen Beziehungen zu verdanken, dass sie so viel Geduld mit uns hatten.

Die Quelle Entwickler haben getestet und die Fehler in den Werkzeugen aufgedeckt. Es war die Aufgabe der SkZI Mitarbeiterin, die gefundenen Fehler zu korrigie-

ren. Es gab auch im Laufe der Zeit weitere Anforderungen, was die Handhabe der Werkzeuge anbetraf. Es wäre unwirtschaftlich gewesen, das ganze Team dort in Nürnberg zu halten, außerdem hatten wir inzwischen andere Toolentwicklungsprojekte. Quelle bezahlte in Stufen nach dem Erreichen eines jeden Meilensteines. Die Strecke bis zum letzten Meilenstein – die endgültige Abnahme des Produktes – konnten wir nur schwer abschätzen. Drum habe ich mich entschieden, Frau Máté dort allein zu lassen und die anderen abzuziehen. Frau Máté konnte die meisten Probleme alleine lösen und wo sie Hilfe brauchte konnten die anderen beiden Frauen einspringen. Gabor Jandrasics von SZAMOK ist auch nach Bedarf dorthin gereist um sie zu unterstützen. Anfang 1981 hat Quelle das Testsystem endlich abgenommen und den verbleibenden Teil des Geldes bezahlt. Alleine hatte ich diese finanzielle Durststrecke nicht aushalten können. Als selbstständiger Unternehmer wäre ich längst pleite gewesen, aber jetzt hatte ich die ungarische Außenhandelsgesellschaft Metrimpex hinter mir. Das erste anwenderspezifische Toolprojekt konnte erfolgreich abgeschlossen werden.

5.1.2 Das R+V IBM-8100 Testsystem

Es folgten bald weitere Testprojekte. Über meine Seminare hat die R+V Versicherung von dem Prüfstand erfahren. Sie war gerade in Begriff ihre Mainframe Anwendungen auf IBM-8100 verteilten Rechner zu migrieren und sie brauchte für den Test der neuen COBOL Programme ein Testsystem. Einer der R+V Berater war ein ehemaliger Kollege von mir bei der H.I.S in Hannover. Er hat mich empfohlen und im Herbst 1980 bekam ich den Auftrag ein solches Testsystem für sie zu konzipieren. Das 8100-Testsystem sollte die gleichen Komponenten haben, wie das Quelle-Testsystem - einen statischen Analysator, ein Instrumentor, ein Testmonitor und ein Batchgenerator. Neu an diesem System waren die Dialogmasken für die Erfassung der Testdaten. Auf dem 8100-Rechner waren die Dialogfähigkeiten anders als auf dem Mainframe. Der Anwender konnte im Voraus die Testtransaktionen aufzeichnen und beim Test wiederholen. Die Inhalte der Ausgabemasken wurden gesichert und mit den erwarteten Maskeninhalten, die der Tester vorher erfasst hatte, verglichen. Damit haben wir das erste Capture/Reply Tool realisiert.



Mit dem 8100-Testsystem löste ich mich von dem reinen Modultest und widmete mich dem Integrationstest zu. Nicht einzelne COBOL Programme wurden getestet, sondern Sequenzen von Programmen. In diesem System haben wir zum ersten Mal Datenbanken generiert und anschließend validiert. Es war mein erster Schritt in Richtung eines allumfassenden Testsystems. Wie bei den Vorgängersystemen – Prüfstand und QSTS - wurde das 8100-Testsystem in der Sprache der getesteten Programme implementiert, nämlich in der 8100-COBOL-Sprache. Für dieses Projekt wurden zwei Entwickler von der SzKI zugewiesen – Lajos Frank und Imre Szálka. Diese beiden Entwickler waren Spitzenkräfte. Später nach der Wende sind sie zu Microsoft nach Redmond, U.S.A. ausgewandert. Ich habe die Toolkomponente mit Pseudo-Code entworfen und die zwei Entwickler haben die Entwürfe in COBOL Code umgesetzt. Im Gegensatz zum Quelle Projekt, in dem die Entwicklung größtenteils beim Kunde stattfand, konnten hier die Programme in Budapest geschrieben und auf dem dortigen Siemens Rechner kompiliert und teils getestet werden. Der Unterschied zwischen dem Siemens BS2000 Betriebssystem und dem 8100 Betriebssystem war nicht so groß. Den SzKI Entwicklern ist gelungen einen Testrahmen für die Simulation der

8100 zu bauen. Für den Unit-Test blieb der Code in der Standard-COBOL Sprache. Erst nach dem Unit-Test wurden die Testmodule nach Wiesbaden gebracht und in der echten 8100 Umgebung weiter getestet. Auf dieser Weise konnten wir die Entwicklungskosten halbieren. Die Reise- und Aufenthaltskosten übertrafen damals bei weitem die Personalkosten. Sie waren neben den Rechenkosten in Budapest die Hauptkostentreiber. Anderseits waren die ungarischen Entwickler froh wegen der Tagesgelder möglichst viel Zeit in Deutschland zu verbringen. Ich habe versucht eine Balance zwischen den beiden Standorten zu finden.

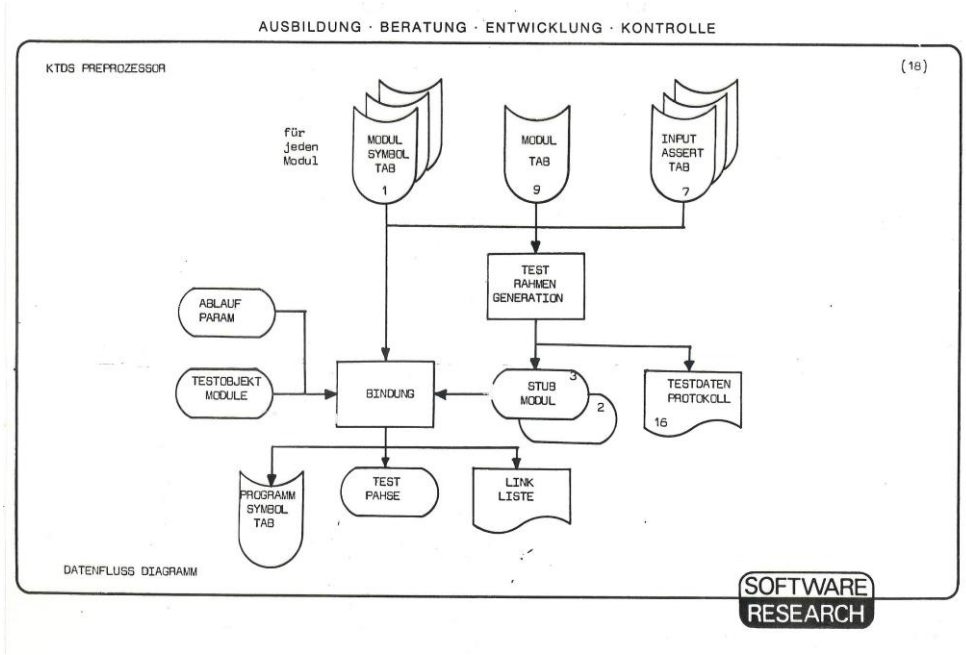
Da ich als Testberater in dem R+V Projekt selbst viel Zeit in Wiesbaden verbringen musste, nutzte ich die Zeit um mein viertes Buch „Software Qualitätssicherung“ zu schreiben. Abends nach der Arbeit saß ich immer auf einer Bank in dem Casinopark und schrieb auf Notizpapier. Dieses 1982 im Rudolf Müller Verlag erschienen Buch beschrieb nicht nur den Test, sondern alle qualitätssichernden Aktivitäten von der Prüfung des Fachkonzeptes bis zur Abnahme des fertigen Systems [Sned82]. Als durchgängige Fallstudie dienste den QS-Prozess, den ich im Begriff war bei der R+V einzuführen. Ich wollte den Prozess und die dazu gehörigen Werkzeuge für einen breiten Kreis zugänglich machen. Nach nur einem Jahr war das Buch ausverkauft. Es verkaufte sich viel besser als das Vorgängerbuch im gleichen Verlag über Softwareentwicklung. Es gab damals eine enorme Nachfrage nach Methoden der Software-Qualitätssicherung und ich galt al führender Experte auf dem Gebiet.

Leider wurde das 8100-Testsystem für den Anwender umsonst gebaut. Kurz nachdem die 8100 Programme in Produktion gingen, entschied die R+V die Migration abzubrechen und auf den Mainframe zu bleiben. Wie im Falle des I.T.S. Projektes hat die Zielmaschine sich als zu schwach erwiesen, die Last der Produktion zu tragen. Kurz danach hatte ÍBM die 8100 Maschinen völlig vom Markt zurückgenommen. Es hat sich als Fehlentwicklung herausgestellt.

5.1.3 Das Kienzle Taximeter Testsystem

Gleichzeitig zum R+V Projekt in Wiesbaden lief ein weiteres Testprojekt im Schwarzwald an. Die Firma Kienzle in Villingen hatte damals noch Taximeter hergestellt und die entsprechende Software dazu. Für ihre Softwareentwicklung nutzte sie eine maßgeschneiderte Programmiersprache PLM – Programming Language for Microprocessors, von der Technischen Universität Karlsruhe. Diese Sprache war speziell für die Programmierung von Mikrogeräte mit eingebauter Realtime Software konzipiert. In dieser Hinsicht war Kienzle ein Vorreiter in der deutschen Industrie. Es fehlte ihr jedoch die geeignete Testtechnologie um das korrekte Funktionieren der eingebauten Steuerungskomponente nachzuweisen. Über die Testseminare der GES

im Schwarzwald sind sie auf mich gestoßen. Ich wurde nach Villingen eingeladen um ihrem Team eine Lösung anzubieten. Dies war mein erster Ausflug in die Welt der embedded Realtime-Systeme, bis dahin hatte ich nur mit Informationssystemen zu tun gehabt. Ich bin gleich auf die vielen Einschränkungen gestoßen die es in dieser Welt gab – die eingeschränkte Speicherkapazität, die Zeitgrenzen, die mangelnde I-O Möglichkeiten und die strenge Genauigkeitskriterien. Es war für mich eine ganz neue Erfahrung. Dennoch habe ich die Herausforderung angenommen und mich in das Thema Mikroprogrammierung eingearbeitet.



In kurzer Zeit hatte ich den Plan für ein Mikrotestsystem ausgearbeitet. Da der Code sich wegen der Veränderung des Laufzeitverhaltens nicht instrumentieren lies, musste die Testüberdeckung auf anderer Weise gemessen werden. Es ging nur über die Analyse der Ein- und Ausgangssignale. Der Testmonitor musste in der gleichen PLM Sprache geschrieben werden wie die Testobjekte. Im Mittelpunkt stand die Testsprache mit Pre- und Post Assertionen. Eingangssignale wurden generiert und der Software zugefügt. Die Ausgangssignale wurden abgefangen und auf ihre Korrektheit geprüft, also das Gleiche, wie in den anderen Testsystemen, nur auf einer tieferen

Detaillierungsebene und unter den gegebenen Einschränkungen. Für diese Aufgabe habe ich Maria Majoros und Kati Erdös von der SzAMOK eingesetzt. Beide waren Mathematiker – Frau Majoros hatte den größten Teil des Quelle Testsystems entwickelt und Frau Erdös hatte früher Instrumente in der Petrochemie Industrie getestet. Neben ihrem Studium der Mathematik hat sie auch ein Ingenieurstudium abgelegt. Zusammen haben sie den Testrahmen für Kienzle gebaut und vor Ort getestet. Bei der Präsentation der Ergebnisse war der oberste Chef Herr Kienzle persönlich dabei. Er war sehr beeindruckt, wie das Tool verborgene Fehler in der Gerätesteuerung aufdeckte und meldete. Damals haben die oberen Manager, in diesem Fall sogar der Firmeninhaber sich mit der Technik beschäftigt. Sie wollten wissen, wofür sie ihr Geld ausgeben.

Die Entwicklungsmannschaft hatte ohnehin an dem Zustandekommen der Testsoftware mitgewirkt und war über alles informiert. Den Schwarzwaldtüftlern konnte man nichts vormachen. Sie wollten genau wissen, wie alles funktioniert. Es war eine Eigenschaft, die ich an den Menschen aus dieser Gegend schätzen gelernt habe. Sie sind Meister ihres Handwerks und nehmen ihre Arbeit sehr ernst. Das erwarten sie auch von anderen die für sie arbeiten. Wenn man seine Arbeit genau nimmt, hat man mit ihnen keine Probleme. Wehe jedoch, wenn man sich seiner Sache nicht 100 % sicher ist. Sie merken das sofort und man ist bei ihnen unten durch.

Wir haben es jedenfalls geschafft, ihre Erwartungen zu erfüllen und ich konnte mit gutem Gewissen abziehen. Im Frühjahr 82 haben Mitarbeiter der Firma über den Einsatz des fertigen Kienzle Test und Dokumentationssystems auf der ersten ACM Tagung Qualitätssicherung in Neubiberg berichtet [Göll82]. Kurz danach ist die Firma in finanziellen Schwierigkeiten geraten und musste das Geschäft mit den Taximetern aufgeben. Schon wieder war eine große Leistung umsonst. Vielleicht haben die beteiligten Ingenieure daraus Erkenntnisse gewonnen die sie woanders anbringen konnten. Von mir aus habe ich erkennen müssen, das einen Großteil der produzierten Software auf der Strecke bleiben muss, damit der Rest weiterkommt. Das hat die Marktwirtschaft so an sich. Technische Hochleistung ist kein Garant für den wirtschaftlichen Erfolg.

5.1.4 Das Tornado Lagerhaltungstestsystem

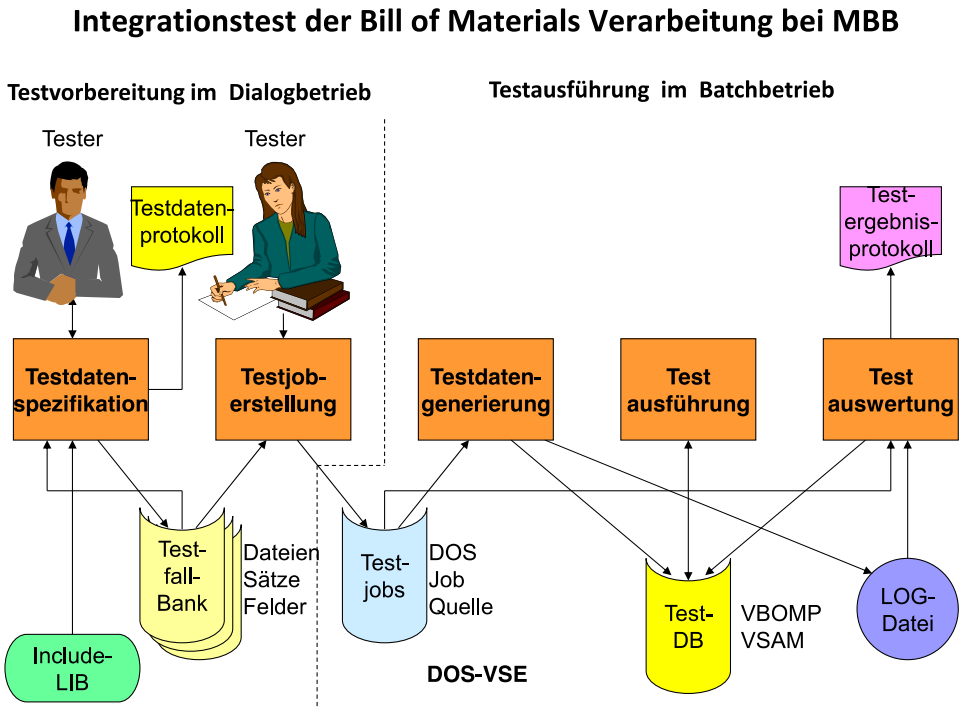
Das größte Projekt, das wir zu jener Zeit hatten, sollte aus der Rüstungsindustrie kommen, nämlich von der MBB in Ottobrunn. MBB war zu dieser Zeit der Hauptlieferant der Bundeswehr und hatte den Auftrag, Teile des damaligen Tornado Jägerbombers zu liefern. Die anderen Teile kamen aus anderen NATO Ländern. Zusammengestellt wurde das Flugzeug in dem Werk in Augsburg. Der dortige IT-Leiter –

ein gewisser Herr O. - kannte mich von den Seminaren und Konferenzen und lud mich ein für ihn ein Integrationstestsystem zu konstruieren. Bei diesem Projekt ging es um die Lagerhaltung der Flugzeugteile. Der Test sollte auf einem kleineren IBM Mainframe unter dem Betriebssystem DOS-VSE in den Werkshallen in Augsburg stattfinden. Es sollten vor allem die Schnittstellen zwischen den Teilsystemen getestet werden. Ich habe dazu wiederum eine Reihe Tools konzipiert:

- Ein Generator um die VSAM Dateien zu generieren
- ein Prozessor um die Stammdateien und Schnittstellendateien vorzubereiten,
- ein Instrumentor um die PL/I Programme zu instrumentieren, allerdings nur auf der I/O Ebene,
- ein Postprozessor um die Abläufe und die Testüberdeckung auszuwerten und
- ein Validator um die neuen Schnittstellendateien mit den alten Dateien abzugleichen.

Da MBB kaum eigene Programmierkapazität hatte, - das Lagerhaltungssystem wurden von externen Auftragnehmern entwickelt - habe ich angeboten, die Werkzeuge von den Ungarn machen zu lassen. Ich habe vom Anfang an kein Hehl daraus gemacht, dass wir die Tools in Budapest implementieren aber in Augsburg testen würden. Implementiert wurden die Werkzeuge in PL/I von einem gemischten Team aus SzAMOK und SzKI Mitarbeitern. Da ich auch anderweitig zu tun hatte, heuerte ich einen deutschen Mitarbeiter an um das Projekt vor Ort in Augsburg zu koordinieren.

Die VSAM Dateien mit den Materialsätzen wurden aus den Input-Assertion-Prozeduren vor dem Test generiert. Die gleichen Dateien wurden nach dem Test gegen die Output-Assertion-Prozeduren validiert. Dazwischen hat der Testmonitor die CICS-Transaktionen simuliert und Eingabedaten den Programmen zugeführt. Die Ausgabedatenströme wurden abgefangen und in Bewegungsdateien abgelegt um später geprüft zu werden. Die Testüberdeckung der Testobjekte fand auf der Schnittstellenebene statt. Alle ausgeführten IO-Operationen sowie alle Modulaufrufe wurden registriert und verglichen mit der Summe aller möglichen IO-Operationen und Modulaufrufe. Ich habe dies als Interface-Coverage bezeichnet [Sned82]. Da es damals noch keine richtigen Integrationstestsysteme wurde ich gezwungen zu improvisieren. Das Ganze war für mich eine riesende Herausforderung, die ich erst viele Jahre später im Begriff bekam. [Sned08]



Die einzelnen Bestandteile des Integrationstestsystems wurden von Mitarbeitern des SzAMOK Instituts gefertigt. Als die Testmodule fertig waren, haben die Ungarn sie nach Augsburg gebracht und auf dem dortigen IBM Rechner mit dem DOS-VSE Betriebssystem installiert. Jeden Tag, wo sie dort getestet haben, mussten sie durch die Werkshalle gehen, in der der Tornado zusammengebaut wurde. Eines Tags, als das Projekt schon weit vorangeschritten war, hat der Wächter am Eingangstor die Pässe kontrolliert. Er ist aus den Wolken gefallen als er entdeckte, dass die Leute, die dort ein- und ausgingen, aus Ungarn – einem sozialistischen Staat kommen. Die Nachricht ging sofort an die Firmenleitung und das Testen in Augsburg wurde gestoppt. Dennoch wollte der Herr O. das Projekt unbedingt abschließen. Ich musste Rechenkapazität bei der IBM in München bestellen damit die Ungarn dort weiter testen konnten. Mein deutscher Mitarbeiter vor Ort hat die getesteten Werkzeuge anschließend in Augsburg installiert und dort zusammen mit den MBB Mitarbeitern den Abnahmetest gemacht. Es hat deshalb alles länger gedauert und mehr gekostet als geplant. Da auch dieses ein Festpreisprojekt war musste ich die zusätzlichen Kosten selber tragen.

Endlich wurden die Werkzeuge abgenommen und MBB hatte ihr Integrationssystem für die Tornado Lagerhaltung. Eigentlich lief alles wie geplant. Das Integrationstestsystem lief zuletzt einwandfrei. Das Projekt hatte jedoch Folgen. Herr O. wurde von seinem Posten enthoben und ist nach Hamburg gezogen um dort für ein Beratungshaus zu arbeiten. 20 Jahre später, als er wieder bei MBB war – die inzwischen zu Daimler-Benz gehörte - hat er mich gerufen, um als Expertenzeuge in einem Prozess mit einem Kunden aufzutreten. Für mich war das auch das Ende meiner Arbeit für die deutsche Rüstungsindustrie. Bis dahin hatte ich in der Firmenzentrale in Ottobrunn zahlreiche Kurse gehalten und sie in Sache Software Qualitätssicherung beraten. Jetzt gab es ein Lex Sneed. Ich durfte nie wieder die Firma MBB oder irgendeinen anderen Rüstungsbetrieb betreten. Ich wurde zur Persona non grata. Noch Jahre später, als die Bundeswehr erwog, die Softorg Werkzeugfamilie einzusetzen, erwies sich dies als Hinderungsgrund. Eine weitere Folge des Augsburg Projektes war, dass mein Telefon ab diesem Zeitpunkt abgehört wurde. Immer, wann ich nach Ungarn telefonieren wollte, erfolgte am Anfang eine Pause, bis das Abhörgerät eingeschaltet wurde. Ich hatte die Ehre von dem BND überwacht zu werden.

Das MBB Projekt war die letzte in der Reihe kundenspezifische Toolentwicklung. Von da an habe ich mich auf den Vertrieb der eigenen Standard-Werkzeuge konzentriert – SoftTest für den Unit-Test und SoftInt für den Integrations- und Systemtest. Bald hatten wir genug Kunden für diese Werkzeuge – vor allen PL/I Betriebe – um die Weiterentwicklung zu finanzieren. Unser Schwerpunkt verlagerte sich vom Test zum gesamten Software-Entwicklungszyklus von der Anforderungsanalyse bis zum Systemtest.

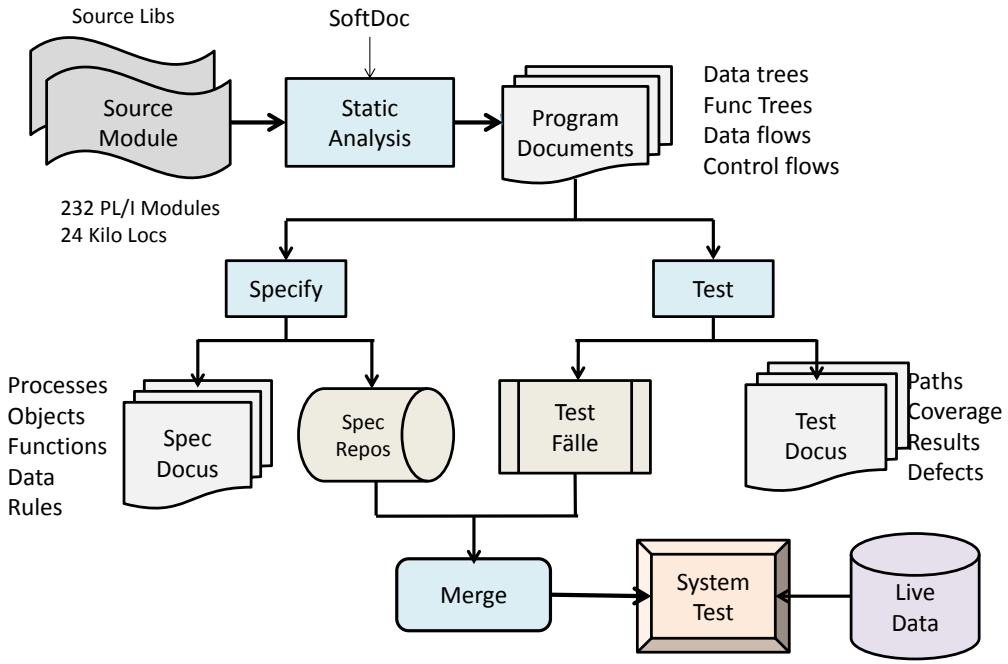
Das Gipfel dieser frühen Testperiode, wo wir unser Geld hauptsächlich mit der Entwicklung maßgeschneiderten Testwerkzeuge verdiente war die erste German ACM Tagung an der Bundeswehrhochschule in Neubiberg mit dem Thema „Software-Qualitätssicherung“. Diese war eigentlich die Geburtsstunde der deutschen Testtechnologie. Alle führenden Wissenschaftler die mit Testen und Qualitätssicherung zu tun hatten waren dabei, u.A. die Gründer der Firma SQS in Köln – Rudolf van Megen und Heinz Böhm – sowie die ersten deutschen Testforscher von der TU Karlsruhe – Voges und Gmeiner. Als Keynote-Speaker habe ich Les Belady von der IBM und Prof. Ramamoorthy von der Universität Califonia eingeladen. Die Konferenzkosten konnte ich aus dem Budget der SES decken. Es war eine der besten Investitionen die ich je betätigt habe. Die SES wurde dadurch bundesweit bekannt [Wieh82].

5.2 Software Reengineering bei Bertelsmann

Zu Beginn der 80er Jahren war ich bemüht, Zeit für unsere neuen Entwicklungswerkzeuge zu gewinnen. Unser Einkommen habe ich mit Seminaren, Beratung und den Testprojekten gesichert. Mein Ziel war jedoch den ganzen Software Entwicklungsprozess zu automatisieren von der Analyse bis zum Abnahmetest. In Budapest ging die Arbeit an diesem Vorhaben langsam voran, es brauchte Zeit, viel Zeit. Während dieser Zeit musste ich meine Geschäftspartner dort bei der Stange halten, d.h. es mussten zumindest alle halbe Jahre Devisen auf die Konten der Außenhandelsgesellschaft fließen. Dies schaffte ich über die Testprojekte und ab 1981 durch den Verkauf unseres ersten IBM-Mainframe Werkzeugs SoftDoc. SoftDoc war eine Neuausgabe des statischen Analysators vom Prüfstand. Es lieferte verschiedene Sichten auf die Assembler, PL/I und COBOL Programme – die Datenstruktur, die Ablaufstruktur und den Datenfluss. Außerdem deckte es Mängel in dem Code auf und rechnete einige Komplexitäts- und Qualitätsmetriken aus. Die Anwender konnten es benutzen, um ihre Programme besser zu verstehen und die Codequalität zu prüfen [Jand83]. Die ersten Käufer dieses Tools waren vorwiegend PL/I Anwender, die besondere Probleme hatten ihre komplexe Software zu verstehen. Die Fachartikel über statische Analyse haben mir sehr geholfen die Anwender auf die Vorteile von SoftDoc aufmerksam zu machen. Zu diesem Zeitpunkt ist mir auch aufgefallen, dass die DV-Welt in zwei Welten zerfällt. In einer Welt waren die Chaoten, die Sprachen der 4. Generation vorzogen um ihre Applikationen quick und dirty zu implementieren. In der anderen Welt waren die Systematiker die nachhaltigen Systeme nach den Prinzipien der Software Engineering anstrebten. Meine Kunden waren die Systematiker. [Sned82a].

Der große Durchbruch erfolgte erst Ende 1981 als die Firma Bertelsmann auf uns zukam. Den DV-Leiter – Herr B - habe ich auf einer der CDI-Struktokongresse in Frankfurt kennengelernt. Er war zu der Zeit schon eine bekannte Persönlichkeit in den deutschen Mangerkreisen. Ich hatte inzwischen in der deutschen DV-Szene einige Anhänger und er war einer davon. Meine Bücher waren ihm bekannt – es waren zu diesem Zeitpunkt erst vier – und er hat mit seinen Mitarbeitern in Gütersloh selber an einem Buch gearbeitet – das „Software Engineering Modell Bertelsmann“ [Bend83]. Er hat mich gebeten das Vorwort dazu zu schreiben. Bertelsmann hatte gerade ein großes Projekt mit einem führenden IT Lieferant als Auftragnehmer gehabt der aber wegen Kostenüberschreitung abgebrochen wurde. Es ging um das Kernsystem für die Warenauslieferung. Das System wurde mit PL/I unter CICS programmiert und nutzte ein ADABAS Datenbank. Es war ein Festpreisprojekt und wie es oft der Fall ist, hatte der Auftragnehmer den Aufwand unterschätzt. Der vereinbarte Preis reichte nicht aus um das System fertigzustellen. Der Auftragnehmer wollte mehr Zeit und mehr Geld, der Auftraggeber wollte nichts mehr als die vertraglich vereinbarte Summe bezahlen. Der Fall endete vor dem Gericht. Der Auftragnehmer zog sich unvollendeter Dinge

zurück. Der Auftraggeber blieb auf einem Haufen nicht fertiggestellter PL/I Programme sitzen –manche waren nicht einmal kompiliert – zurückgelassen. Bertelsmann war jedoch ein COBOL-Shop und die eigenen Entwickler konnten mit dem fremden PL/I Code nicht viel anfangen. Also kam der DV-Leiter auf mich zu und hat mich um Hilfe gebeten. Ich sollte mit meinem Team aus Budapest seine Investition retten, d.h. den Code sanieren, in COBOL umsetzen, austesten und nachdokumentieren. Dies war genau das Projekt auf das ich gewartet habe – die Gelegenheit, meine neuen Werkzeuge im Verbund einzusetzen.



Sanierung des Bertelsmann Versandsystems

Es gab Verhandlungen zwischen der Geschäftsführung von Bertelsmann und den Vertretern der ungarischen Außenhandelsgesellschaft. Der Bertelsmann IT-Leiter, Herr B. kam sogar mit seinem Vorstand nach Budapest. Es sollte ein größeres Projekt werden, das bis zu zwei Jahre dauern soll. Der Kostenrahmen wurde wiederum mit einem Festpreis fixiert. Daneben gab es einen variablen Teil nach Aufwand für zusätzliche Aufgaben. Beide ungarische Institute waren involviert. Die Mitarbeiter von SzAMOK sollten die PL/I Programme nachdokumentieren und testen. Das Team von der SzKI sollte sie mit Hilfe von SofSpec nachspezifizieren und in COBOL umsetzen.

Die neuen COBOL Programme sollten dann gegen die alten PL/I Programme getestet werden. Die Arbeit sollte teils in Gütersloh, teils in Budapest stattfinden. Für den Test konnten wir den IBM Rechner in Gütersloh benutzen.

Der Bertelsmann Reengineering Projekt war das größte Projekt dieser Art, das es in Deutschland bisher gegeben hat. Es hatte bis dahin nur Konversionsprojekte gegeben wobei die Applikationssoftware von einem Rechner zum anderen portiert wurde. Dabei musste der Code angepasst werden, aber noch keiner hat versucht, den Code gleichzeitig zu sanieren und in eine andere Sprache zu versetzen. Das war eine echte Pionierleistung, vor allem, wie wir es gemacht haben. Wir haben die strukturierte Spezifikation aus der Dokumentation der alten PL/I Programme gewonnen und daraus wieder neue Programme in COBOL geschaffen. Ein signifikanter Teil der Spezifikation war die Testspezifikation in einer Prädikatensprache mit Vor- und Nachbedingungen. Aus dieser sogenannten Assertionssprache wurden die Testprozeduren automatisch erzeugt. Der Test der sanierten und konvertierten Programme lief weitestgehend automatisch ab. Es wäre mit der Menge der zu testenden Objekte in der Kürze der Zeit gar nicht anders gegangen. Dieses Projekt war nicht das erste Reengineering Projekt in Deutschland, es war auch ein Meilenstein in der Testautomation [Majo82].

Das Projekt hatte die üblichen Stolpersteine, es gab Schwierigkeiten mit der Komptabilität der Datentypen in den beiden Sprachen und Probleme mit der Datenübertragung. Manches hat auch mehr Zeit gebraucht als geplant. Der Kunde war aber geduldig und kam uns entgegen. Die Zusammenarbeit vor Ort hat gut geklappt. Es gab auch – wie immer – Performanz-Engpässe aber wir konnten sie umgehen. Zum Schluss waren alle zufrieden. Die Bertelsmann Mitarbeiter, die das System übernehmen sollten, haben in Gütersloh mitgewirkt und waren mit unseren Tools vertraut. Sie konnten diese nachher übernehmen und damit weiterarbeiten. Über die Ergebnisse des Projektes habe ich im Sommer 1983 einen Artikel verfasst der 1984 in der IEEE Software Magazine erschien [Sned84]. Wir hatten 232 PL/I Programme mit 250.000 Codezeilen erfolgreich saniert und konvertiert. Später, als ich 1987 die dritte Software Maintenance Konferenz in Austin, Texas besuchte, wurde ich wegen dieses Projektes als Pionier der Reengineering Technologie gefeiert. Neun Jahre später wurde ich von der IEEE Computer Society für meine Pionierleistung auf dem Gebiet der Software Reengineering ausgezeichnet. Diese Anerkennung hatte ich dem Herrn B. und dem damaligen Bertelsmann Projekt zu verdanken.

5.3 SoftSpec schafft den Durchbruch

Schon während des Bertelsmann Projektes haben wir unsere ersten SoftSpec Kunden gewonnen. Aufgrund der Erfolgsberichte über das Projekt und die Empfehlungen vom Herrn B. haben zwei weitere Unternehmen die Tools SoftSpec, SoftDoc und SoftTest bestellt. Eine war die Württembergische Feuerversicherung in Stuttgart und die andere BMW in München. Ich bekam in Budapest mehr Mitarbeiter zugestellt und in Deutschland habe ich einen Landsmann von mir angestellt der sich mit dem IBM Mainframe gut auskannte um die beiden neuen Kunden technisch zu betreuen. Der Landsmann aus New York – Paul Orgas – sollte noch 10 Jahre bei mir bleiben. Ich selbst bin immer hin und her zwischen Gütersloh, Stuttgart, München und Budapest gereist. Es begann für mich eine sehr anstrengende Zeit. Ich musste nicht nur die Toolentwicklung steuern, sondern auch die Toolkunden in der Nutzung der Tools einschulen und beraten. Wie Bertelsmann wollte BMW damals Software Engineering auf breiter Front einführen und hat mich beauftragt, die ganze Entwicklungsmannschaft zu schulen. Ende 1983 stand die SES gut da. Wir hatten es geschafft mehrere einflussreiche Kunden zu sammeln. Die Seminare wurden gut besucht und unser Name stand in Verbindung mit dem Begriff „Software Engineering“.

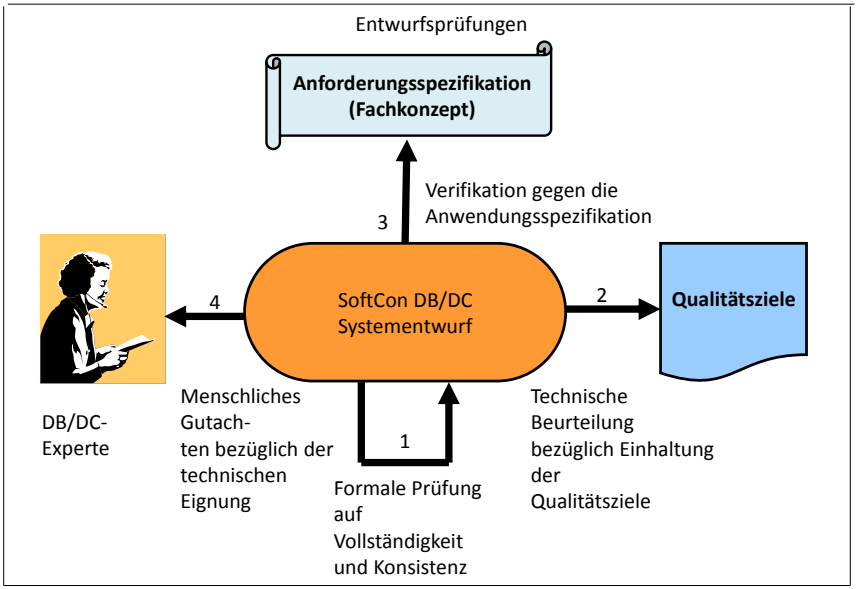
5.3.1 Württ-Feuer baut mit SoftSpec eine eigene Umgebung auf

Die Württ-Feuer setzte SoftSpec ein um ihr neues Sachversicherungssystem zu spezifizieren. Die Fachbereichsanalytiker dort kamen damit gut zurecht. Es war genau das was sie brauchten um ihre Fachkonzepte zu erstellen. Sie wollten jedoch nach Fertigstellung des Fachkonzeptes einen eigenen Weg einschlagen. Die DV-Gruppe baute eine eigene Programmierumgebung auf, die auf SoftSpec aufsetzte und Programme für den IMS-DB/DC Betrieb generierte. Der Chefprogrammierer dort hat sogar eine eigene Programmiersprache entwickelt um die vielen IMS-COBOL Bausteine zu erzeugen. 20 Jahre später wurde ich mit dieser Sprache konfrontiert als es darum ging die Software der Württembergischen Versicherungen zu vermessen. Erika Nyary mit ihrem SoftSpec Team und mein Landsmann, Paul Orgas, haben sie dabei unterstützt [Nyar83]. Ich wollte damit nichts zu tun haben, denn ich war schon immer gegen die Nutzung einer non-standard Programmiersprache. Ich verfolgte eine Strategie des Gerüstbaus. Danach sollten Spezifikation und Entwurf nur als Gerüst für den Bau des endgültigen Systems in einer Standardsprache dienen. Nachdem der Bau steht könnte das Gerüst abgebaut werden, die Programme sollten in einer allgemein bekannten und international genormten Sprache weiterbestehen. Ich habe Anwender immer vor der Nutzung exotischer oder eigener Sprachen gewarnt. Der Tag würde kommen, wenn sie die Sprache ablösen müssen und keine Sau kennt sich damit aus. Der Tag ist auch 22 Jahre später gekommen. Ich musste als Aufwandsschätzer für die ANECON schätzen was es kosten würde diese selbstgestrickte Lösung abzulösen. Es gab nur zwei Alternative – das System wieder von vorne neu zu entwickeln oder es zu kap-

seln und in die neue Umgebung einzubinden. Der Kunde hat entschieden es zu kapseln. Die Neuentwicklung wäre viel zu teuer gewesen.

5.3.2 BMW entwickelt mit SoftSpec und SoftCon ein Personalverwaltungssystem

BMW wollte wie Bertelsmann den standarisierten Weg mit unserem Tool SoftCon für den technischen Entwurf weitermachen. Sie hat es eingesetzt ihr neues Personalverwaltungssystem – APIS – zu implementieren. APIS baute auf IPAS auf, ein Standardsystem von der ADV-ORGA für die Personalverwaltung, aber es gab darum herum etliche Satellitensysteme welche die Sonderanforderungen der Personalabteilung abgedeckt haben. Interessanterweise wurde IPAS als COBOL Quellencode ausgeliefert. Der Anwender konnte diesen Code beliebig verändern und eigene COBOL Anweisungen einfügen. Der Haken dabei war, dass beim nächsten Release der Anwender sämtliche bisherige Änderungen wieder einbauen mussten. Das hat viel Aufwand gekostet und war recht fehlerhaft. Für das umfangreiche Projekt hatte BMW viele externe Entwickler engagiert. Auch wir haben einige Mitarbeiter von der SzKI bereitgestellt. Darauf werde ich später zu sprechen kommen.



In dem BMW Projekt wurde SoftSpec in Zusammenhang mit einem selbst entwickelten IMS Data Dictionary System eingesetzt. Die spezifizierten Geschäftsobjekte

wurden samt ihrer Datenattribute und Beziehungen in ein unternehmensweite Entity/Relationship Modell überführt. Dort wurden sie als Ausgangsbasis für andere Projekte verwendet. Zu diesem Zweck hat das Team von Frau Nyary eine allgemeingültige Exportschnittstelle implementiert über die der Inhalt der SoftSpec Datenbank in andere Tooldatenbanken übernommen werden könnte. Später als die SoftSpec Entwicklung eingestellt wurde kam dies den SoftSpec Anwendern sehr zugute [Sned84].

Ich war bei BMW sehr bemüht den Übergang vom Fachkonzept zum technischen Entwurf möglichst reibungslos und ohne Informationsverlust zu schaffen. Aus den fachlichen Entitäten – Geschäftsprozesse, Vorgänge, Funktionen, Geschäftsobjekte, Datengruppen und Datenattribute – sollten technischen Entitäten werden – Jobs, Transaktionen, Programme, Module, Prozeduren, Masken, Datensegmente, logische und physische Datenbanken. Es war eine große Herausforderung an der ich die Zähne ausgebissen habe. Das erste lauffähige CICS/COBOL/DB2 Anwendungssystem wurde erst 1988 bei der Bundesbahn generiert. Dann war es für den Markt fast zu spät. Ich hatte die Zähne an diesem Problem ausgebissen.

Zu dieser Zeit als ich bei BMW und Bertelsmann gearbeitet habe, kamen weitere Werkzeugkunden hinzu. Meine Strategie, die Werkzeuge unabhängig voneinander einsetzen zu können zahlte sich aus. Anfang 1984 hatten wir schon 15 Werkzeugkunden, die meisten für SoftDoc und SofTest, darunter solche Firmen wie Bertelsmann, Württ-Feuer, BMW, MBB, Allianz, Volksfürsorge, Bundesrechenanstalt Wien, Creditanstalt Wien, TÜV in Essen, ÖVA in Karlsruhe, Kommerzbank, Westdeutsche Landesbank, die Börse in Frankfurt, Hoechst Chemie. Wir bekamen von ihnen die jährlichen Wartungsgebühren für die Nutzung der Produkte. Dazu kamen die Projekte, die wir bei Bertelsmann, Württ-Feuer und vor allem bei BMW hatten. Ich verbrachte einen Großteil meiner Zeit in München bei der BMW, wo ich im Projekt mitgearbeitet habe und wöchentlich über Software Engineering vorgetragen habe, sowie bei Bertelsmann in Gütersloh wo die Werkzeugentwicklung vorangetrieben wurde.

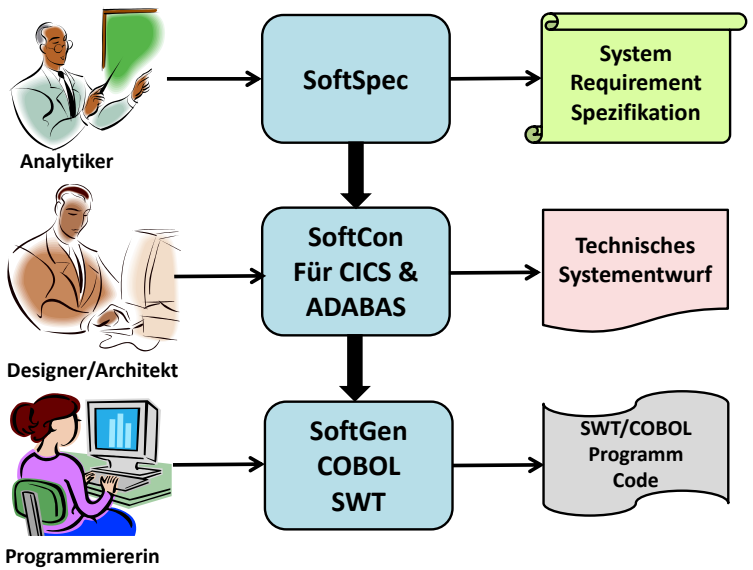
Nebenbei hielt ich jeden Monat ein öffentliches Seminar an verschiedenen Orten in der Bundesrepublik. Die Seminare zu den verschiedenen Themen des Software-Engineering waren zu diesem Zeitpunkt voll ausgebucht. Diese waren eine weitere Einnahmequelle und zugleich die beste Werbung für die Softorg Produkte. Die meisten Werkzeugkunden waren frühere Seminarteilnehmer. Schließlich haben wir allein in jedem Quartal mehr als 5000 Software Engineering Newsletters versendet. Wir hatten damals eine große Adressen-Datenbank aufgebaut. Jedes Mal als ich selbst an einem Seminar oder einer Konferenz teilnahm habe ich die Teilnehmerliste mitgenommen. (siehe Newsletter Beispiel)

5.3.3 Die Deutsche Bank kommt hinzu

Im Jahre 1984 kam ein wichtiger Softorg Kunde dazu – die Deutsche Bank. Sie hatten ein neues Projekt für die Verarbeitung der Emissionsdaten und wollten es probieren, mit den Softorg Tools durchzuführen, angefangen mit SoftSpec für die Anforderungsanalyse und - Dokumentation. Im August begannen wir mit der Installation von SoftSpec und SoftCon. Ich sollte das Projekt betreuen und bin in der ersten Zeit fast jede Woche nach Eschborn bei Frankfurt gefahren.

Die Spezifikation des Emissionssystems lief gut an. Die Analytiker der Bank haben schnell gelernt mit SoftSpec umzugehen und nach zwei Monaten stand die Anforderungsspezifikation. Danach wurde SoftCon installiert und der Systementwurf generiert. Hier stießen wir auf die ersten Probleme. Zum einen hatte die Deutsche Bank das Datenbanksystem ADABAS von der Software AG. Wir müssten SoftCon erweitern um die ADABAS Data Dictionary zu bevölkern und ein ADAWAN Schema zu generieren.

Vertikale Arbeitsteilung bei der DeutschenBank



Zum andern war die Deutsche Bank Kunde der BonnData – ein Softwarehaus aus Bonn und setzte deren Tool SWT ein, um COBOL Programme zu generieren. Zu diesem Zeitpunkt gab es viele sogenannte Sprachen der 4. Generation auf dem Markt und SWT zählte dazu [Gutz88]. Statt Standard COBOL müssten wir SWT-COBOL

aus SoftCon heraus produzieren. Dies erforderte von uns einen zusätzlichen Aufwand. Schon wieder war ich gezwungen mich mit einer fremden Non-Standard Sprache auseinander zu setzten. Da die BonnData Leute im Projekt uns als Konkurrenten ansahen, waren sie nicht besonders kooperativ. Es ist immer schwierig, wenn verschiedene Softwarehäuser im gleichen Projekt arbeiten.

Ich war im Begriff zu lernen, dass jeder Anwender eine einmalige Entwicklungsumgebung hat, mit Komponenten von unterschiedlichen Lieferanten. Es gab kaum ein Anwenderbetrieb, der nur die Standard IBM Sprachen - COBOL und PL/I einsetzte. Wir konnten deshalb unseren Code-Generator nicht verwenden. Wir mussten den des fremden Lieferanten in unseren Entwicklungszyklus einbauen – zwischen SoftCon und SoftDoc. Wir machten es auch, aber der von SWT generierte Code passte überhaupt nicht zu SoftDoc. Er war voll mit GO-TO Verzweigungen und benutzte viele künstliche Datenvariabel. Meine Vorstellung von generierten Code war, dass er so aussehen sollte als ob ein guter Entwickler ihn geschrieben hätte – strukturiert, formatiert und kommentiert mit sprechenden Namen, so, wie es ich in meinen Seminaren gelehrt habe. Ich dachte an die Nachhaltigkeit der Software. Es sollte für den Anwender möglich sein den Code zu übernehmen und weiterzupflegen, d.h. der generierte Code sollte auf eigenen Beinen stehen, ohne das Gerüst der Entwicklungswerkzeuge. Die Anwender haben das anders gesehen. Sie wollten nur möglichst schnell zu einer Lösung kommen die den momentanen Bedarf befriedigt.

Es gehörte zur Strategie der meisten Toollieferanten jener Zeit, die Anwender von sich abhängig zu machen. Es sollte für ihn unmöglich, oder nur mit hohen Kosten möglich sein aus der Abhängigkeit von ihren Werkzeugen loszukommen. In dieser Hinsicht hatten sie etwas gemeinsam mit Drogenhändler. Erst abhängig machen und dann kassieren. Es gab Reibereien zwischen mir und dem Generatorlieferanten. Da aber die anderen Programme der Bank alle in SWT-COBOL waren, hatte ich keine Chance dagegen anzukommen. Die Manager der Bank haben sich um die Nachhaltigkeit ihrer Software nicht gekümmert. Im Gegenteil, der zuständige IT-Leiter – der von einer namhaften Beratungsfirma kam, erklärte mir seine persönliche Erfolgsstrategie „Up and Out“. Man steigt in eine Firma ein, verursacht einen großen Wirbel, produziert einen Scheinerfolg und verlässt die Firma ehe die Folgen seines Tuns offensichtlich werden. So machte man Karriere in der Marktwirtschaft. Ich hatte noch vieles zu lernen.

Allen Hindernissen zum Trotz, haben wir das Emissionsprojekt bis zum Systemtest in sechs Monaten durchgezogen. Wir haben die SWT-COBOL Programme schlecht oder recht generiert und mit SoftTest getestet. Für den Test in Eschborn habe

ich einen neuen Mitarbeiter gefunden – Dr. Manfred Gerish aus Dresden. Er ist ihm gelungen über die damalig von Franz Josef Strauß ausgehandelte Austauschvereinbarung aus der DDR herauszukommen. Er hatte dort bei Robotron in Dresden gearbeitet und lernte mich über die Zusammenarbeit mit den Ungarn kenne. In der DDR hatte er bereits zusammen mit einem Kollegen dort ein Buch über Softwareentwurf geschrieben in dem er viele meine früheren Schriften zitierte[Geri84]. Als er in Westdeutschland war, hatte er sich gleich an mich gewandt. Er war technisch versiert, gerade der richtige Mann für den Test der Emissionsprogramme. Mit dem Tool SofTest hat er sich gleich angefreundet und wirkte sogar an der Weiterentwicklung mit. Der Test hat noch länger gedauert als gedacht. Ich konnte mich zurückziehen und das Projekt dem Dr. Gerisch überlassen. Bis Sommer 1985 war das Emissionssystem im Einsatz – 3 Monate später, als geplant, aber immerhin erfolgreich. Wie immer war es der Test der so lange gedauert hat. Die Zeit die man zum Testen braucht wurde immer wieder unterschätzt, auch von mir. Die Tools SoftSpec und SoftTest hatten sich bewährt und blieben noch lange Zeit bei der Bank im Einsatz. Dr. Gerisch übernahm ein anderes SoftTest Projekt bei einem neuen Kunden in München. Später ist er selbstständig geworden.

5.3.4 Weitere Entwicklung bei Bertelsmann

Zu Beginn des Jahres 1985 bin ich zwischen Gütersloh, Frankfurt, München und Budapest hin und her gependelt. In Gütersloh gab es nach dem erfolgreichen Abschluss der VVA Projektes weitere Entwicklungsprojekte in Bereich Versand. Inzwischen kamen die Analytiker dort mit SoftSpec allein zu recht. Es wurde zum Standard-Werkzeug zur Erstellung der Fachinhaltsbeschreibungen. Ein Projekt nach dem Anderen wurde mit SoftSpec spezifiziert und an die DV-Abteilung zur Implementierung übergeben. Die Vision des Herrn B eine Art innerbetriebliche Softwarefabrik mit einem standardisierten Entwicklungsprozess aufzubauen war im Begriff Wirklichkeit zu werden. Das „Software-Pipeline“, so wie es hieß, stand schon.

Das Bertelsmann Modell für Software Engineering stützte sich zunehmend auf die Softorg Werkzeuge SoftSpec und Softcon. Da SoftGen noch nicht fertig war, wurde der endgültige COBOL Code mit dem SoftCon Pseudo Code und Datenstrukturen als Vorlage manuell fertiggestellt. Die Entwickler brauchten nur den Pseudocode in COBOL Code zu übersetzen. Später als SoftGen dazu kam, konnten sie den PseudoCode auf Knopfdruck in COBOL umsetzen. Die DV Analytiker waren von dem Verfahren begeistert. Die Entwickler gaben sich damit zufrieden. Die Endanwender zeigten sich weniger begeistert und kündigten Widerstand an. Herr B. hatte die Rechnung ohne den Wirt gemacht. Die Fachabteilungen wurden nämlich gezwungen sich sehr früh und sehr detailliert auf eine fachliche Lösung festzulegen. Sie fühlten sich

dadurch gegängelt und der zentralen DV-Abteilung ausgeliefert. Dieser Konflikt zwischen den Fachabteilungen und der zentralen Entwicklungsabteilung sollte sich später verschärfen in dem Maße, wie die Fachabteilungen strengere Vorlagen bekamen.

5.4 Der omnipräsente Konflikt zwischen IT und Fachbereich

Der Konflikt zwischen der zentralen Datenverarbeitung und den Fachabteilungen war zu diesem Zeitpunkt keineswegs Bertelsmann-spezifisch. Er brach überall dort aus, wo es eine starke DV gab, die den Fachabteilungen Vorschriften machten. Ich wurde in dieser Zeit manchmal als Berater gerufen, um diesen Streit zu schlichten. Es schien eine unumgängliche Auseinandersetzung zu geben. Die zentrale DV-Entwicklung wollte die Anwender zu Disziplin zwingen, die sollten sich gewisse Arbeitsnormen unterwerfen, z.B. wie sie ihre Anforderungen beschreiben. Die Fachabteilungen wollten ihrerseits frei sein und so arbeiten, wie sie wollten, z.B. ihre Anforderungen nach ihren eigenen Vorstellungen formulieren. Der Konflikt war vorprogrammiert.

Bei der BMW herrschte der gleiche Konflikt. Es ging so weit, dass der fachliche Projektleiter mit dem DV Projektleiter nicht reden konnte. Obwohl sie im gleichen Zimmer saßen, haben sie nur schriftlich miteinander kommuniziert. Da es zu dieser Zeit noch keine Email gab, war diese Form der Kommunikation äußerst umständlich. Der Streit eskalierte sich sogar weiter bis zur Vorstandsebene so wie dies auch bei Bertelsmann der Fall war. So verhärtet waren die Fronten damals zwischen DV und Fachabteilung [Bend83].

Ein ähnlicher Fall gab es in einer großen Chemie-Firma in Nordrhein-Westphalen. Ein Vertreter der Fachabteilungen kam eines Tages zu mir nach Neubiberg und bat mich an einem Coup gegen den DV-Leiter teilzunehmen. Ich sollte alles zusammentragen, was er falsch gemacht hat und dies in seiner Anwesenheit auf einem Wochenendseminar in Bad Neuenahr vor dem Vorstand präsentieren. Ich stand gerade in Verhandlungen mit dieser Firma wegen dem Kauf von SoftDoc und SoftTest, so wollte ich nicht ablehnen. An einem Freitag im Herbst 1985 bin ich zu dem Konferenzhotel an dem Ahr gefahren und am nächsten Tag begann der Workshop. Diesmal stand ich auf der Seite der Fachabteilungen. Ich hielt einen Vortrag über die Vorteile der verteilten Verarbeitung und wie der Konzern darauf umsteigen sollte. Dann folgten die Fachabteilungsvertreter und zählten alle Projekte auf, bei denen die zentrale DV mit ihrem starren Phasenkonzept versagt hatte. Der DV-Leiter, der gar nicht ahnte, was auf ihm zukommt, war von dem geplanten Angriff völlig überrascht. Er wurde immer blasser. Seine Assistentin ging hinaus zum Telefonieren.

Es sah zunächst so aus, als ob der Coup gelingen würde, aber es kam anders. Am nächsten Tag kamen die Berater des IT-Leiters und haben die Sünden der Fachabteilungen aufgezählt. Sie argumentierten gegen eine Verteilung der DV-Ressourcen. Es würde den totalen Verlust der Kontrolle bedeuten und zu einem Chaos führen, wenn jede Fachabteilung seine eigene Rechenkapazität und sein eigenes Entwicklungspersonal hätte. Ich versuchte klar zu machen, dass die Tage des zentralen Mainframe Computers und der zentralen Entwicklung nach einem strengen Phasenkonzept gezählt waren. In Amerika gab es schon ein Trend zur verteilten ad hoc Entwicklung. Das hätte ich nicht anbringen sollen. Das Wort Amerika ist ohnehin in Deutschland ein Reizwort. Dies hat die Stimmung noch mehr angeheizt. Die Kräfte der Zentralisierung haben Oberwasser gewonnen, der einer Vorstand, der alles angezettelt hat, musste sich entschuldigen, der Vorstand, der hinter dem DV-Leiter stand, wurde immer wortlauter. Es sollte sich nichts ändern. Es blieb bei der Zentralisierung. Ich habe mich davongeschlichen und bekam nie wieder einen Auftrag bei diesem Konzern. Die Sache mit den Werkzeugen war erledigt. So teilte es mir der DV-Leiter höhnisch mit.

Es war für mich eine Lehre in der deutschen Unternehmenskultur. Dort herrscht ein erbarmungsloser Machtkampf unter Alpha-Tieren. Es geht gar nicht um die Sache, sondern darum wer das Sagen hat. Wer Produkte und Dienstleistungen verkaufen will, darf seine potentiellen Kunden nicht beraten, erst recht nicht in solchen sensitiven Angelegenheiten. Jeder Bereichsleiter benimmt sich wie ein Kurfürst und versucht sich gegen den Kaiser zu behaupten. Wenn ein Kurfürst in Ungnade fällt wird seine gesamte Gefolgschaft abgestraft. Wenn der Kaiser selbst gekippt wird werden alle die ihm geholfen haben an der Macht zu bleiben abgesetzt. Die Auseinandersetzung zwischen der zentralen DV-Abteilung und den Fachabteilungen schwellte überall. Sie schien unumgänglich zu sein. Der eine Seite will mehr Ordnung haben, der andere mehr Freiheit [Oest03]. Als externer Berater ist es besser, sich herauszuhalten. Bald sollte ich selber zum Opfer dieses Konfliktes werden.

Das Jahr 1985 fing für mich gut an. Ich hatte vier Großkunden für alle Softorg Werkzeuge und 18 Kunden mit einzelnen Tools. Wir hatten Entwicklungsprojekte mit den Werkzeugen bei Bertelsmann, bei der Württ-Feuer, bei der BMW und bei der Deutschen Bank. Dort waren überall SES Mitarbeiter im Einsatz, deutsche wie auch ungarische. Unsere öffentlichen Seminare waren voll ausgebucht und es gab kaum eine Ausgabe der Computerwoche in der ich nicht erwähnt wurde. Damals war ich ein bevorzugte Interview Partner. Im Frühjahr 1985 haben wir ein Benutzertreffen in Budapest mit Vorträgen und Erfahrungsaustausch sowie Zigeunermusik, Palinka und Csardas veranstaltet – das volle Programm. Vertreter der führenden deutschen Fach-

zeitschriften waren eingeladen. Diesmal war die ungarische Presse auch dabei und hat unsere Kooperation als Vorbild für die neue Ost-West Beziehungen nach dem Helsinki Abkommen hochgelobt. Vorbei war die Zeit als ich von der Polizei als Staatsfeind angeprangert wurde. In Ungarn wurde ich geehrt und zum Held der Arbeit gehoben. Es sah auch so aus, als ob ich mein Ziel an der Spitze der deutschen IT Welt zu kommen trotz des Verlusts der Rüstungskunden zum Greifen nahe war, - doch bald folgten die nächsten Rückschläge [Schnei85].

5.5 Mein unfreiwilliger Abgang von BMW

Es war nie meine Absicht gewesen, das Personalvermittlungsgeschäft – Body Leasing genannt – zu betreiben. Ich hielt es für unehrlich und unehrenhaft – eine Art Sklavenhandel. Ich war eher der Meinung, der Kunde sollte die Aufgaben spezifizieren und zu einem festen Preis vergeben. Das externe Softwarehaus soll in der Verantwortung stehen die erwünschten Ergebnisse zu liefern. Leider habe ich damit die Anwender überschätzt. Sie waren meistens dazu gar nicht in der Lage und auch wenn sie es wären, wäre der Aufwand für sie zu hoch gewesen. So blieb ihnen nichts Anderes übrig, als Personen befristet anzuheuern und unter ihrer Leitung vor Ort arbeiten zu lassen. Auch ich sah mich gezwungen dieses Spiel mitzumachen, denn von der Benutzerseite aus war es nicht möglich die zu leistende Arbeit im Voraus zu definieren und von der ungarischen Außenhandelsgesellschaft stand ich unter Druck ungarische Entwickler zu vermitteln. Das war der Preis für ihre Unterstützung bei der Weiterentwicklung der Werkzeuge.

Die Firma bei der wir die meisten solcher Leiharbeiter hatten war BMW. In dem APIS Projekt waren drei Ungarn beschäftigt. Hinzu kam die Betreuung des SoftCon Tools durch das SoftCon Team und die Betreuung des SoftDoc Tools durch den zuständigen Gruppenleiter – Dr. Merey - vom SzAMOK Institut. BMW wie andere Großanwender pflegte externe Mitarbeiter für einmalige Projekte einzusetzen. Man glaubte, wenn das Projekt zu Ende wäre könnten sie die externen Kräfte wieder abbauen und damit ihre Festkosten niedrig halten. Es war seitens der Gewerkschaft sogar verboten Personal nur befristet anzustellen. So kam es das der Projektleiter von dem APIS Projekt eines Tages auf mich zu kam und darum bat eine junge Frau an zustellen und als Projektsekretärin ihm anzubieten. Sie war die Freundin eines seiner externen Entwickler. Dieses Gefallen haben wir ihm gewährt und Fräulein Z. wurde als Projektsekretärin eingesetzt. Ihre Hauptaufgabe bestand darin, die Time Sheets zu führen, die Arbeitsstunden zu buchen und den Personalaufwand zu belegen. Ich habe mich in dem Projekt um den Einsatz von SoftCon gekümmert und war mit deren Entwicklung voll beschäftigt. Gleichzeitig musste ich die Projekte bei Bertelsmann

und der Deutschen Bank betreuen. Ich hatte keine Zeit mich um Fräulein Z. zu kümmern. Das sollte sich bald als schwerwiegende Fehler erweisen.

Später ist die BMW darauf gekommen, dass die wöchentlichen Arbeitsstunden der externen Mitarbeiter viel zu hoch waren. Laut den Aufwandsberichten haben sie 60 bis 70 Stunden die Woche gearbeitet. Die BMW Rechtsabteilung vermutete Betrug und hat die Sache näher untersucht. Sie sind schnell darauf gestoßen, dass Fräulein Z. die Stundenberichte zu Gunsten der Externen fälscht. Es wurde ihr sofort gekündigt und es kam zu einem Prozess – jetzt war meine Firma dran. Als „Arbeitgeber" waren wir für sie verantwortlich. Wir wurden zu einer Geldstrafe verurteilt und ich durfte BMW nicht mehr betreten. Die Werkzeuge konnten bleiben aber ich musste weg. Meine Zeit bei BMW war vorbei.

Das war schmerzhaft, da wir gerade in BMW dabei waren, unsere Werkzeuge zu integrieren. SoftSpec war bereits als allgemeines Tool für die Anforderungsanalyse akzeptiert. SoftCon war im Begriff als technisches Dokumentationswerkzeug angenommen zu werden – sogar die Standardsoftware in APIS wurde damit dokumentiert. SoftGen sollte demnächst für die Generierung von PL/I Programme eingesetzt werden. SoftDoc war schon dort länger im Betrieb. Sämtliche PL/I Programme wurden damit geprüft, ehe sie für die Produktion freigegeben wurden. Das Tool war Bestandteil des Release Prozesses-. SoftTest wurde schon probeweise benutzt, um die PL/I Programme zu testen. Ich stand bei BMW vor dem großen Durchbruch – dann kam die Affäre mit Fräulein Z.

Das war mein dritter großer Rückschlag in München – erst Siemens, dann MBB und jetzt BMW. SoftSpec, SoftCon und SoftDoc sind als Werkzeuge geblieben. Ich und die anderen SES Mitarbeiter müssten abziehen, bis auf den Dr. Merey. Er hat politisches Asyl beantragt und wurde von der BMW angestellt. Von der ungarischen Seite her dürfte ich mit ihm nicht weiter zu tun haben. Zwei andere ungarische Mitarbeiter haben sich einer anderen Personalvermittlungsfirma angeschlossen um dort weiterarbeiten zu können.

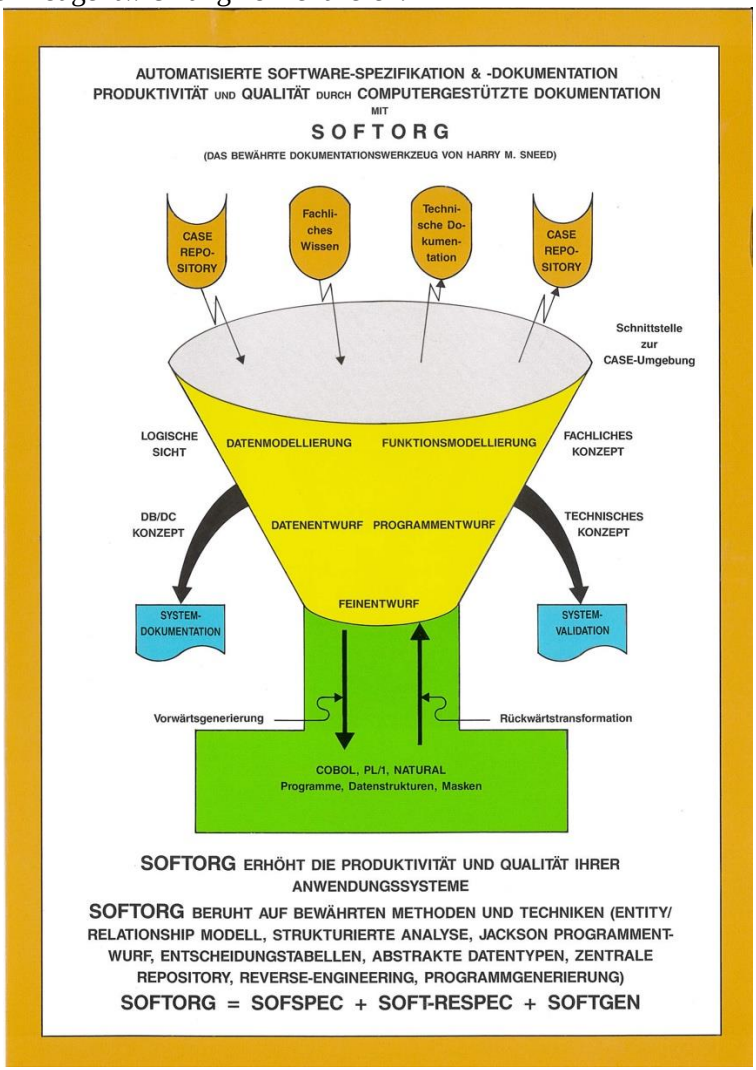
Der Verlust von BMW als Stammkunde war nicht leicht zu verkraften. Auch persönlich war es für mich nachteilig, da es für mich bequemer gewesen war in München zu arbeiten. Unser Büro war in Neubiberg und ich wohnte bei Holzkirchen. Jetzt musste ich nach Gütersloh reisen um die Werkzeugentwicklung von dort aus voranzutreiben. Es hätte aber auch schlimmer ausfallen können. Wie der Anwalt von BMW mir mitteilte, haben sie auf mich Rücksicht genommen. Sonst wäre meine Firma viel härter bestraft worden.

Der Asylantrag von Dr. Merey bei BMW hatte leider weitere Konsequenzen für mich. Er war sowas wie ein Präzedenzfall für die anderen SzAMOK Mitarbeiter. Kurz darauf haben drei von Ihnen um politisches Asyl gebeten, dieses Mal bei Bertelsmann in Gütersloh. Darunter war Frau Majoros, meine langjährige SoftTest Projektleiterin. Sie war vom Anfang an dabei gewesen und war für mich eine große Stütze im Testbereich. Sie hatte an allen Testwerkzeugen seit dem Prüfstand mitgewirkt und war die Einzige die sich mit SoftTest wirklich auskannte. Ich bin mit ihr immer gut ausgekommen und schätze sie als Mensch und Fachkraft. Leider konnte ich für sie unter den damaligen Bedingungen in Ungarn nicht mehr tun. Umso mehr hat es mir wehgetan, sie zu verlieren. Sie und ihr Mann wurden von Bertelsmann übernommen. Ich dürfte mit ihr nicht mehr zusammenarbeiten. So bekam ich ein neues Problem, womit ich nicht gerechnet hatte, den Verlust des Schlüsselpersonals und deren Knowhow. Frau Erdös, die für SoftInt zuständig war, musste jetzt auch noch SoftTest übernehmen und Herr Jandrasics übernahm SoftDoc von Dr. Merey. Er war auch für die SoftGen Entwicklung zuständig.

5.6 Die Bundesbahn setzt Softorg ein

Das Inkasso-Projekt bei der Württ-Feuer lief weiter, allerdings nicht mit der gleichen Intensität wie zuvor. Frau Nyary und unsere beiden SES Mitarbeiter haben dort die Entwicklungsprojekte unterstützt. Da Emissionsprojekt bei der deutschen Bank war in der Schlussphase. Der Test wurde von unserer neuen Mitarbeit aus der DDR betreut. Ich konnte mich dort in Eschborn zurückziehen. Im Herbst 1985 hat die Bundesbahn entschieden, die ganze Softorg Werkzeugfamilie auszuprobieren. Das war für mich ein großer Erfolg. Die ersten Werkzeuge – SoftMan und SofSpec – wurden in Frankfurt installiert und ich begann sie sofort zu schulen. Ein junger Hauptmann von der Bundeswehr hatte schon vorher mit mir Kontakt aufgenommen wegen einer Beschäftigung im Frankfurter Raum – er sollte bald nach 12 Jahren Wehrdienst entlassen werden. Er hatte unsere Firma über die Newsletter kennen gelernt und war von der Idee des SoftOrg Entwicklungszyklus fasziniert. Ursprünglich hatte ich ihn für das Deutsche Bank Projekt vorgesehen, aber jetzt sah ich die Möglichkeit ihn bei der Bahn einzusetzen. Die Bahn DV-Abteilung in Frankfurt hatte begonnen die ersten Systeme mit SoftSpec zu modellieren und brauchte eine Betreuung vor Ort. Zusammen mit den Bahnanalytikern haben wir die Anforderungen für die Frachtverkehrssteuerung ermittelt und auf dem Mainframe unter TSO/ISPF dokumentiert. Jeder Frachtobjekttyp und jeder Transportvorgang müsste erfasst und beschrieben werden. SofSpec erwies sich als das geeignete Tool dies zu bewerkstelligen. Mein neuer Mitarbeiter hat sich bald bewährt. Er passte sehr gut zu dieser Aufgabe und gewann in

kürzester Zeit das Vertrauen der Bahnmitarbeiter. Es gibt nichts Besseres als eine militärische Ausbildung. Ich konnte ihm das Bahnprojekt überlassen und mich auf die Werkzeugentwicklung konzentrieren.



Das Jahr 1986 stand in Zeichen der SoftOrg Weiterentwicklung und der Anwenderbetreuung. Die Werkzeugentwicklung teilte sich auf zwischen Budapest und Gütersloh. Der Code wurde in Budapest geschrieben, editiert und kompiliert. Dann wurde er nach Gütersloh zum Testen gebracht. Die Teams von SzKI und SzAMOK haben sich in Gütersloh alle zwei Wochen abgewechselt. Wir hatten dort ein großes Zim-

mer, freier Zugang zum Rechner und jede Menge Testprojekte. Wir waren immer noch damit beschäftigt Programme, Datenbanken und Bildschirmdialoge in verschiedenen Sprachen vollständig zu generieren. Dazu musste das Tool SoftCon alle erforderlichen Entwurfsdaten beinhalten. Dies setzte eine enge Zusammenarbeit zwischen den Teams der beiden Institute voraus und ich musste mich besonders um diese Schnittstelle kümmern. Wegen der Rivalität der Institute war dies keineswegs einfach. Jedenfalls dank der Unterstützung von Bertelsmann ist es uns gelungen bis Ende des Jahres die ersten Programme zu generieren. Gleichzeit wurde das Tool SoftMan für die Projektplanung und -steuerung endlich fertiggestellt. Die Bahn wollte sie benutzen um ihre viele Projekte zu planen und deren Kosten nach COCOMO sowie nach Function-Point zu schätzen. Das ist ebenfalls bis Ende 1986 gelungen.

Eins dieser Projekte war das Abrechnungssystem. Die Bahn hatte begonnen das System in Frankfurt zu spezifizieren. Sie hatte allerdings nicht vor, es selbst zu entwickeln. Sie sollte lediglich die Spezifikation erstellen. Die Entwicklung war der Firma Krupp-Atlas, dem Auftragspartner der Bahn zugedacht. Hier gab es einen Bruch in den Projektablauf zwischen dem Auftraggeber – die Bahn in Frankfurt - und dem Auftragnehmer – Krupp-Atlas in Essen. Der Bund hatte damals angeordnet, dass Softwareentwicklungsprojekte ab einer gewissen Größe nach außen zu vergeben sind. Dadurch gewann ich zwar einen weiteren Kunden aber ich erbte damit ein weiteres Problem, nämlich die Abstimmung zwischen Auftraggeber und Auftragnehmer. Es stellte sich die Frage – wo hört die Anforderungsspezifikation auf und wo beginnt die Entwicklung des Zielsystems. In dem SoftOrg Lebenszyklus Modell lag diese Schnittstelle zwischen SoftSpec und SoftCon, aber die Frage lieb offen, wie weit man mit SoftSpec gehen sollte. Die Antwort blieb aus. Wir haben jedenfalls SoftCon bei Krupp in Essen installiert damit die Entwickler dort mit der Implementierung auf dem IBM-Mainframe beginnen konnten.

5.7 Programmierung bei Krupp-Atlas

Die Implementierung des Abrechnungssystems der Bahn begann schon im Sommer 1986. Krupp-Atlas hat dafür ein neues Entwicklungszentrum in Troisdorf bei Bonn eingerichtet. Ich habe mich entschlossen, mich dem Entwicklungsteam anzuschließen und als COBOL Entwickler dort zu arbeiten. Das war sozusagen mein Sommerurlaub 1986. Tatsächlich habe ich dort jede Woche von Anfang Juli bis Mitte September gearbeitet. Es war meine erste Programmiertätigkeit seit der Entwicklung des Prüfstandes bei Siemens und es dauerte einige Tage, bis ich wieder drin war. Damals hatte ich mit der Sprache SPL gearbeitet, mit COBOL hatte ich das letzte Mal bei der Göttinger Stadtverwaltung im Jahre 1974 programmiert.

Mir kam es darauf an zu erfahren, wie es ist, mit einer SoftSpec Anwendungsspezifikation und einen SoftCon Systementwurf zu arbeiten. Ich wollte in der Praxis erfahren wie ich das Fachmodell in einen DV-technisches Modell am besten überführen konnte, d.h. die beiden Werkzeuge – SoftSpec und SoftCon aufeinander abstimmen. Dabei lernte ich was anders, wie schwierig es ist anhand fremder Vorhaben ohne Anwesenheit der Sachgebietsexperten zu arbeiten. Die SoftSpec Vorgaben waren wirklich sehr detailliert mit Baumdiagrammen, Datenflussdiagrammen und Entscheidungsbäume spezifiziert. Es gab auch eine Menge Text dazu. Dennoch fehlte immer noch die eine oder andere Detailinformation, um die programmtechnische Lösung fertigzustellen. Hier kam die Erkenntnis, dass solange als der Entwickler selbst kein Sachgebietsexperte ist, muss die Spezifikation auf der gleichen Detaillierungsebene sein, wie der fertige Code selbst. D.h. jede IF Bedingung muss vorgegeben werden. Ist dies der Fall, dann ist Programmierung eine reine Übersetzungsarbeit. Der Programmierer übersetzt die Sprache der Spezifikation in die jeweilige Programmiersprache. Dies war hier aber nicht der Fall. Es gab eine Menge offener Frage zu den Rechnungsbedingungen. Die Leute, die sie hätten beantworten können, waren nicht verfügbar. Die Experten für das Rechnungswesen saßen in Frankfurt. Es war sehr frustrierend, nicht nur für mich, sondern auch für die anderen Krupp Entwickler.

Als Notlösung wurde entschieden, einen Abrechnungsexperten von der Bahn in Frankfurt nach Troisdorf zu senden, jedoch nur für 3 Tage die Woche. Die Entwickler standen Schlange vor seinem Zimmer. Auch ich musste mich einreihen, um eine Detailfrage zu klären. Meistens hat eine Stunde gereicht um die Frage zu klären, aber dafür musste man mehrere Stunden oder gar Tage warten. Mir kamen Zweifel über den gesamten Top-Down Entwicklungsprozess auf. Es sei vielleicht doch nicht möglich, das Fachliche von dem Technischen zu trennen. Sie sind auf der untersten Ebene doch noch miteinander eng verwoben. Diese Erkenntnis habe ich zwei Jahre später in einem Paper für die internationale Maintenance Konferenz zusammengefasst [Sned89].

Ich kam mit einem unguten Gefühl weg von Troisdorf. Diese zwei Monate in der Projektpraxis hat mir gezeigt, wie wichtig das Fachwissen ist. Es muss bei dem letzten Entwickler vorhanden sein. Auch eine ausführliche Spezifikation ist kein Ersatz für weitgehendes Fachwissen, wenn es darum geht, einen fachlichen Vorgang vollständig zu kodieren. Es ist erstaunlich, wie viele Anwender dies nie begriffen haben und das namhafte Wissenschaftler diese Trennung vom fachlichen Entwurf und technischen Implementierung immer noch propagieren. Das Entwicklungsmodell der Bundesverwaltung war darauf aufgebaut [Bröhl93]. Mein ungutes Gefühl nach mei-

ner Erfahrung in Troisdorf kam auch daher, dass ich begonnen habe an meiner eigenen Lehre zu zweifeln. Die Rechnung für diese Irrlehre sollte bald, im nächsten Jahr folgen.

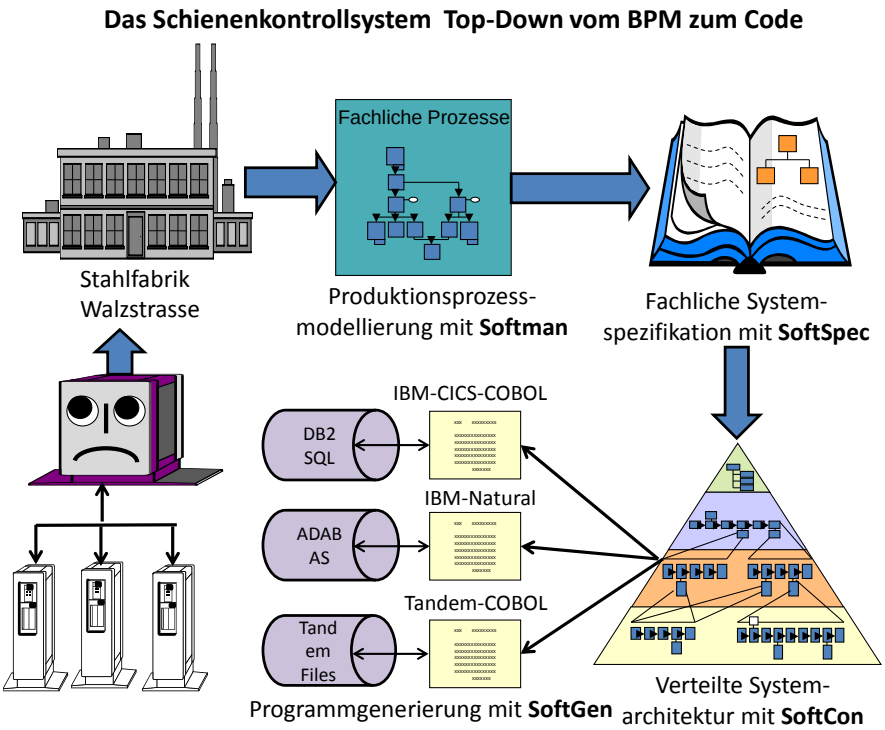
5.8 Der große Test bei Thyssen Stahl

Bereits im Sommer 1986 hatten Vertreter der Thyssen-Stahl uns beim Bertelsmann besucht, um sich die Softorg Produkte anzuschauen. Sie waren im Begriff ein neues Großprojekt für die Herstellung von Eisenbahnschien zu lancieren und suchten geeignete Werkzeuge, Über das Newsletter sind sie auf uns gestoßen. Ich habe sie in Gütersloh empfangen und gezeigt, wie die Tools in der Praxis funktionieren. Inzwischen waren SoftMan und SoftInt ebenfalls dort im Einsatz, so dass ich ihnen den gesamten Lebenszyklus von der Projektdefinition bis hin zur Produktintegration zeigen konnte. Es hat sie scheinbar sehr beeindruckt denn in September haben sie sämtliche Werkzeuge in Auftrag gegeben. Ich sollte nach Duisburg-Hamborn kommen und das neue Projekt beraten. Um dieselbe Zeit kam einen weiteren jungen Bundeswehroffizier auf mich zu – der Herr Kroeger - und bat mich um eine Anstellung. Auch er stand kurz vor der Entlassung und suchte eine Stelle in der Zivilgesellschaft. Da er in Köln wohnte, habe ich mich entschieden, ihn bei Thyssen als Projektmanagementberater einzusetzen.

Im Oktober 1986 ging es mit dem ersten Projekt in Duisburg-Hamborn los – das Schienenkontrollsystem. Ich bekam ein eigenes Zimmer im oberen Stockwerk des Verwaltungsgebäudes umgeben von alten Plantannen. Herr K saß unten mit dem Projektleiter, einem sympathischen älteren Ingenieur zusammen. Die Ungarn kamen, um die Werkzeuge zu installieren. Alle fünf Teams sind nach einander angereist und haben ihre Werkzeuge installiert und in der neuen Umgebung getestet. Die Tools waren dort, aber es gab nicht genüg trainiertes Personal um sie zu bedienen. Ich habe bald erkannt, dass wir mehr Mitarbeiter vor Ort benötigten, um das Projekt voranzubringen. Thyssen hatte kaum Leute für dieses Projekt und diese brauchten Unterstützung bei der Bedienung der Tools. Die Ungarn waren alle mit der Entwicklung der Tools beschäftigt.

Ich hatte in der lokalen Zeitung gelesen, dass das Arbeitsamt in Duisburg viele arbeitslose Lehrer zu vermitteln hatte, denn zu dieser Zeit herrsche in dem Ruhrgebiet eine hohe Arbeitslosigkeit. Anderseits war Thyssen wie auch BMW kaum geneigt neue Mitarbeiter fest anzustellen. Sie waren dabei ihre Stammmannschaft abzubauen. So konnte Günter Wallraff als getarnter Türke in die Produktionshallen kommen um Material für sein Buch „Ganz Unten“ zu sammeln. Ich habe mich entschieden, zwei

junge Lehrer für das Projekt anzustellen – ein Mann und eine Frau, beide blond und urdeutsch. Sie sollten lernen mit den Werkzeugen umzugehen um die Thyssen Fachanalytiker bei der Spezifikation der vielen Datenobjekte und Abläufe mitzuhelfen. Sie sollten den festangestellten Analytikern technisch gerade einen Schritt voraus. Ich habe viel Zeit damit verbracht, sie auszubilden. Sie nahmen an allen meiner Seminare teil und wurden bald, wenn nicht Informatikexperten, denn wenigstens Softorg Experten. Der Hauptmann Kroeger hatte, wie auch Hauptmann Schäffer, an der Bundeswehrhochschule Informatik studiert. Mit seiner Erfahrung aus der Bundeswehr war er sehr wohl in der Lage die neuen Mitarbeiter zu führen. Jetzt waren wir zu viert bei Thyssen ohne die Ungarn die gelegentlich kamen um die Werkzeuge anzupassen. Getestet wurde bei Thyssen nicht. Dafür hatten wir noch den Rechner in Gütersloh.



Ich hatte mit der IT-Leitung einen Festpreis für das Gesamtprojekt vereinbart, aus dem monatliche Zahlungen erfolgten. Es hat gerade gereicht um die Personal- und Reisekosten zu decken. Ich bin dazu übergegangen, mit der Bahn nach Duisburg zu fahren um Kosten zu sparen – Sonntagnacht mit dem Schlafwagen und freitags Nachmittag zurück mit dem Intercity. Da ich in Duisburg nicht wohnen wollte, nahm

ich ein Zimmer in Dinslaken und bin jeden Tag mit dem Rad durch Marxloh und den gelben Dunst der Fabrikabgase dorthin gefahren. Die Fahrt dauerte eine Stunde und das gab mir Zeit etwas nachzudenken. Das hatte ich auch nötig, denn dieser Herbst 1986 hat von mir viel abverlangt. Neben dem großen Projekt bei Thyssen gab es noch die Projekte bei der Bahn und bei Bertelsmann. Hinzu kamen die Ausbildungsveranstaltungen, die ich immer noch jeden Monat ausführte und die Flüge nach Budapest, wo ich weiterhin die Werkzeugentwicklung steuern musste. Ich war an der Grenze meiner physischen und psychischen Kapazität angelangt.

In den ersten Monaten in Duisburg ging alles planmäßig. Wir haben das Schienenkontrollsystem mit SoftMan geplant und mit Anweisungen, Function Points und Data-Points geschätzt. Die Erfahrungen meines ehemaligen Werkstudenten Thomas Noth sind in das Tool SoftMan eingegangen [Noth84]. Das Ganze sollte nicht länger als ein Jahr dauern und nicht mehr als DM 400.000 kosten. Herr Kroeger kam mit dem Projektleiter gut zurecht und sie konnten die Projektabläufe detailliert planen. Mit Softman konnte man auf Basis des Datenmodells den Aufwand schätzen und auf Basis des Funktionsmodells die Projektaufgaben planen. Ich habe mit Hilfe der beiden Praktikanten die Systemspezifikation in allen Details ausgearbeitet. Es dauerte nicht lange bis wir glaubten uns in der Schienenproduktion auszukennen. Zuerst haben wir den fachlichen Rahmen abgesteckt und dann mit Hilfe der Thyssen Verfahrensingenieure den Inhalt hinzugefügt. Bis Dezember 1986 waren alle Werkzeuge bis auf SoftGen eingesetzt. Da wir nicht nur den Host sondern auch die verteilten Tandem Rechner als Zielrechner hatten, mussten wir SoftGen noch erheblich anpassen. Mitte Dezember erfolgte ein großer Design Review unter Beteiligung aller zukünftigen Anwender statt. Sie waren zu dem Zeitpunkt alle mit den vorläufigen Dokumenten einverstanden. Das IT-Management war mit dem Stand des Projektes zufrieden und am letzten Arbeitstag vor die Weihnachtsferien wurde groß gefeiert. Das Projekt war angeblich auf einem guten Weg.

Beim Übergang ins Jahr 1987 schien alles noch in Ordnung zu sein. Wir hatten BMW und Württ-Feuer als Projektkunden verloren aber sie haben immer noch die Werkzeuge gehabt und bezahlten dafür. Das Projekt bei der deutschen Bank war erfolgreich abgeschlossen und die Bank hat die Tools SoftSpec und SoftTest behalten. Bei Bertelsmann liefen die Projekte weiter und bei der Bahn liefen sie gerade an. Die größte Herausforderung war das Projekt bei Thyssen-Stahl. Es war entscheidend für die Zukunft der SoftOrg Werkzeuge. Anfang 1987 kam ein neuer Kunde dazu – das Bremer Lagerhaus. Dieser Kunde begann gleich mit SoftMan ein Projekt zu planen und mit SoftSpec das Zielsystem zu spezifizieren. Unsere beiden ehemaligen Bundeswehroffiziere hatten sich inzwischen gut eingearbeitet und waren voll im Einsatz.

Der Teamleiter von SoftCon – Herr Fehervari – hatte die Betreuung von Krupp übernommen. Die Teamleiterin von SoftSpec – Frau Nyary – hat die Continental Reifen in Hannover betreut. Sie musste das System auch noch auf Französisch für die Continental Tochter in Belgien umstellen. Ich selber hatte die Hände voll zu tun mit dem Thyssen Projekt und jetzt auch noch mit dem Bremer Lagerhaus.

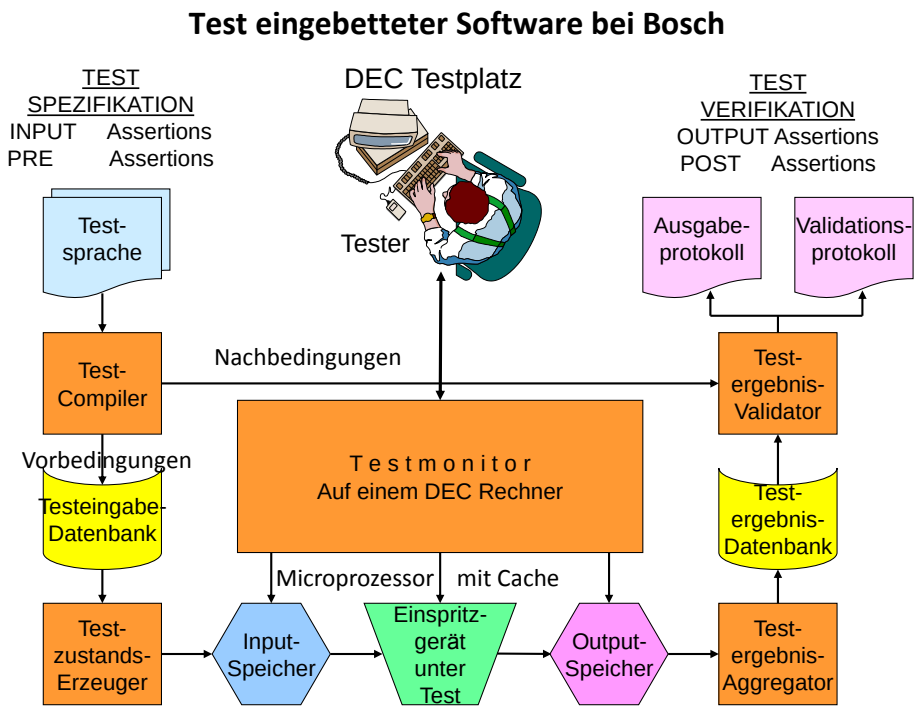
5.9 Der Test eingebetteter Realtime Software

Mitten drin in dieser Hektik kam eine weitere Belastung hinzu. Die SES bekam zu meiner größten Überraschung ein EU-Forschungsprojekt für den Test eingebetteter Systeme. Wir hatten schon früher den Antrag gestellt zusammen mit Prof. Michael Hennell von der Universität Liverpool. Ich hatte aber nie damit gerechnet, dass wir das Projekt tatsächlich bekommen. Im Januar 1987 haben wir mit der Arbeit begonnen. Jetzt musste ich auch noch nach England fliegen, um dort mit den anderen Projektpartnern abzustimmen. Da Herr Orgas bei der BMW nicht mehr arbeiten durfte und bei der Württ-Feuer auch fertig war, habe ich ihn für das EU-Testprojekt vorgesehen. Zur seiner Verstärkung habe ich einen Physiker und erfahrener Qualitätsmanager von dem Reaktorzentrum in Garching als freier Mitarbeiter angeheuert. Die beiden sollten zusammen mit mir die neuen embedded Testsysteme entwickeln.

Neben der Entwicklung der neuen Tools, musste ich diverse Papers liefern um die Förderungsgeld zu beanspruchen. Die Papers konnte ich abends in Dinslaken und unterwegs in der Bahn schreiben. Die Tools versuchten wir aus den bestehenden Tools abzuleiten sowie Prof. Hennell es auch machte. Die Reisen nach England zu den Projektpartnern waren eine angenehme Abwechslung aber die Zeit fehlte mir für die Projekte in Deutschland und die Toolentwicklung in Ungarn, die jetzt in eine kritische Phase war. Die ersten Probleme mit den Instituten sind dort aufgetaucht. Die staatlichen Subventionen wurden gekürzt und die Institute bestanden auf mehr Einnahmen von meiner Seite. Das Embargo auf alle Hightech Produkte für die COMECON Länder als Folge der russischen Intervention in Afghanistan zeigte seine Wirkung. Der Siemens Rechner bei der SZKI war veraltet und der IBM Rechner bei der SZAMOK war gar nicht mehr im Betrieb. An seiner Stelle bekam SZAMOK einen russischen R55 Rechner aus der ESER-Serie, die nie richtig funktioniert hat. Es war klar zu sehen, dass der Osten technologisch abgehängt war. Die Ungarn bekamen keine Ersatzteile für deren alten Rechner und auch keine neue Rechner. Ich war jetzt völlig auf Bertelsmann angewiesen, die uns ihren Testrechner zur Verfügung stellte. Das hat zu Folge, dass die ungarischen Entwickler immer mehr Zeit in Gütersloh verbringen mussten. Da schlugen die Reisekosten zu Buche. Statt für Rechenkapazität in Budapest musste ich für den Aufenthalt der Ungarn in Gütersloh bezahlen. Der

Testbetrieb in Gütersloh war für die Erhaltung und Weiterentwicklung der Werkzeuge unentbehrlich und die Werkzeuge waren die Voraussetzungen für unsere Projekte. Ohne die Tools hätten wir kein einziges Projekt gehabt.

Ich war in dieser Zeit ständig unterwegs zwischen Budapest und Liverpool sowie zwischen Gütersloh, Bremen und Duisburg. Zuhause in München war ich nur kurz dazwischen. Die Büroführung habe ich voll in den Händen der Frau Molnar überlassen. Ich selbst hatte keine Zeit mehr dafür. Ich konnte gerade noch das Newsletter verfassen und die Seminare vorbereiten, sonst nichts. Die Buchhaltung, das Rechnungswesen und die Seminarorganisation hat alles Frau Molnar gemacht.



Das Testforschungsprojekt – TRUST – war glücklicherweise von der Softorg-Entwicklung abgekoppelt. Dafür hatten wir einen eigenen Rechner mit Mittel der EU beschafft – einen SUN Rechner mit einem UNIX Betriebssystem. Dieses Projekt lief in einer völlig anderen Welt ab. In dieser Welt kamen andere Kunden hinzu – Firmen aus dem Embedded System Bereich. Von der Firma Bosch in Ludwigsburg bekamen wir ein Projekt, um eine Motronics Testsoftware für den Test von Einspritzgeräte zu

entwickeln. Die Hardware dazu hatten sie schon – ein Digital Equipment Rechner auf dem Mikroprozessoren in einem Testrahmen ausgeführt werden konnten. Wir sollten die Software dazu liefern. Mit den beiden Mitarbeitern von dem TRUST Projekt habe ich den Eingabegenerator und den Ausgabe-Validator geliefert. Hier handelte es sich um einen reinen Black-Box Test. Es war nicht möglich an dem Code nur das Geringste zu ändern bzw. zu instrumentieren. Wir müssten die Testüberdeckung anhand der von dem Einspritzgerät erzeugten Signale messen. Wir konnten die Ergebnisse auch nicht während des Tests kontrollieren, dafür ging alles viel zu schnell. Die Durchlaufzeit einer Signalumsetzung dauerte nur 37 Mikrosekunden. Wir mussten die Ergebnisse in einer Oracle relationalen Datenbank sammeln, um nach dem Test gegen die spezifizierten Sollwerte zu prüfen. Das Projekt ähnelte dem früheren Kienzle Projekt. Für die Spezifikation der Sollwerte haben wir die EPOS Spezifikationssprache von Professor Lauber an der TU-Stuttgart erweitert. Bosch konnte mit dieser auf dem DEC Rechner laufender Simulation fortwährend testen und jede kleine Abweichung von dem Soll registrieren [TRUST89].

Dieser Einspritzgerät-Test passte sehr gut zu dem EU-Forschungstestprojekt. Dort ging es um den Test der Flugsteuerungssoftware von dem neuen Airbus. Dafür wurde die BacktoBack Testtechnik eingesetzt. Zwei getrennt voneinander entwickelten Programme – ein in ADA und ein in Pascal – liefen neben einander und wurden ständig mit einander verglichen. Wir konnten unsere Erfahrung vom Bosch Projekt dort hineinbringen. Aus diesen beiden embedded Testprojekten habe ich viel gelernt. Die gleiche BacktoBack Testtechnik konnte ich später für ein eigenes Testwerkzeug auf dem PC verwenden. Das Testen der embedded Gerätesoftware markierte den Höhepunkt meiner Karriere als Tester. Nachher habe ich begonnen, mich mehr mit der Wartung und Sanierung bestehender Systeme zu befassen.

Von Thyssen hatte ich eine Reihe alter COBOL Programme bekommen, die sie in einer anderen Umgebung wiederverwenden wollten. Dazu mussten sie aber erst überarbeitet werden Es wäre zu aufwendig gewesen, die Programme manuell zu überarbeiten. So habe ich dafür ein neues Tool konzipiert, welches das Team von Herrn Jandrasics für mich implementierte. Dem neuen Tool habe ich den Namen „SoftRecon“ gegeben weil es um die Rekonstruktion bestehender Software ging. Das Tool hat die Ablauflogik restrukturiert, die GOTO Anweisungen weitgehend entfernt und die großen Programme in mehrere kleine Module zerlegt. Das Ergebnis waren kleinere, strukturierte Module an der Stelle der großen, monolithischen Programme. Es war meine erste Erfahrung mit der automatisierten Restrukturierung von COBOL Programmen [Sned87]. Das Tool SoftRecon sollte das Erste in einer Reihe neuen Tools für die automatisierte Reengineering alter Mainframe-Programme sein mit

SoftRespec für die Wiederherstellung einer fachlichen Spezifikation und SoftRetest für den Regressionstest der sanierten Programmen. Diese Tools habe später ich in das dritte Esprit Forschungsprojekt namens DOCKET eingebracht, das 1990 beantragt wurde. Ich hatte inzwischen europaweit einen Ruf als Fachmann für Software Reengineering und wurde deshalb eingeladen an dem Projekt teilzunehmen [DOCK91].

Im Herbst 1987 bin ich nach zehn Jahren wieder nach Amerika zurückgekehrt um auf der internationalen IEEE Maintenance Konferenz in Austin, Texas vorzutragen. Mein Vortrag ist gut angekommen und ich bin dadurch in den Kreis der amerikanischen Maintenance Experten aufgenommen. Ich wurde sogar Ehrenmitglied der amerikanischen Maintenance Association und erhielt das passende Maintenance Hemd mit der Inschrift „I am proud to be a Maintenance Programmer". Dies markierte den Anfang einer neuen Karriere als Software ReEngineer. Dies war auch gut so, denn mit meiner Karriere als Software Engineer war ich in stürmische Wasser geraten.

5.10 Das unglückliche Ende des Thyssen Projektes

Das Projekt bei Thyssen – die Steuerung der Walzstraße für die Schienenproduktion – kam im Frühjahr 1987 in eine kritische Phase. Die ersten Subsysteme waren inzwischen fertig spezifiziert und sollten in die Entwurfsphase übergehen. Dass hieß, SoftCon war jetzt gefragt. Hier hatten wir zwei Aufgaben zu meistern. Erstens musste das SoftSpec Datenmodell in eine ADABAS Datenbankschema für den Mainframe sowie in eine netzartige Datenbankstruktur für den Tandem Rechner umgesetzt werden. Parallel dazu musste das SoftSpec Funktionsmodell sowohl in einen CICS Transaktionsrahmen für den Mainframe als auch in einen Tandem Dialograhmen für die verteilten Rechner in der Fabrik verwandelt werden. Der Walzprozess wurde nämlich vom Mainframe aus geplant und überwacht aber von dem Tandem Rechner in der Produktionshalle gesteuert. Mit SoftCon wurde die Software für beide Rechnertypen auf dem Mainframe entworfen. Wir mussten viele Erweiterungen an SoftCon vornehmen, um diese neuen Anforderungen zu erfüllen. Ich war hauptsächlich damit beschäftigt. Mein junger Assistent, Herr Kroeger, betreute die Analytiker, die für die Modellierung der Prozessteuerung zuständig waren. Er wurde von den beiden Lehrlingen unterstützt. Just in dieser kritischen Situation erfolgte ein Rückschlag.

Alle Monate gab es eine Managementsitzung an der alle für das Projekt zuständigen Manger beteiligt waren – der Entwicklungsleiter, der IT-Leiter und von der fachlichen Seite der Abteilungsleiter. Ich habe dort immer über den Stand des Projektes berichtet. Irgendwann im Frühjahr 1987 habe ich entschieden, auch Herrn K. zu dieser Sitzung mitzunehmen, da er mittlerweile in dem Projekt eine entscheidende Rolle spielte. Wie immer bei solchen Managertreffen in der Stahlindustrie haben die Bosse

ihre Zigarren herausgeholt. Formhalber haben sie gefragt, ob das jemanden stört. Gewöhnlich war jedem einverstanden – auch ich, obwohl ich unter den starken Tabakrauch gelitten habe. Diesmal kam es anders – zu meiner Überraschung hat sich unser Herr K. gemeldet und sagte, das Zigarrenrauchen würde ihn stören. Die Manager haben sich etwas verdutzt angeschaut und die Zigarren wieder eingesteckt. Die Sitzung ging weiter ohne Zigarrenrauch.

Eine Woche später wurde ich zum Entwicklungsleiter gerufen. Er teilte mit, dass Herr K. dort nicht mehr erwünscht sei. Ich habe diese Aufforderung kritiklos zur Kenntnis genommen, hatte ja geahnt, dass so etwas kommen würde – und habe Herr K. am nächsten Tag weggeschickt. Es war ein harter Schlag. Nicht nur habe ich meine rechte Hand im Projekt verloren, auch musste ich für ihn eine andere Beschäftigung finden. Es begann eine schwere Zeit für mich. Neben der Generierung der verteilten Tandem Programmen und Datenbanken, musste ich mich fortan auch mit den SoftSpec Anwendern in der Schienenproduktion befassen. Das war in Anbetracht meiner anderen Verpflichtungen wahrlich zu viel gewesen. Wir schrieben das Jahr 1987 und noch herrschte in der deutschen Industrie die unangefochtene Herrschaft der oberen Führungskräfte. Interessanteweise hatte die Bundeswehr sich von der Tradition der Wehrmacht befreit, die Industrie aber nicht. Günter Wallraff hatte gerade das Buch „Ganz Unten“ herausgebracht, in dem er die Herrschaftsstrukturen bei Thyssen stark anprangerte. Er hatte sich inkognito als türkischer Gastarbeiter in den Betrieb eingeschlichen und hatte erlebt, was es bedeutet, unter den autoritären Strukturen eines urdeutschen Industriebetriebes zu leiden. Herr Kroeger war bei der Bundeswehr verwöhnt was die Rechte des einzelnen Menschen anbetrifft. In der Industrie herrschten anderen Regel. Also ging Herr Kroeger nach Bremen zum Lagerhaus und ich blieb alleine mit den beiden Praktikanten bei Thyssen.

Bei Thyssen lief alles andere als glatt. Seit dem Herr Kroeger weg war, war ich dort doppelt belastet. Einerseits musste ich die Entwickler beraten, anderseits zusehen, dass die Werkzeuge richtig funktionierten. Mit der Generierung der Mainframe COBOL Programme hatten wir keine Probleme, aber mit den Tandem Programmen gab es Schwierigkeit und zwar mit dem Einzug der Copy Strecken. Dies geschah nämlich rekursiv. Eine Copy konnte andere Copies beinhalten, was auf dem Mainframe verboten war. Für die SoftGen Entwicklung bekam ich Verstärkung aus Budapest. Der neue Entwickler von SzAMOK war Doktor der Mathematik und glaubte alles besser zu wissen. Dabei konnte er allein nicht einmal ein Band einlesen. Statt den Codegenerator so wie er war weiter auszubauen, wollte er das Tool wieder von vorne an neu entwickeln. Ich musste ihn in dieser Absicht bremsen und überreden das aller Notwendigste zu machen um kompilierbare Tandem Programme zu

generieren. Mit ihm musste ich viel Zeit verbringen, Zeit die ich für die anderen Aufgaben dringend brauchte. Eine Zeitlang brach die Beziehung zu dem Endbenutzer ab. Sie bekamen lediglich die Programmentwurfe zur Sicht, die sie kaum zur Kenntnis genommen haben. Das war ein gravierender Fehler, der sich später rächen sollte. Es hat wenig Sinn Anwender mit komplexen Entwürfen zu überhäufen, sei es strukturierte Designmodelle oder UML Modelle. Ein Grundsatz der agilen Entwicklung lautet, ausführbarer Code ist immer besser als tote Dokumente [Fowl01]. Ich war im Begriff das zu verinnerlichen.

Für mich war der Sommer 1987 eine bittere Zeit. Das Thyssen-Stahl Projekt ging dem Ende zu. Wir sind mit unserem Top-Down Ansatz von der Anforderungsspezifikation bis zum lauffähigen Programm gescheitert. Zum Schluss ist es dennoch gelungen ausführbare Tandem-Programme zu generieren, aber es waren nicht die Programme, welche die Endbenutzer eigentlich haben wollten. Als das Schienen-Kontrollsystem zum ersten Mal live vorgeführt wurde, bemerkte der leitende Verfahrensingenieur lakonisch: „so haben wir uns das nicht vorgestellt“. Trotz aller Entwurfsdiagramme und Programmablaufbilder hatten die Endbenutzer nicht begriffen, was auf sie zukommt. Erst am Ziel der Reise bemerkten sie, dass sie irgendwo waren, wo sie eigentlich nicht hin wollten. Was nun?

Laut dem Vorgehensmodell hätten wir die Anforderungsspezifikation in SoftSpec überarbeiten und über SoftCon und SoftGen den Code wieder generieren und testen müssen. Das hätte jedoch mindestens zwei Monate gedauert. Das Alternativ war der bereits generierte Code zu flicken um auf dieser Weise die neuen Anforderungen zu erfüllen. Dadurch würde aber der Code nicht mehr im Einklang mit der Spezifikation und dem Entwurf sein. Der Zusammenhang zwischen Code und Dokumentation ginge dadurch verloren. Wir hätten rückwärts den Entwurf und dann die Spezifikation dem Code anpassen müssen.

Wir haben unter dem Druck, fertig zu werden nachgegeben und den Code direkt verändert. Es folgten weitere Tests und nach jedem Test wurde wieder etwas geändert. Wir waren in einer Anpassungsschleife, der schlimmste Fall für den Lieferant wenn er einen Festpreisvertrag hat. Es hat noch einige Monate gedauert, aber schließlich wurde das System abgenommen Für mich war das ein Piräus-Sieg. Die aufwändige Anforderungsspezifikation mit SoftSpec war abgehängt und nutzlos geworden. Es hätte wiederum viel Zeit und Geld gekostet, den SoftCon Entwurf zu aktualisieren. Das Projektbudget war erschöpft. Ich habe meinen Kostenrahmen gesprengt und musste den Abschluss der Entwicklung aus der eigenen Tasche bezahlen. Ungefähr zur gleichen Zeit ist der Projektleiter vom Nachbarprojekt – die Stranggussanlage –

nachts auf die Autobahn gegangen und ließ sich überfahren. Auch er ist mit seinem Projekt nicht termingerecht fertig geworden. Er konnte mit seinem Versagen nicht fertig werden. Soweit wollte ich nicht gehen aber für mich war das eine harte Lehre. Ich musste einen großen Verlust einstecken und war selber schuld.

Trotz Misserfolg und finanzieller Verlust habe ich entschieden weiterzumachen, aber eins habe ich mir versprochen - nie wieder ein Kundenentwicklungsprojekt zu machen. die Risiken sind einfach zu hoch und die Kosten nicht kalkulierbar. Wir hätten erst einen Prototyp von dem System bauen sollen, damit die Endbenutzer ihre wahren Anforderungen aufdecken könnten, aber das kam für diese Anwender nicht in Frage. Sie waren nur bereit in endgültige Produkte zu investieren. Sie wollten nicht wahr haben das sie nicht entwickeln können was sie nicht definieren können und um es definieren zu können müssen sie es erst entwickeln. Dadurch hätten wir das gleiche System zwei Mal gebaut, einmal für die Entdeckung der Anforderungen und ein zweites Mal für die Produktion. Die Kosten wären fast doppelt gewesen. Das spricht für eine agile Entwicklung, die jedoch zu diesem Zeitpunkt keiner kannte. Das agile Manifest erfolgte erst 12 Jahre später.

5.11 Der Verlust von Bertelsmann als Stammkunde

Bald darauf folgte der nächste noch größere Rückschlag. Die betriebsinterne Auseinandersetzung zwischen der zentralen Datenverarbeitung und den Fachabteilungen bei Bertelsmann hat sich zugespitzt. Der Aufstand der Fachabteilungen gegen die Bevormundung der zentralen Datenverarbeitung wurde immer stärker und im Sommer 1987 erreichter er einen Höhepunkt. Der Vorstand – ein Dr. M. - der bisher hinter dem DV-Leiter – Herrn B. – stand und ein Freund der Ungarn war, ist zurückgetreten. Ein neuer Vorstand ist aus dem Vertrieb gekommen. Unser Gönner – Herr B. - war zu der Zeit im Urlaub. Als er zurückkam, ist er mit sofortiger Wirkung seines Postens enthoben. Ein neuer DV-Leiter kam von einer Tochtergesellschaft in Baden. Seine erste Aktion war den Vertrag mit uns zu kündigen. Die Gängelung der Fachabteilungen durch die allmächtige DV-Abteilung soll ein Ende haben. Die Fachabteilungen wollten keine detaillierten Spezifikationen mehr schreiben und sich zu dem Inhalt bekennen. Sie wollten frei sein, zu tun und lassen was sie wollten. D.h. die Entwickler sollten bei ihnen sitzen und direkt unter ihrer Regie arbeiten. Der Rückkehr zur verteilten Entwicklung kündigte sich an. Wir durften unser Raum noch bis Ende des Jahres behalten und dem Testrechner bis dahin auch weiter nutzen, aber dann war Schluss. Nach fünf Jahren war es dort mit der SoftOrg Entwicklung aus. Ich musste eine andere Mainframe Rechenmöglichkeit suchen.

Diese Kündigung durch Bertelsmann kam für mich zu einem sehr ungünstigen Zeitpunkt. Das Thyssen Projekt war gerade in einer kritischen Phase. Wir waren dabei, die Programme zu generieren. Die Projekte bei der Bahn sind angelaufen und waren bereits spezifiziert. Beim Bremer Lagerhaus hatte das erste Projekt mit SoftOrg gerade begonnen. Bei Krupp war das Projekt für die Bahn in der Testphase. Die Programme wurden generiert, es gab aber eine Menge Fehler noch zu korrigieren. Soft-Test war dort im Einsatz und die Krupp Tester mussten intensiv betreut werden. Leider war der Dr. Gerisch schon weg und wir hatten keine Tester mehr. Ich hätte neues Personal als Tester ausbilden müssen, aber dazu fehlte die Zeit.

5.12 Verlegung des Testbetriebes nach München

Bei dem Bremer Lagerhaus war das erste Projekt schon angelaufen. Der Projektleiter dort durfte früher an unseren Seminaren als Student kostenlos teilnehmen und ich hatte ihn mit seiner Diplomarbeit geholfen. Er war deshalb schon mit unseren Werkzeugen vertraut und konnte zusammen mit Herrn Kroeger die Anforderungsspezifikation des neuen Container-Lagerhaltungssystems in kurzer Zeit fertig stellen. Ich musste am Anfang nur zu Schulungszwecken dahingehen.

Die Bahnprojekte in Frankfurt konnte ich dem Herrn Schäfer voll überlassen. Er hatte sich hervorragend bewährt und konnte sich dort souverän behaupten. Ende 1987 habe ich entschieden, die Geschäftsführung der SES mit ihm zu teilen. Neben dem Büro in München gab es fortan ein zweites Büro in Fulda, wo er wohnte. Dadurch wurde Frau Molnar auch etwas entlastet. Sie war mit dem Rechnungswesen, Buchhaltung, Büroführung, den Versand des Newsletters und der Organisation der monatlichen Lehrveranstaltungen am Ende ihrer Kräfte. Wir haben deshalb auch entschieden, das Newsletter statt viermal, nur zweimal jährlich zu versenden.

Ich musste nach der Kündigung bei Bertelsmann nach einen anderen Rechenmöglichkeit umschauen. Wir brauchten für die Werkzeugentwicklung weiterhin Mainframe Rechenkapazität. In Budapest gab es keine Aussicht auf einen neuen Rechner. Ich musste einen Mainframe-Rechner in Deutschland finden wo wir auch Arbeitsplätze hätten.

Der entlassene DV-Leiter von Bertelsmann, zu dem ich weiterhin einen persönlichen Kontakt pflegte, empfahl seinen ehemaligen Adjutanten. Der leitete inzwischen ein Rechenzentrum in München bei einer großen Versicherung. In diesem Rechenzentrum war es möglich, Rechenzeit zu mieten. Auch Räume wurden zur Verfügung gestellt. Er war in der Prinzregentenstraße in München, also nicht weit von unserem

Büro. Für mich war diese Lösung jedenfalls bequemer aber sie hatte ihren Preis. Fortan müssten wir für jede Rechenstunde bezahlen, sowie früher in Ungarn. Dies war eine neue große Belastung für unser Budget. Ohne die Einnahmen von den EU-Forschungsprojekten hätten wir uns das nie leisten können. So konnten wir zumindest einen Teil der Rechenkosten gegen das neue Metrik-Forschungsprojekt – METKIT – buchen.

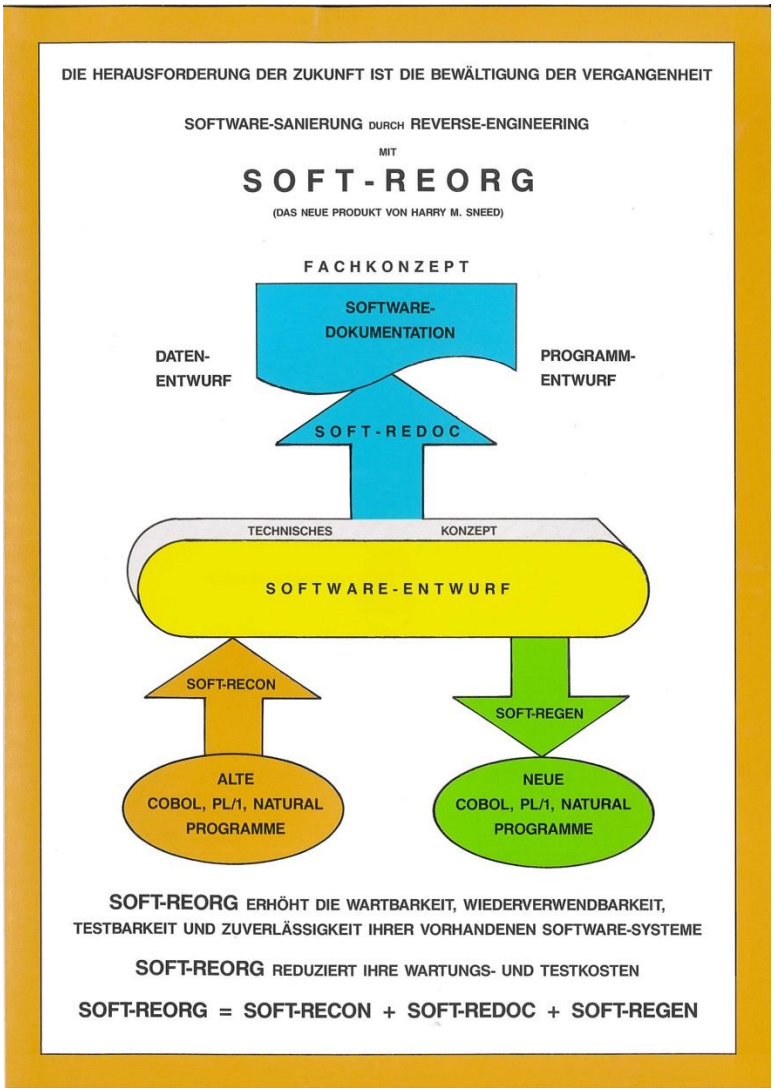
Bis Ende 1987 war unser Raum in Gütersloh geräumt. Wir haben uns von den Bertelsmann Entwicklern verabschiedet. Auch sie waren mit der neuen Situation nicht gerade glücklich. Sie wurden in die einzelne Fachabteilungen versetzt und mussten dort neben den Anwendern auf Zuruf arbeiten. Die Zeit der DV-Herrschaft war vorbei.

Als das Jahr 1987 zu Ende ging, habe ich geahnt, dass es so nicht weiter gehen konnte. Ich konnte nur hoffen, dass, wenn wir ausharren, doch etwas Positives geschehen würde, um die SES und die Softorg Entwicklung zu retten. Ich begann schicksalsgläubig zu werden – nicht ich steuerte die Ereignisse, die Ereignisse haben mich gesteuert. Im Januar und im März 1988 habe ich die letzten zwei Seminare bei Bertelsmann gehalten, mit den Thema Systemspezifikation und Programmgenerierung. Noch haben die Entwickler dort mit den Tools gearbeitet. Am Rande der Veranstaltung im März traf ich mich mit dem alten DV-Leiter, Herrn B. Er schilderte mir, wie er vorhat, aus der DV-Welt auszusteigen und ein neues Leben anzufangen. Bei Bier und Rotwein waren wir uns einig wie verworren und ungerecht die IT-Welt ist. Das war das letzte Mal, dass ich ihn sehen würde. Später erfuhr ich, dass er nach Jugoslawien zum Segeln in der Adria gegangen ist. Noch später habe ich erfahren, dass er dort in den Bürgerkriegswirren verschollen ist. In den 90er Jahren wurde er endlich offiziell für tot erklärt.

5.13 Der Umstieg auf Reengineering

Den Vertrag mit dem Alldata Rechenzentrum in München habe ich notgedrungen abgeschlossen und ab Juli 1988 haben wir begonnen dort die SoftOrg Werkzeuge weiter zu testen. Inzwischen hatte auch Thyssen die Zusammenarbeit beendet. Es blieben uns nur noch die Bahn, Krupp, Continental Reifen und das Bremer Lagerhaus als Entwicklungskunden erhalten. Auf der anderen Seite gewannen wir neue Testkunden. Das größte Testprojekt haben wir von Lidl und Schwarz in Neckarsulm bekommen. Wir mussten Ideal Programme mit einer IDMS Datenbank für die Lagerhaltung testen. Das Team von Frau Erdös hat die Tools SoftTest und SoftInt dort installiert und vier Monate lang zusammen mit den Entwicklern vor Ort gearbeitet die Tools der

neuen Umgebung anzupassen. Das war unser erster großer Integrationstest. Wir haben künstliche Datenbanken generiert, den Test unter Kontrolle eines Testmonitors mit einem Bildschirmsimulator auf dem Mainframe durchgeführt und die Ergebnisdaten gegen die Assertions geprüft. Es war eine gewisse Wiederholung des alten Tornado Lagerhaustest aus dem Jahre 1982. Die Werkzeuge waren aber jetzt viel weiter und der Test viel umfangreicher. Mit weniger Tester konnten wir viel mehr testen als zuvor.



Zur gleichen Zeit bekamen wir ein Integrationstestprojekt bei einer großen Krankenversicherung in Hamburg. Dort musste Frau Erdös große IMS-Datenbanken generieren und validieren. Sie musste zwischen Hamburg und Heilbronn hin und her pendeln. Wir trafen uns mal in Heilbronn, mal in Hamburg um das weitere Vorgehen zu besprechen.

Von der Firma Hellige in Freiburg bekamen wir ein Testprojekt für medizinische Röntgengeräte. Hellige hatte, wie auch wir bei Thyssen, ein Werkzeug eingesetzt, um die Software für das Röntgengeräte zu spezifizieren und modellieren. Ihr Werkzeug war eins der neuen CASE Tools aus Amerika - IEF von der amerikanischen Texas Instruments Inc. Das Resultat war das Gleiche wie bei SoftOrg. Nach mehreren Monaten der Modellierung stellte sich heraus, dass das Modell so nicht umsetzbar war. Die generierten C Programme waren nicht das, was sie haben wollten. Sie standen, wie wir beim Thyssen vor der Wahl, zurückzugehen und das Modell zu überarbeiten, oder vorzupreschen und den Code beim Test zu überarbeiten. Sie haben sich – wie wir bei Thyssen – entschieden, den Code anzupassen. Damit haben sie sich von dem Case Tool abgekoppelt.

Wir bekamen die Aufgabe neben den Entwicklern zu testen und ihnen die Fehlermeldungen zu übergeben. Dieses Paralleltestverfahren dauerte einige Monate, bis die Software das tat, was die Anwender wollten. Aus der ursprünglichen CASE Entwicklung wurde gezwungenermaßen eine agile Softwareentwicklung. Dieses und das Thyssen Projekt sowie die Erfahrung bei Bertelsmann hat mich animiert, einen umstrittenen Beitrag zur ICSM-90 „Top-Down Approach considered harmful“ zu schreiben [Sned90]. Es hatte sich herausgestellt, dass die Anwender nicht in der Lage sind, ihre Anforderungen klar und deutlich zu spezifizieren, weil sie sie nicht kennen. Sie müssen erst lernen, was sie eigentlich wollen und das kann nur im Verlauf eines Projektes geschehen. Also ist Softwareentwicklung gleich einem Forschungsprojekt. Es geht darum die Anforderungen zu entdecken. Nebenbei wird die Software hergestellt aber mit fortwährenden Verbesserungen und Ergänzungen. Schon vorher hatten einige namhafte Informatikwissenschaftler das Lebenszyklus Modell in Software Engineering in Frage gestellt [Brok87]. Es folgten noch weitere skeptische Stellennahmen von Spillner in Deutschland [Spil94] und Jackson in England [Jack98] ob Software Engineering in der Praxis je ankommen würde. Mary Shaw, Professorin an der Carnegie-Mellon Universität in Amerika glaubte noch dran [Shaw90]. Auch Ernst Denert in Deutschland blieb ein standhafter Vertreter von Software-Engineering obwohl auch er sich über die Kluft zwischen Theorie und Praxis beschwerte [Dene93]. Ich empfand dass meine Zeit zu Ende neigte und versuchte meine Kenntnisse und Erfahrungen zum Thema Software Engineering in Artikel zusammenzufassen. Über den „State

of the Art“ im Softwaretest schrieb ich einen Bericht für das InformatikSpektrum [Sned88] und über den Stand der Software-Engineering im Allgemeinen verfasste ich einen Artikel für die EDV-Fachzeitschrift des Hanser Verlags – PIK – mit dem Titel „Software-Engineering – ein Überblick [Sned89]. Darin stellte ich fest, dass Wartung und Wiederverwendung die Entwicklung als Hauptthemen der Software Engineering in den 90er Jahren ablösen würden. Ich war schon auf dem besten Wege dahin.

In dieser Zeit habe ich begonnen, mich mehr mit einem anderen Teilgebiet zu befassen, nämlich die Aufarbeitung bestehender Systeme. Thyssen hatte uns eine Reihe alte COBOL Programme gegeben, die wir für die Wiederverwendung in einem neuen System sanieren sollten. Zur gleichen Zeit bekamen wir alte Pascal Programme für die Verkabelung von Lokomotiven von der BBC in Mannheim. Die Entwickler dieser Programme hatten die Firma verlassen und kein Mensch wusste mit dieser Codemenge von über 200.000 Anweisungen etwas anzufangen. Die anderen Entwickler dort waren nämlich COBOL Programmierer. Dennoch waren die Programme unternehmenskritisch und mussten weiterentwickelt werden. In dieser Situation ist das Management auf mich zugekommen und hat um Hilfe gebeten. Ich versprach zu machen, was ich konnte. Wir haben die Programme erst automatisch redokumentiert und dann teils automatisch, teils manuell restrukturiert. Am Ende haben wir sie in COBOL umgesetzt, damit die anderen Entwicklern dort mit ihnen umgehen konnten. Dies war für mich eine Bestätigung, dass so etwas möglich ist, man braucht dazu nur die richtigen Werkzeuge. Also habe ich mit dem Teams von Erika Nyary und Gabor Jandrasics begonnen, eine neue Werkzeugreihe – SoftReorg – zu entwickeln. Zwei Komponente - SoftRecon und SoftRespec - waren schon vorhanden.

In dem Bremer Lagerhaus habe ich diesen Weg ebenfalls eingeschlagen. Bei der Migration eines alten Systems von Natural-1 zu Natural-2 haben wir statt neue Natural-2 Programme zu generieren, die alten Natural-1 Programme nachdokumentiert und daraus eine Spezifikation in der SoftSpec Sprache abgeleitet. Das Werkzeug dafür war SoftRespec. Nachdem die alte Programmlogik in der SoftSpec Repository gespeichert war, haben die Entwickler das Modell angepasst und anschießend Natural-2 Programme generiert. Frau Erdös hat den Natural Generator neben ihren anderen Testprojekten geschrieben. Ich nannte dieses Verfahren – der Generierung neuer Software aus der alten Software – Software Recycling. Man konnte damit viel schneller, viel billiger und viel risikoloser zu einem lauffähigen Neusystem kommen als wenn man wieder von vorne mit einer Neuentwicklung anfängt. Dieser Ansatz hat auch den entscheidenden Vorteil, dass die Kosten der Softwarewiederaufbereitung einigermaßen abschätzbar sind. Ich konnte diese Dienstleistung zu einem festen Preis anbieten. Allerdings muss der Anwender bereit sein, Kompromisse zu machen. Er

muss mit der alten fachlichen Lösung weiterleben. Man darf an der Fachlichkeit nichts ändern. Der fachliche Inhalt bleibt gleich, die technische Struktur verändert sich. Der Versuch dieses Konzept der Softwarewiederverwendung zu verwirklichen sollte mich noch viele Jahre weiter beschäftigen.

6 Die Wende

Am Ende der 80er Jahren erfolgte der große Wende. Das sozialistische System ist in sich zusammengebrochen und mit ihm mein Geschäft. Es war vorauszusehen dass es kommen würde, aber in dem Moment wo es tatsächlich kommt ist man überrascht. Man hat sich zu lange an den Gedanke gewohnt, dass es irgendwie weiter gegen würde. Als es doch gekommen ist, ging alles so schnell, das man gar nicht mehr mitkommt. Ein morsches Gebäude in dem man sich schlecht oder recht eingerichtet hat beginnt auseinander zu fallen und es gibt kein Halt mehr. Als die staatlichen Subventionen weggefallen sind, konnten sich meine Partnerinstitute nicht mehr halten und wurden innerhalb kürzester Zeit abgewickelt. Die Mitarbeiter müssten zusehen wie sie weiterkommen, jeder für sich. Sie hatten plötzlich das was sie schon immer gewollt haben – die große Freiheit. Für viele war dieser Wende ein großes Glück, für mich war es eine Katastrophe. Ich musste mitten in meinem Leben ein neues Kapitel aufschlagen.

6.1 Konsequenzen aus der Niederlage bei Thyssen-Stahl

Das Ende des Thyssen Projektes markierte das Ende meines Versuches Softwareentwicklung zu automatisieren. Jetzt wusste ich, dass dies nicht gelingen könnte. Wir müssten unsere Ziele ändern. Nachher fiel mir ein Artikel ein der bereits 1982 in dem ACM Software Engineering Notes veröffentlicht wurde mit dem Titel „Problems with the Software Life Cyle Model“. Darin haben einige renommierte Computer Scientists in einer Reihe Aufsätze mit dem Titel „Life cycle considered harmful“ vor der Automatisierung des Lebenszykus-Modells gewarnt [McCra82].

Michael Jackson schrieb: „System requirements can never be stated fully in advance, not even in principle, because the user doesn`t know them in advance - not even in principle“ [Jack81].

Michael Gladden behauptet in seiner Stellungnahme “Stop the Life Cycle – I want to get off” das der konventionelle Lebenszyklusansatz die Probleme der Softwareerhaltung nur verschärft: „Each modification to the requirements adversely effects the system by impacting each subsequent task ... the result is a vicious cycle compounding the maintenance problem ... requirements are always incomplete when development begins“ [Glad82].

Professor Bob Balzer von der Universität Southern California notiert seinem Aufsatz über die inevitable intertwining of specification and implementation „In actual practice developments steps are not refinements of the original specification, but instead redefine the specification itself ... there is a much more intertwined relationship between specification and implementation than the standard rhetoric would have us believe“ [Balz82].

Der Computer Wissenschaftler Alan Perlis unterstreicht die heuristische Natur einer Softwareentwicklung wenn er behauptet Menschen lernen indem sie experimentieren. Sie sind nicht in der Lage auf Anhieb nahezu perfekte Mechanismen zu konstruieren [Perl76]

Der Philosoph Prof. Fetzer von der Universität Minnesota schreibt dazu “Programs are mere conjectures and testing is an attempted and all too frequently successful refutation”. Aus der Sicht eines Mathematikers sind Programme lediglich Hypothesen und ein Test ist der allzu oft erfolgreiche Versuch diese Hypothesen zu widerlegen [Fetz88]

Die Professoren Rich und Waters von M.I.T kommen zum Schluß: „Writing a complete specification in a general-purpose specification language is seldom easier and often incredibly harder than writing a program. Furthermore, there has been little success in developing automatic systems that compile efficient programs from specifications“. Die hätten es wissen müssen. Immerhin hatten sie seit Anfang der 60er Jahren an dem Automatic Programming Projekt bei M.I.T. mitgearbeitet [Rich88].

Ich hatte diese Artikel sogar ausgeschnitten aber nie richtig zur Kenntnis genommen. Zu sehr hat mich der Zeitgeist jener Zeit beeinflusst und er unterschrieb das Wasserfallmodell. Ich war besessen diesen Software-Entwicklungsprozess zu automatisieren und diese Besessenheit hat mich blind gemacht. Die Top-Down phasenweise Vorgehensweise war das Modell auf dem ich aufbaute. In dem Schienenkontrollprojekt wurde mir alles klar. Ich bin davon ausgegangen, dass die Anwender wissen was sie wollen und dies auch artikulieren können. Jetzt stand ich dort, ich armer Tor und wusste nicht weiter. Es folgte eine Menge Änderungswünsche und wir standen vor der Frage, ob wir zurückgehen und die Spezifikation ändern oder ob wir die Änderungen gleich in den Code einbauen sollen. Hätten wir die Spezifikation im Sinne des Wasserfallmodells angepasst, hätten wir alle darauf folgenden Transformationen wiederholen müssen. Das hätte zu lange gedauert und es wären dadurch neue Fehler entstanden, Fehler, die wir glaubten längst korrigiert zu haben. Allerdings wären Spezifikation, Entwurf und Code weiter im Einklang miteinander geblieben. Mit

der direkten Anpassung des Codes wurden die Wünsche der Anwender schnell erfüllt, aber der Code wäre nicht mehr im Einklang mit dem Entwurf und der Anwenderspezifikation. D.h. wir hätten den Transformationsprozess nie wiederholen können. Spezifikation und Code müssten getrennt weiterentwickelt werden und würden unweigerlich auseinander driften. So hätten wir die gleiche Situation wie es schon früher war. Die fachliche Spezifikation und die technische Entwurfsdokumentation wären nur zur Planungszwecke gewesen. Wir waren dort bei Thyssen mit den SoftOrg Werkzeugen dort schon angekommen wo die anderen CASE Toolhersteller in Amerika erst hinkommen mussten. Sie hatten gerade begonnen mit Millionen von Investmentkapitaldollar Tausende nichts ahnenden IT-Anwendern in diese falsche Richtung zu verführen, einschließlich IBM mit ihrem hochgelobten AD-Cycle Entwicklungssystem. Dieser fehlgeleitete Versuch das Wasserfallmodell zu automatisieren würde dazu beigetragen das Weltkonzern an den Rand der Pleite zu treiben [Mont91]

Trotz Misserfolg und finanziellen Verlust habe ich entschieden weiterzumachen, aber eins habe ich mir versprochen: nie wieder ein Kundenentwicklungsprojekt zu machen. Die Risiken sind einfach zu hoch und die Kosten nicht kalkulierbar. Die Entwickler sollten erst einen Prototyp bauen, damit die Endbenutzer ihre wahren Anforderungen aufdecken können. Dann kann man sich schrittweise an eine endgültige Lösung heranarbeiten, aber dafür war bei Thyssen keiner bereit zu bezahlen. Das bedeutet das gleiche System zwei Mal zu bauen, einmal für die Entdeckung der Anforderungen und ein zweites Mal für die Produktion sowie es Donald Knuth schon in den 60er Jahren propagiert hat [Knut74]. Die Kosten scheinen sich zu verdoppeln. Aber am Ende ist es vielleicht doch billiger. Das spricht für eine agile Entwicklung, die jedoch zu diesem Zeitpunkt keiner kannte. Das agile Manifest erfolgte erst 15 Jahre später.

6.2 Die letzten Anwendungsentwicklungsprojekte

So schnell – von heute auf morgen – konnten wir mit den Entwicklungsprojekten doch nicht aufhören. Wir hatten noch festvereinbarten Entwicklungsprojekte für Continental Reifen in Hannover, für das Bremer Lagerhaus in Bremen und für die Bundesbahn in Frankfurt. Bei Krupp-Atlas wurde die Entwicklung glücklicherweise nach „Time and Material“ bezahlt.

6.2.1 Ein Produktionssteuerungssystem mit ADR-Online

Continental Reifen in Hannover hatte zu diesem Zeitpunkt schon lange SofSpec in Einsatz um ihre betriebliche Informationssystem zu spezifizieren. für die Programmentwicklung benutzten sie eine 4GL Sprache der Firma ADR. Wie Natural von der

Software AG mit dem ADABAS Datenbanksystem war ADR-Online mit dem ADR Datenbanksystem verbunden. Diese 4GL Entwicklungsumgebung war durch SoftCon nicht abgedeckt. Also musste dieser Anwender den Übergang von der Spezifikation zum Programmcode auf eigener Faust schaffen. Dies erwies sich als äußerst aufwendig, so bat er uns um ein ADR-Online Generator.

Das Team von Frau Nyary übernahm die Aufgabe und schaffte es innerhalb 6 Monaten den gewünschten Generator zu bauen. Wir wurden auch beauftragt, den Kunden beim ersten Einsatz zu unterstützen. Es handelte sich um ein komplexes System für die Planung, Produktion und Steuerung der Reifenproduktion. Das Projekt war hier voll in der Verantwortung des Anwenders, unser Team hat lediglich Toolunterstützung geleistet. Bis auf ein paar Ausbildungsveranstaltungen und gelegentliche Beraterbesuche habe ich das Projekt der Frau Nyary überlassen. Es gab am Anfang die üblichen Detailprobleme mit dem generierten ADR- Online-Programmen und der Test dauerte doppelt so lange wie geplant, aber nach einem Jahr waren die ersten Teilsysteme in Produktion. Der Kunde konnte ohne uns weiterarbeiten.

6.2.2 Ein Container Lagerverwaltungssystem mit Natural/Adabas

Mit dem Bremer Lagerhaus war es nicht so einfach. Die IT dort sollte ein Lagerhaltungssystem für ihr großes Container-Lager entwickeln und die Container vom Eingang bis zum Ausgang verfolgen. Die vorderen Online-Dialog Programme wurden mit Natural/ADABAS und die hinteren Batch-Programme mit COBOL implementiert. Sowohl die Natural Programme als auch die COBOL Programme haben wir aus SoftCon heraus generiert. Die Spezifikation des Systems wurde mit SoftSpec zusammen mit dem Kunden ausgearbeitet. Herr Kröger und Herr Funk kamen von dem Thyssen Projekt und halfen bei der Spezifikation der Anwendung. Sie konnten ihre Erfahrung bei Thyssen gut einbringen. Hier haben wir abgewartet bis die Spezifikation weitestgehend fertig war, ehe wir ein Festpreisangebot abgegeben haben. Bis Ende 1988 war es endlich so weit. Es wurden drei Phasen definiert – Entwurfsphase, Implementierungsphase und Testphase. Am Ende jeder Phase war ein Drittel der Gesamtkosten fällig – sofern das Phasenergebnis vom Kunden abgenommen wurde, also ein klassisches Wasserfallprojekt.

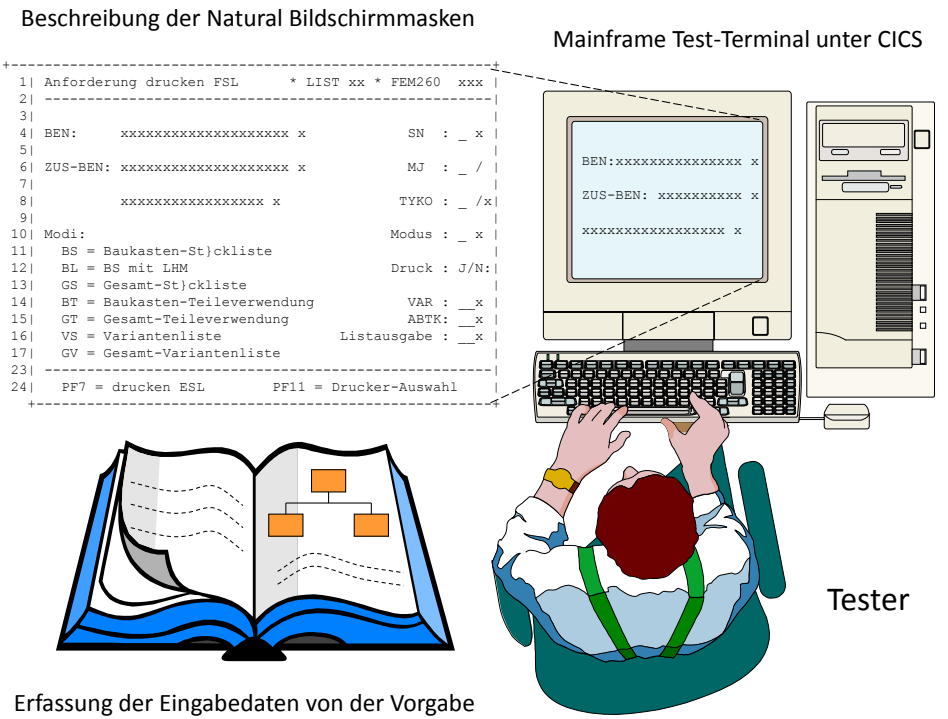
Das SoftCon Team unter Leitung von Laszlo Fehervari war für den Entwurf zuständig. Sie haben das SoftSpec Datenmodell in ein allgemeines Datenbankschema und das SoftSpec Funktionsmodell in Online-Programmentwürfe sowie in Batch-Programmentwürfe umgesetzt. Die beiden Entwürfe hatten jeweils einen anderen Aufbau und eine andere Ablauflogik. Die Umsetzung selbst fand in einer einzigen Nacht statt. Der SoftCon Generierungslauf dauerte fast 8 Stunden auf dem Mainfra-

me. Anschließend musste das Team weitere sechs Wochen damit verbringen, die Natural und COBOL-spezifische Eigenschaften einzubauen. Nach zwei Monaten konnte das SoftGen Team die Entwurfsdatenbank übernehmen. Die erste Teilzahlung war fällig.

```
LAGERBEWEGUNG                          (MUSTERBELEG)

         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
+--------------------------------------------------------------------------------+
|                                                                                |
|                          L A G E R B E W E G U N G                             |
|                          =========================                             |
|                                                                                |
|          DATUM: XXXXXXXX                                  UHRZEIT: XXXXX       |
|                                                                                |
|          BITTE.EINGEBEN:                                                       |
|                                                                                |
|          BEWEGUNGSART..====> X        (Z.=.ZUGANG.|.A.=.ABGANG.|.E.=.ENDE)    |
|          ARTIKELNUMMER.====> 999999                                            |
|          ARTIKELMENGE..====> 999999                                            |
|          KUNDENNUMMER..====> 999999                                            |
|                                                                                |
|          MELDUNG.......====> XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |
|  FEHLERTEXT: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |
|  NEUE.MELDUNG: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX                             |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
+--------------------------------------------------------------------------------+
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6         7         8
```

```
SEQU EBENE                              OPTIONS:

                *SYSTEM*
 1.0  1     |---> * ARTIKEL (2000)
 2.0  2     |       |--> @* ARTIKELBEWEGUNG
 3.0  3     |       |       |
            |       |       +-----> ARTIKELPROTOKOLL
 4.0  2     |       |--> @* LAGERBEWEGUNG
 5.0  3     |       |       |
            |       |       +---> @ BESTELLPOSTEN
 6.0  4     |       |               |
            |       |               +-----> BESTELLISTE
 7.0  2     |       +---> @ LIEFERANTENAUFTRAG
 8.0  3     |               |
            |               +---> * BESTELLPOSTEN (20)
 9.0  1     +---> * LIEFERANTEN-DATEN (150)
10.0  2             |--> @* LIEFERANTENBEWEGUNG
11.0  3             |       |
            |       |       +-----> LIEFERANTENPROTOKOLL
12.0  2             +-----> ZZ-AKT-ARTIKEL

SEQU EBENE                              OPTIONS:

                AKTUALISIERE-KUNDEN
 1.0  1     |-----> INIT-AKTUALISIERE-KUNDEN
 2.0  1     |-----> LIES-KUNDEN-MASKE
 3.0  1     |-----> STEUERE-DIALOG
 4.0  2     |       +---> * VERARBEITUNG-DIALOG
 5.0  3     |               |---> - PLAUSI-KUNDENMASKE
            |               |           SIEHE SEITE 023
 6.0  3     |               |-----> STEUERE-STANDARD-INDIVIDUELL
 7.0  4     |               |       |---> @ STEUERE-STANDARD-AUSWAHL
 8.0  5     |               |       |       |---> @ VERARBEITE-ZUGANG
 9.0  6     |               |       |       |       |-----> KUNDEN-MASKE-AN-DATEN
            |               |       |       |       |
10.0  6     |               |       |       |       +-----> VERARBEITE-KUNDENZUGANG
11.0  5     |               |       |       |---> @ VERARBEITE-ABGANG
            |               |       |       |       |
12.0  6     |               |       |       |       +-----> VERARBEITE-LIEFERANTABGANG
13.0  5     |               |       |       +---> @ VERARBEITE-AENDERUNG
14.0  6     |               |       |               |-----> KUNDEN-MASKE-AN-DATEN
            |               |       |               |
15.0  6     |               |       |               +-----> VERARBEITE-KUNDENAENDERUNG
16.0  4     |               |       +---> @ STEUERE-INDIVIDUELLE-AUSWAHL
17.0  5     |               |               |
            |               |               +--> -@ DRUCKE-KUNDEN-LISTE
            |               |                         SIEHE SEITE 024
18.0  3     |               +-----> LIES-KUNDEN-MASKE
19.0  1     +-----> ENDE-AKTUALISIERE-KUNDEN
```

Aus der Entwurfsdatenbank wurden die einzelnen Natural und COBOL Programme, sowie das ADABAS Datenbankschema erzeugt. Schwierigkeiten gab es nur mit den Natural-II Masken. Die Attributen-Bytes konnten nicht alle automatisch gesetzt werden. Die Maskenverarbeitung musste manuell nachgebessert werden. Auch die COBOL Programme mussten in Detail angepasst werden, vor allem die Eingabe- und Ausgabe-Operationen. Diese sind in jeder Sprache schwierig, aber für ADABAS waren sie besonders kompliziert. Mit der Ablauflogik, die Auswahl- und Wiederholungsstrukturen gab es die wenigsten Probleme. Alle Programme wurden mit einem Standard Muster ausgestanzt, wie es in der Generierungstechnik allgemein üblich ist. Nach weniger als sechs Wochen hatten wir schon über 800 Programme generiert und kompiliert. Die zweite Teilzahlung war fällig.



Kurz davor an einen sonnigen Sommertag saß ich mit dem Leiter des Softgen Teams, Gabor Jandrasics, im Hafen und wir schauten zu, wie die Schiffe beladen wurden. Wir ahnten dass uns die größte Arbeit noch bevor stand – der Test. Testen wird immer unterschätzt. In fast allen Projekten macht der Test mindestens die Hälfte des Aufwands aus. Das heißt, wir hatten schon zweidrittel der Gelder bekommen aber

über die Hälfte der Arbeit lag noch vor uns. Dies wäre die beste Zeit gewesen auf ein Schiff zu steigen und in die Ferne zu reisen. Eigentlich durfte der Kunde erst bezahlen, wenn das ganze Produkt einwandfrei läuft, das wäre aber für den Auftragsnehmer keine gute Lösung gewesen, denn er hätte die ganze Entwicklung vorfinanzieren müssen. Ich meinte, es müsste möglich sein, das Gesamtprojekt in zahlreiche kleine Teilprojekte zu zerlegen und nach jedem Teilprojekt ein Teil des Gesamtproduktes liefern. Die Arbeit sollte komponentenweise statt phasenweise bezahlt werden. So könnte der Auftraggeber das Projekt stückweise über einen längeren Zeitraum bezahlen und trotzdem sichern, dass die Ergebnisse stimmen. Genau das haben die Autoren des agilen Manifests zehn Jahre später vorgeschlagen [Beck01]. Als ich das Manifest zum ersten Mal las, musste ich an diesem Gespräch an der Wasserkante in Bremen denken.

Wir sind jedenfalls nicht aufs Schiff gegangen – wir blieben an unserem Posten und haben weiter getestet. Da wir keine Testautomation für den Dialogbetrieb hatten, mussten wir jede Transaktion manuell eingeben und dabei die Daten variieren. Da es so viele Einzeldaten in jeder Maske gab, dauerte es bis zu 20 Minuten eine einzige Transaktion abzusenden. Falls ein Fehler auftrat, musste der Tester alles wiederholen. Bei der Beobachtung dieser mühseligen Arbeit fiel mir ein, dass wir diesen Testvorgang unbedingt automatisieren müssten. Er war der größte Engpass im Projekt – für dieses Projekt war das aber zu spät. Wir mussten durch und das hieß, mit großem Personalaufwand. Es kamen weitere Kollegen aus Budapest um mit zu testen. Ihre Reisekosten mussten bezahlt werden und so musste ich zuschauen, wie unser ohnehin bescheidener Gewinn dahinschmolz.

Der Test hatte doppelt so lange gedauert, als die anderen Phasen zusammen und hat das Dreifache gekostet. Den Aufwand den wir durch die Automatisierung der Entwicklung eingespart haben wurde durch den Testaufwand wieder wettgemacht. Es traten in fast jeder vierten Transaktion irgendwelche Fehler auf. Die Korrektur hat meistens nur wenig Zeit in Anspruch genommen, die Wiederholung des Tests dauerte aber wieder eine Viertelstunde. Bei 450 Online Transaktionen und 350 Batchläufe macht das was aus. Es gab kein Capture/Replay Testtool für Natural und wir hatten keine Zeit eins zu entwickeln. Es blieb nur übrig so schnell wie möglich die Tests abzuarbeiten. Das Schlimme ist, man weiß nicht, ob eine Transaktion fehlerfrei ist, bis man die letzte Variante der Transaktion getestet hat und gerade hier bei der Container-Verfolgung gab es etliche Varianten, z.B. ob Container per Schiff, per Zug, per Lastwagen oder per Luftfracht weitergeleitet wird und für jeden Transportweg gab es wieder etliche Varianten. Hinter jeder Transaktionsart steckt ein Baum von weit verzweigten Ablaufpfaden und jeder überquert eine andere Kombination von Codean-

weisungen. Wie wir festgestellt haben, genügte es auch nicht alle Anweisungen zu überdecken, wir mussten jede Kombination von Anweisungen bzw. jeder Anweisungspfad testen. Daher die vielen Testfälle und die lange Testdauer. Es war uns alle klar dass es nicht mehr so weiter gehen könnte. Etwas müsste geschehen um den Test großer Systeme zu beschleunigen [Bach01].

Wegen dem noch nicht abgeschlossenen Test wurde der Übergabetermin mehrfach verschoben. Zuletzt hat es gereicht und das Container-Verwaltungssystem ging Ende 1990 in Produktion. Wir hatten unseren Auftrag erfüllt, mussten aber am Ende wieder draufzahlen. Mit den Festpreis-Entwicklungsprojekten hatte ich nur Verluste. Es schien unmöglich zu sein die vollen Kosten im Voraus zu planen. Und wenn es möglich gewesen wäre, hätte der Kunde sie wahrscheinlich nicht akzeptiert. Keiner wollte wahr haben wie viel es kostet komplexe Softwaresysteme neu zu entwickeln und auszutesten.

6.2.3 Ein Rechnungssystem mit COBOL/DB2

Das Projekt bei Krupp-Atlas in Essen hatte schon währen des Thyssen Projektes angefangen. Zum Glück lagen die beiden Standorte nicht weit auseinander. Ich konnte mit dem Fahrrad zwischen Duisburg und Essen hin und her pendeln. Krupp-Atlas hatte einen Auftrag von der Bundesbahn bekommen, ein neues Abrechnungssystem zu entwickeln. Da die Bahn die Anforderungsspezifikation mit SofSpec erstellt hat, wollte der Projektleiter von Krupp die Programme mit SoftCon und SoftGen implementieren. Wir hatten die beiden Werkzeuge in Essen installiert und der SoftCon Team hat sie betreut. Die eigentliche Entwicklung fand in Troisdorf bei Bonn statt wo Krupp eine Entwicklungsstelle unterhielt. Im Sommer 1986 habe dort selbst mitentwickelt. Das Erlebnis dort hat mir nochmals deutlich gemacht, wie wichtig das Fachwissen ist. Mit Werkzeugen allein kann man keine Probleme lösen. Glücklicherweise haben wir für das Krupp-Atlas Projekt keine Verantwortung übernommen, nur für die Werkzeuge. Ansonsten haben wir lediglich ein paar Entwickler aus Ungarn beigesteuert.

Das Haupthindernis im Krupp Projekt war nicht die Werkzeuge, sondern der mangelnde Anwendungswissen. es ging um das Rechnungswesen. Die Bahnmitarbeiter in Frankfurt hatten die Abrechnungsvorgänge – teilweise sehr ausführlich mit Entscheidungsbäumen, Ablaufplänen und Datenflussdiagrammen – beschrieben, aber es gab immer noch weiße Flächen in der Vorgabe. Wenn fremde Programmierer ohne Anwendungs-Know-how am Werk sind, muss die Spezifikation die allerletzte Detailentscheidung regeln, d.h. die Spezifikation muss auf der gleichen semantischen Stufe wie das Programm selbst sein. Sonst muss der Entwickler ständig Rückfragen stellen.

Anderseits ist der normale Analytiker oder Sachbearbeiter kaum bereit so weit zu gehen. Oft können sie gar nicht so weit gehen weil sie selber nicht wissen. Es ist erstaunlich zu erkennen wie viele Menschen eine Tätigkeit Tag ein, Tag aus ausführen ohne diese genau erklären zu können. Das Ergebnis ist die Verständniskluft zwischen Anwendern und Entwicklern – die sogenannte Comprehension Gap [Shne80].

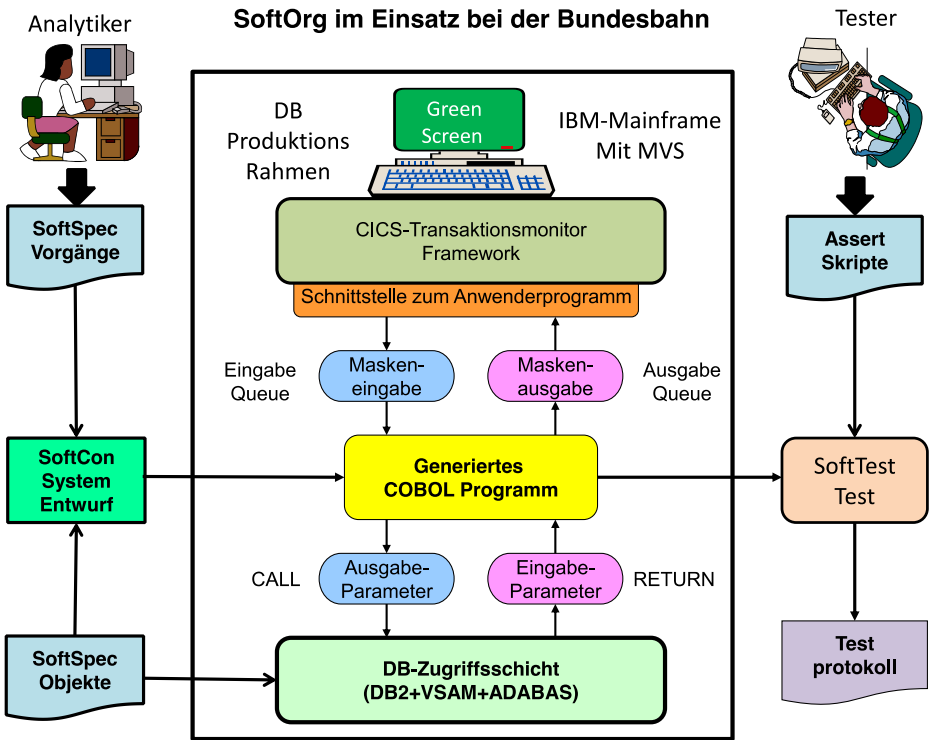
Zum Glück wurde das Krupp-Atlas Projekt in Troisdorf nach Aufwand bezahlt. Es dauerte wegen der mangelhaften Kommunikation zwischen den Anwendern und den Entwicklern noch lange bis es endlich fertig wurde, aber es wurde von der Bahn abgenommen. Krupp-Atlas hat daran gut verdient und wir bekamen unseren Anteil mit. Dennoch seitens des Auftraggebers war das Projekt falsch aufgesetzt. In Troisdorf hatten 9 Entwickler Zugang zu nur einem Fachexperten und das nur 3 Tage in der Woche. Ich selbst stand oft in der Schlange vor seinem Zimmer um eine Detailfrage zu klären. Leider ist ein Programm erst dann fertig wenn die letzte Detailfrage geregelt und getestet ist. Entweder hätten die Sachbearbeiter der Bahn jedes Detail in der Spezifikation vorschreiben müssen oder sie hätten neben den Entwicklern sitzen müssen. So hat sich das Abrechnungsprojekt sich ewig lange hingeschleppt. Irgendwann ist es doch schließlich fertig geworden. Ich war nicht mehr dabei. Ein ähnliches Projekt habe ich später in den 90er Jahren erlebt – das FISCUS Projekt vom Finanzministerium. Auch dort glaubte man, man könne die Entwicklung von der Analyse trennen. Das Projekt ist schließlich gescheitert. In beiden Fällen war der Projektleiter der Gleiche.

6.2.4 SoftOrg – all the way down

Gegen Ende der 80er Jahren hat die Bahn in Frankfurt begonnen ihre Software selber zu entwickeln. Die SoftOrg Werkzeuge – SofSpec, SoftCon und SoftGen – waren, dank dem Einsatz von Herrn Schäfer, der die Bahn betreute, dort auf dem IBM Mainframe installiert und im Einsatz. Nicht nur das Rechnungswesen war damit spezifiziert, ein anderes Projekt für die Steuerung der Güterverkehr wurde damit voll durchgezogen. In diesem Projekt ist es endlich gelungen ein lauffähiges COBOL/CICS Programm aus der Spezifikation über SoftCon ohne manuellen Eingriff zu generieren. Anschließend wurde das Programm mit SoftTest getestet und mit SoftDoc nachdokumentiert. Das Ergebnis wurde in einer Kneipe hinter dem Hauptbahnhof groß gefeiert. Zum ersten Mal konnte mit der SoftOrg Entwicklungsumgebung ein Anwendungssystem vollständig vom Stapel laufen. Es ist uns gelungen – zumindest in diesem Projekt –die Entwicklung zu industrialisieren. Das war nur möglich, weil die Sachgebietsexperten die Werkzeuge selber bedient haben. Die komplette fachliche Lösung bis zur letzten Bedingung wurde in die Spezifikation eingebaut. SoftCon hat diese fachliche Lösung in den vorgefertigten DB/DC Rahmen für

CICS und DB2 überführt und SoftGen hat daraus COBOL Programme generiert. Entscheidend hier war der ausgereifter technischer Framework der vor der hiesigen Projektgruppe erstellt und getestet wurde.

Parallel zu diesem Projekt lief ein anderes für das Bauwesen. Die Bahn wollte die Daten über alle Brücken und Tunnel zentral speichern und auswerten. Sie sollte damit die Wartung dieser Anlagen planen und steuern. Ich habe vorgeschlagen, das Tool SoftMan zu verwenden. Mit SoftMan konnten wir die verantwortlichen Dienststellen und die Standorte erfassen. Die Brücken und Tunnel wurden als Objekte, die Kontroll- und Wartungsvorgänge als Prozesse definiert. Die Mitarbeiter, die verteilten Computer und die eingesetzten Softwarepakete wurden als Betriebsmittel behandelt. Die Dienststellen haben Betriebsmittel gehabt, mit denen sie Kontroll- und Wartungsprozesse durchgeführt haben. Diese Prozesse haben die Objekte kontrolliert und geändert, bzw. nachgebessert. Dabei wurden die Zeiten und die Kosten erfasst. Ich habe großen Wert auf die Messbarkeit der Prozesse sowie der Produkte gelegt. Ich wollte eine Metrikdatenbank für künftige Projekte aufbauen [Sned91].



Die Bahn war mit dieser Lösung recht zufrieden. Die SES hatte jetzt einen guten Ruf bei der Bahn. Ich habe in der Folge ein weiteres Projekt in München bekommen, wieder etwas mit dem Rechnungswesen. Der Projektstandort war in München, die Projektmitarbeiter waren aber über die ganze Bundesrepublik verteilt. Für die Projektarbeit mussten sie nach München kommen. Sie sind Montag angereist und donnerstags Nachmittag wieder abgereist, da blieben für das Projekt gerade 2,5 Tage dazwischen. Die Projektmitarbeiter haben die meiste Zeit mit der An und Abreise verbracht. Das Projekt ist deshalb trotz Werkzeugunterstützung nur langsam vorangegangen. Für die anspruchsvolle geistige Arbeit in einem IT Projekte brauchen Menschen eine gewisse Anlaufzeit bis sie wieder drin sind. Projekte die in kurzen Intervallen immer wieder unterbrochen werden leiden darunter.

Für mich war diese ruhige Atmosphäre bei der Bahn eine willkommene Erholung von dem ganzen Stress bei den Projekten in der Industrie. Im öffentlichen Dienst laufen die Uhren eben langsamer. Deutschland war ein zweigeteiltes Land, nicht nur politisch in Ost und West, sondern arbeitsmäßig in dem ruhigen, geschützten Bereich des öffentlichen Diensts und dem turbulenten, hektischen Bereich der freien Marktwirtschaft. In dem öffentlichen Bereich spielen sich die IT-Projekte anders ab. Es fehlen der Kundendruck und damit die Dringlichkeit. Ich habe mich oft gefragt ob die Menschen wirklich in so einem aufreibenden Konkurrenzkampf auf einander gehetzt werden müssen um etwas leisten zu können. Das drohende Ende des sozialistischen Experiments schien das zu bestätigen.

6.3 Das Ende des Sozialismus zeichnet sich ab

Gerade dann, als die Softorg Werkzeuge endlich einigermaßen liefen und wir auch bessere Projekte in Aussicht hatten, kam der Wende. In Ungarn war es schon länger abzusehen, dass es so nicht mehr weiter laufen konnte. Die IT-Institute hatten viel zu viel Personal und zu wenig Einkommen. Sie mussten vom Staat subventioniert werden, der Staat war aber schon tief verschuldet. Man hätte die Hälfte des Personals entlassen müssen, angefangen mit den politischen Führungskräften. Dies war aber in dem Sozialismus nicht möglich. Es verstößt gegen die Prinzipien. So suchte man nach anderen Wegen die Krise zu meistern. Die Mitarbeiter der Institute wurden ermuntert, eigene Firmen innerhalb der staatlichen Institute zu gründen und ein Teil ihres Erlöses an das Institut abzugeben. Dies hatte merkwürdige Auswüchse. Überall sind private Unternehmungen aus den staatlichen Betrieben gesprossen.

Bei meinen ungarischen Mitarbeiter ging es ähnlich. Ihre Gehälter sind geblieben, die Preise schießen in die Höhe, so dass ihnen nichts anderes übrig blieb, als eine

zweite Beschäftigung nachzugehen. Sie haben kleine Firmen gegründet um zusätzliches Geld zu verdienen. Es war für mich eine schwierige Situation. Jetzt musste ich zweimal bezahlen, einmal an die Außenhandelsgesellschaft und einmal an die Firmen der Mitarbeiter. Das war für mich eine schwere finanzielle Belastung, denn laut meinen Vertrag als Agent der Außenhandelsgesellschaft musste ich 75% der Erlöse durch die Werkzeuge und der Projekte an sie abgeben. Jetzt musste ich die verbleibenden 25 % mit den Mitarbeitern der Institute teilen, damit sie weiterarbeiten.

Der Titel meines früheren Interviews in der ungarischen Wirtschaftszeitschrift traf voll zu. Mein Kapital steckte in der Tat in den Köpfen meiner Mitarbeitern. In der Software Branche kann man Mitarbeiter nicht ohne weiteres austauschen. Das hatte ich oft genug bei meinen Kunden erlebt. Wer komplexe Software Systeme entwickelt, begibt sich in eine große Abhängigkeit, nämlich zu denen, die jene Software implementieren. Nicht um sonst sind einzelne Schlüsselpersonen in der Lage die Firmen zu erpressen. Man kann sonst so viel kommentieren und dokumentieren, das volle Wissen darüber steckt in den Köpfen der zuständigen Entwicklern. In der kritischen Situation, in der ich mich befand, hatte ich keine andere Wahl als mitzumachen. Es hätte Jahre gedauert ein neues Team von Experten heranzuziehen. Dazu hatte ich weder die Kraft noch die Lust. Damals habe ich mir selbst geschworen, sollte ich je wieder eine neue Generation von Werkzeugen entwickeln, würde ich sie alle selber programmieren.

6.4 Eine Welt bricht zusammen

Ab 1988 ging der Zerfall des sozialistischen Systems in Ungarn immer schneller. Der Staat war pleite, der alte Parteisekretär, Janos Kadar ist zurückgetreten, die neue Führung bekam die Lage nicht mehr im Griff. Überall gab es Auflösungserscheinungen. Das machte sich bei meinen Partnerinstituten besonders bemerkbar. Nicht nur, dass die Mitarbeiter ihre eigene Arbeit besorgen mussten – das wäre nicht so schlimm gewesen – die Außenhandelsgesellschaft verlor ihre Kontrolle über das Auslandsgeschäft und die Betriebe verloren die Kontrolle über ihre Mitarbeiter. Im August 1989 fiel die Entscheidung der neuen Regierung die Grenze nach Österreich zu öffnen. Die ungarischen Bürger erhielten ihre Pässe und dürften ohne Einschränkung im Ausland arbeiten. Bis dahin war es mein Vorteil gewesen, dass meine Mitarbeiter ausreisen dürften – deshalb wollte jeder zu mir kommen. Jetzt ging diese Vormachtstellung verloren. Wer wollte, konnte fortan direkt für ausländische Kunden arbeiten. Es lag nur beim ausländischen Kunden die Arbeitserlaubnis zu besorgen. Außer Geld hatte ich keine andere Möglichkeit die Mitarbeiter an mich zu binden.

Mitten in diesem Chaos kam die Flüchtlingswelle aus der ehemaligen DDR. Jetzt, wo die ungarische Grenze offen war, wollten viele DDR Bürger über Ungarn in den Westen fliehen. In September 1989 war ich ausnahmsweise mit dem Auto in Budapest. Auf dem Rückweg standen schon viele junge Deutsche an der Autobahn und suchten nach einer Mitfahrtgelegenheit. Ich hatte zwei Mädchen aus Frankfurt an der Oder – beide Krankenschwester - mitgenommen und in dem Flüchtlingslager in Freilassing an der bayerischen Grenze abgesetzt. An der ungarischen Grenze gab es keine Kontrolle mehr. Vorher habe ich ihr sehnlichster Wunsch erfüllt und sie zum McDonald eingeladen. Als sie dort ihre Eltern angerufen haben sind die armen Eltern aus allem Himmel gefallen, als sie erfuhren, ihre Töchter sind im Westen. In meinem Kellerbüro in München dachte ich nach, wohin dass alles führen sollte. Es war nicht nur eine Wende in der politischen Landschaft, es war auch eine Wende in meinem beruflichen Leben. Nichts würde so bleiben, wie es war.

Danach haben sich die Ereignisse überstürzt. Mein Zimmer im Burgviertel wurde gekündigt. Ich musste in Budapest eine neue Bleibe suchen. Nach einigen Zwischenstationen landete ich im Gartenhaus von Frau Erdös am Rande der Stadt. Das SzAMOK Institut wurde aufgelöst, meine Mitarbeiter mussten ausziehen. Zunächst haben wir dort Zimmer für sie in dem alten Institutsgebäude gemietet. Später hat Gabor Jandrasics für sein Team ein eigenes Büro in Buda gemietet. Schon 1990 gründete er eine neue GmbH, die SoRing Kft. Frau Erdös ging zunächst mit ihrem Team zu einer Auffanggesellschaft. Dort hatten sie wenigsten einen Raum und einen, wenn auch veralteten, IBM Rechner gehabt. Diese Gesellschaft hat sich aber nur kurze Zeit gehalten. Später haben sich die Teammitglieder alle selbstständig gemacht und haben von zuhause aus gearbeitet.

Das SzKI Institut blieb noch eine Weile bestehen. Inzwischen hat Erika Nyary mit ihrem Team auch eine eigene GmbH gegründet – die Softing Kft. Sie zog damit in die frühere Altbauwohnung ihrer Eltern am Pester Donauufer ein. So konnte wenigstens die SofSpec Weiterentwicklung eine Weile gesichert sein. Das SoftCon Team hat sich aufgelöst. Der Teamleiter, Laszlo Fehervari, setzte sich mit seiner Frau und führenden Mitarbeiterin nach Wien ab. Er bekam dort eine Stelle bei einer Wiener Softwarehaus für das ich später Schulungen gemacht habe. Wir blieben privat weiterhin im Kontakt und im Jahre 2011 hielt ich seine Grabrede im Zentralfriedhof von Wien.

Mitten in diesem Auflösungsprozess kam unser Kunde – Krupp-Atlas – an und wollte die SoftOrg Werkzeuge eventuell kaufen. Krupp-Atlas in Essen war der Hauptlieferant der Bundesbahn für Anwendungssoftware. Sie war auch ein wichtiger Partner der Bundeswehr. Die Bundeswehr war gerade dabei eine Software Entwicklungsumgebung zu beschaffen. Sie hatte schon mehrere der damaligen Entwicklungs-

systeme untersucht und war auf SoftOrg gekommen. Das verantwortliche Team verfasste sogar ein Buch über die Ergebnisse ihrer Untersuchung – „Tools für die Softwareentwicklung“ im Springer Verlag, in dem SoftOrg recht gut abgeschnitten hat. [Bröh90]. Das Beschaffungsteam war geneigt SoftOrg auszuprobieren und hatte zu diesem Zweck schon mit Herrn Schaefer bei der Bahn gesprochen. Sie dürfte aber natürlich keine Software aus einem sozialistischen Land einsetzen. Wäre Krupp Besitzer der Werkzeuge gewesen, hätte sie keine Probleme gehabt.

Das Management bei Krupp-Atlas war daran interessiert, diesen Wunsch zu erfüllen. Die Geschäftsleitung hatte vor ein Entwicklungszentrum in West-Berlin aufbauen und dieses von dem Berliner Förderungsfund finanzieren zu lassen. Die ungarischen Entwickler würden dorthin versetzt werden. Deutsche Mitarbeiter sollten mit ihnen zusammenarbeiten und später die Weiterentwicklung übernehmen. Ich sollte sie in der Anfangszeit beraten und dann weiterziehen. Auch die Krupp Manager haben die starke Abhängigkeit der Software von den Menschen die sie entwickelt haben, nicht richtig erkannt. Es ist gar nicht so einfach eine Softwaremannschaft auszutauschen und den geistigen Führer der Mannschaft erst recht nicht. Den Wert einer Software steckt tatsächlich in den Köpfen ihrer Entwickler. Wer Software als Gut besitzen möchte, muss die Menschen besitzen welche die Software beherrschen [Tock05].

Der Geschäftsführer von Krupp Atlas war mit einer Delegation in Budapest und hat mit der Außenhandelsgesellschaft und Vertreter der beiden Partnerinstitute Gespräche geführt. Ich habe diese Gespräche nur von der Ferne verfolgt, da ich eigentlich gar nicht mitreden dürfte. Die Rechte an den Werkzeugen hatte ich bereits 1980 an die Außenhandelsgesellschaft abgetreten. Meine einzige zukünftige Rolle wäre als Berater zu der Entwicklungsstelle in Berlin gewesen. Sonst bliebe ich bei diesen Gesprächen draußen vor. Es wäre möglicherweise zu einer Übernahme der Produkte gekommen, wenn auch nicht die DDR zusammengebrochen wäre. Im Moment, als die Mauer fiel. fiel auch die Berlin Förderung für die Betriebe dort. Krupp-Atlas konnte nicht mehr mit einer Subvention rechnen. Als Folge sind die Gespräche erfolglos abgebrochen wurden. Ich erinnere mich nur an den Worten des Krupp-Atlas Geschäftsführers als es darum ging, welche Rolle ich in dem neuen Entwicklungszentrum spielen sollte. Er meinte, Krupp könne keine bunten Papageien in ihre Reihen dulden. Ich weiß immer noch nicht, ob ich diese Aussage als Kompliment oder als Herabwürdigung auslegen sollte.

Parallel zu den Gesprächen mit Krupp-Atlas liefen auch Gespräche mit einem holländischen Softwarehaus - Rendek. Die Holländer hatten über einen gemeinsamen deutschen Kunden von den Werkzeugen erfahren und waren daran interessiert die

Tools für die Qualitätssicherung zu übernehmen. Die Teams von der SZAMOK Institut sind in Amsterdam gewesen und haben die Tools SoftDoc, SofTest und SoftInt zur Probe installiert. Die Prüf- und Testwerkzeuge hatten Ende der 80 Jahren immer noch 28 Anwender. Rendek war daran interessiert an diese Firmen heranzukommen. Sie versprach sich dadurch eine Belebung ihres Geschäftes. Als die holländischen Unterhändler bald erkannt haben in welchem desolaten Zustand sich ihr Partner befand haben sie die Verhandlungen eingestellt. Also blieben die Ungarn auf den Werkzeugen sitzen, sie konnten sie weder verkaufen noch allein weiterentwickeln.

6.5 Das Ende der Mainframe-Entwicklung

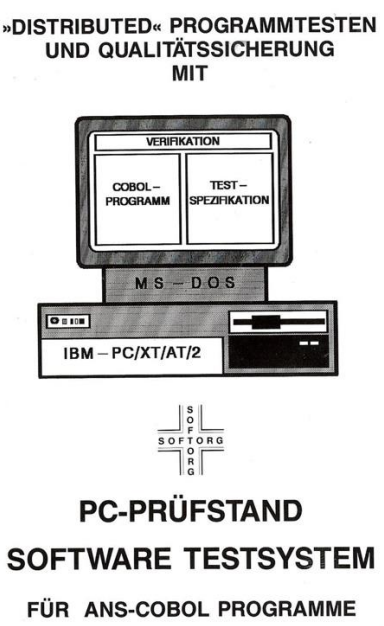
Es kam noch eine weitere Aspekt dazu. Die Werkzeuge liefen auf dem falschen Trägersystem. Als wir 1980 begonnen haben die Softorg Werkzeuge zu entwickeln haben die Anwender ihre Anwendungssoftware auf dem Rechner entwickelt auf dem sie später laufen soll und das war in den Großbetrieben der Mainframe. Ergo haben wir die Werkzeuge für diese Maschine gebaut. Es gab in diesem Bereich damals nur wenig Konkurrenz. Die Kosten der Mainframe-Toolentwicklung waren einfach zu hoch, nicht nur wegen der teuren Rechnernutzung sondern auch wegen der hohen Personalkosten. Kleine Toolentwickler ohne ausreichende Kapitaldeckung hatten auf diesem Markt keine Chance. Von einer Softwareentwicklung auf deduzierten Kleinrechner war nur in den Forschungslabors die Rede.

Dann kam der PC. Auf einem Seminar in Portofino, Italien, bei dem ich eingeladen war über Testen zu sprechen, habe ich schon 1980 vom Rank-Xerox Projekt im fernen California erfahren, aber das erschien mir auch Lichtjahren entfernt zu sein. Es ging darum objekt-orientierter Software auf einem speziellen Entwicklungsrechner mit graphischen Oberflächen zu entwickeln. Einige Jahre später kam der erste PC-Rechner auf dem Markt. Es dauerte nicht lange bis die ersten amerikanischen Softwarehäuser dafür Tools gebaut haben um Anwendungssysteme zu konzipieren und zu entwickeln. Der große Durchbruch kam mit dem MicroFocus COBOL Compiler. Damit konnte man COBOL Programme, die für den Mainframe gedacht waren auf einem PC-Rechner bis zur Compilierung bringen. Die Analyse und der Entwurf der Programme konnten mit graphischen Mittel unterstützt werden. Das war auf dem Mainframe nicht denkbar. Diese neue Generation von Entwicklungswerkzeugen wurden als CASE – Computer-aided Software Engineering bezeichnet. Ed Yourdon, der damals neben James Martin der größte Guru der Softwarewelt war, predigte dass diese Tools den Anwendungsmarkt in kurzer Zeit erobern würden [Your92]. Er wurde in dieser Überzeugung von etlichen anderen unterstützt [Rock92].

Die neuen CASE Tools aus den USA liefen alle auf PC-Arbeitsplätzen. Der neue Trend war auf PC-Arbeitsplätzen zu entwickeln und die generierten Programme erst nach der Kompilierung auf den Host zu übertragen. Die Entwicklung auf dem Mainframe mit SPF/TSO geriet in den Verruf, altmodisch und umständlich zu sein. 1990 kam IBM mit ihrer neuen Entwicklungsumgebung AD/Cycle auf den Markt, die auf eine verteilte Entwicklung setzte – am PC entwickeln, auf dem Host integrieren und testen [Mart91]. Die SoftOrg Mainframe Werkzeuge waren in Anbetracht der neuen CASE Tools technologisch abgehängt. Mit unserer hostgestützten Entwicklung ritten wir sozusagen auf einem toten Pferd.

6.6 Der Versuch auf den PC umzusteigen

Es hätte eine riesen Investition vorausgesetzt die Werkzeuge technologisch zu erneuern. Dennoch hatte ich in einzelnen Fällen schon damit begonnen. Sobald wie der Microfocus COBOL Compiler für den PC 1988 heraus kam, begann ich ein Projekt zusammen mit der ungarischen Akademie der Wissenschaft, ein Tool für das Testen von COBOL Programme auf dem PC zu entwickeln. Das neue Tool hieß MicroTest und sollte das Erste in einer Reihe von PC-Entwicklungswerkzeuge sein. Das Entwicklungsteam bestand aus sehr hoch qualifizierten Akademikern, die mit Compilerbau vertraut waren. Prof. Tibor Gyimothy von der Universität Szeged hat den COBOL Parser beigesteuert.



MicroTest war die Krönung in einer langen Reihe von Testwerkzeugen, die ich bis dahin konzipiert hatte. Die Kernidee war der Test eines Programmes gegen ein Anderes. In diesem Fall war das andere Programm ein Pascal Programm, das aus der Testspezifikation generiert wurde. Der Tester beschrieb die Vor- und Nachbedingungen in einer Testskript und lies die Skript in ein Pascal Programm umsetzen. Das Pascal Programm wurde neben dem Testobjekt, das COBOL Programm, ausgeführt und überprüfte alles was das COBOL Programm tat. Jede geringe Abweichung zwischen den beiden Programmen wurde registriert und dokumentiert. Außerdem wurde der COBOL Code instrumentiert um die Testüberdeckung aller Zweige und Pfade sowie aller Datennutzungen zu messen [Sned94].

Das war für die damalige Zeit ein großer Schritt in der Testtechnologie. Jetzt konnte der Anwender Programme für den Mainframe auf dem PC nicht nur entwickeln und kompilieren, er konnte sie dort auch austesten. Auf einer Tooltagung in London hat das MicroTest den ersten Preis gewonnen. Im Rückblick war dies eine meiner größten technischen Errungenschaften. Das Problem mit diesem Tool war,

dass es von Wissenschaftlern für Wissenschaftler konzipiert war. Der durchschnittliche deutsche COBOL Entwickler war damit überfordert. Die exakte Spezifikation des Programmverhaltens mit einer Prädikatenlogik erster Ordnung war zu anspruchsvoll. Es war ein wunderbares Tool um auf Konferenzen vorzuzeigen, es hatte alle Eigenschaften die man von einem Modultestwerkzeug erwarten konnte, aber gerade deshalb war es für die COBOL Programmierpraxis weniger geeignet. Microfocus ist kurz danach mit einem eigenen COBOL Testwerkzeug für den PC herausgekommen, das viel einfacher war. Es hat mit den originalen Daten von dem Mainframe gearbeitet. Der Tester brauchte dafür keine Skripte schreiben.

Vielleicht hätte MicroTest langfristig eine Zukunft gehabt wäre die Entwicklung weiter gegangen. Aber auch die Akademie der Wissenschaften ist in den Zog des allgemeinen Systemverfalls geraten. Viele Forschungsprojekte sind eingestellt worden, u.A. auch das MicroTest Projekt. Von den drei Teammitgliedern ist nur eins in der Akademie geblieben. Die anderen zwei einschließlich der Teamleiter haben sich selbstständig gemacht. Mit dem Teamleiter – Endre Somos – hatte ich später in dem Wella Projekt zu tun gehabt. Er hat einen COBOL-Instrumentor für das AS400 Testsystem geliefert. Mit dem Abbau der Akademie ist die MicroTest Entwicklung eingestellt worden.

Parallel zur Entwicklung des MicroTests in der Akademie der Wissenschaften lief in dem SzKI Institut ein Projekt zur Übertragung des SoftSpec Mainframe Tools auf den PC. Erika Nyary hat mit ihrem Team begonnen ein neues SoftSpec mit einer graphischen Oberfläche in C zu implementieren. Das neue Tool hieß PC-Spec. So wie wir zu Beginn mit der Mainframe-Entwicklungsumgebung angefangen hatten, begannen wir nun auf dem PC mit dem Anfang und Ende des Software Entwicklungszyklus - Spezifikation und Test. Die beiden Tools PC-Spec und PC-Test sollten die Grundlage für eine neue Werkzeugfamilie auf dem PC sein. Mit PC-Spec kamen die Anwender gut zurecht. Es fiel ihnen leichter die Datenbanken, Oberflächen, Berichte und Vorgänge zu modellieren. Bei einigen SoftSpec Kunden, z.B. bei der Bahn, war PC-Spec Ende 1989 bereits produktiv im Einsatz. Dann folgte der Zusammenbruch und das eins stolze SzKI Institut wurde abgewickelt. Das Team von Erika Nyary ist ausgeschieden und hat eine eigene Firma gegründet. Der Teamleiter vom Softman ist nach Zürich ausgewandert. Die verbleibenden Teammitglieder sind zu der Firma von Frau Nyary – Softing – gestoßen. Ich habe sie im Rahmen des Esprit Forschungsprojekt für Softwaremetrik – METKIT - eine Weile beschäftigen können. Ich hätte die Weiterentwicklung gerne weiter betrieben aber mir fehlte das Kapital dafür. Als Folge der Auflösung der Institute hatte ich genug Probleme allein den Status Quo zu erhalten. Vor einer Modernisierung der Tools konnte nicht die Rede sein.

Unstrukturierte Programm-Metriken

Metrik		Originalversion	Korrigierte Version	Adaptierte Version	Erweiterte Version
K1)	Modulkomplexität	0,86	0,86	0,87	0,88
K2)	Graphkomplexität	29	29	34	37
K3)	Datenkomplexität	1,87	1,87	1,87	1,98
K4)	Sprachkomplexität	0,033	0,032	0,032	0,026
K5)	Testkomplexität	0,409	0,409	0,422	0,437

Restrukturierte Programm-Metriken

Metrik		Originalversion	Korrigierte Version	Adaptierte Version	Erweiterte Version
K1)	Modulkomplexität	0,33	0,39	0,39	0,42
K2)	Graphkomplexität	25	26	26	31
K3)	Datenkomplexität	1,83	1,84	1,86	1,88
K4)	Sprachkomplexität	0,031	0,030	0,029	0,024
K5)	Testkomplexität	0,360	0,369	0,371	0,368

Reengineered Programm-Metriken

Metrik		Originalversion	Korrigierte Version	Adaptierte Version	Erweiterte Version
K1)	Modulkomplexität	0,38	0,40	0,43	0,44
K2)	Graphkomplexität	19	21	21	24
K3)	Datenkomplexität	1,66	1,66	1,70	1,79
K4)	Sprachkomplexität	0,044	0,042	0,039	0,031
K5)	Testkomplexität	0,354	0,339	0,333	0,353

6.7 Die Abwicklung des SoftOrg Geschäfts

Ab 1990 habe ich begonnen die Kunden-Wartungsaufträge zu kündigen. Bei den Kunden auf der Entwicklungsseite bot ich an, ihre Daten in eine Repository ihrer Wahl zu übertragen. Die generierten Programme konnten sie ohnehin selber warten. Ich habe vom Anfang an Wert darauf gelegt, dass die Programme so aussehen als ob ein Mensch sie geschrieben hätte. Sie waren modular, strukturiert, formatiert und gut kommentiert. Ein Entwickler mit Kenntnissen der strukturierten Programmierung dürfte mit ihnen ohne weiteres zurechtkommen. Mit SoftDoc konnte man den Code auch automatisch nachdokumentieren.

Der letzte große Anwender war die Bundesbahn. An meinem 50. Geburtstag im März 1990 habe ich mit dem Anwenderteam der Bahn das Ende der SoftOrg Entwicklung in einem herabgekommener Kneipe hinter dem Frankfurter Bahnhof gefeiert, dort wo wir zwei Jahre zuvor den größten Erfolg des ersten SoftOrg Projektes gefeiert hatten. So schnell hat sich die Lage geändert. Mir war zum Feiern nicht zumute, aber in Deutschland muss man auf die Riten achten. Alles wird gefeiert, der Erfolg wie auch die Niederlage. Es war nur bezeichnend, dass das Ende der SoftOrg

Entwicklung mit meinem 50. Geburtstag zusammenfiel. Das Ereignis markierte in jeder Hinsicht ein Wendepunkt in meinem Leben.

6.8 Die EU-Forschungsprojekte

Parallel zu den letzten SoftOrg Projekten lief am Ende der 80er Jahren eine andere etwas erfreulicher Entwicklung an. Meine Firma – die SES GmbH – wurde von der EU im Rahmen des Esprit Forschungsprogrammes zur Förderung der europäischen IT-Industrie finanziell unterstützt. Dafür mussten wir angewandte Forschung auf dem Gebiet der Software-Engineering betreiben und zwar auf drei verschiedenen Teilgebieten:

- Softwaretest
- Softwaremessung und
- Software Reverse Engineering.

6.8.1 Das TRUST Testforschungsprojekt

Aufgrund meiner Veröffentlichungen bei der IEEE Computer Society und der ACM wurde ich von der EU-Kommission als Gutachter für das erste große Esprit Forschungsprojekt REQUEST für Software Qualitätssicherung gerufen. Das war bereits im Jahr 1986. Ich musste im Rahmen dieser Tätigkeit die Qualitätssicherungsvorschläge der Partnerfirmen begutachten. Dazu gehörten solche renommierte Firmen wie Thompson in Frankreich, ICL in England, Olivetti in Italien und Siemens in Deutschland. Später habe ich ein eigenes Forschungsprojekt für den Test eingebetteter Software bekommen. Dieses Projekt habe ich im letzten Kapitel schon geschildert. Es begann im Jahre 1987 und dauerte bis Ende 1989. Wir haben neben den zwei Werkzeugen ADATest und TestSpec etliche Papers zum Thema Test geliefert. Die Zusammenarbeit mit dem Team von Prof. Hennell in Liverpool war sehr lehrreich. Es ging darum die Restfehlerwahrscheinlichkeit aufgrund der Testüberdeckung und der bisher gefundenen Fehler hochzurechnen. Ich hatte eine Idee wie wir das machen konnten – Sneed's Conjecture – das die Restfehleranzahl der natürliche Logarithmus von der Codeüberdeckung und der gefundenen Fehleranzahl sei, aber leider konnten wir diese These nie richtig nachweisen [Mars89]. Heute wissen wir, dass Codeüberdeckung nicht ausreicht um Fehlerfreiheit zu gewährleisten.

6.8.2 Das METKIT Metrik-Forschungsprojekt

Auf das TRUST Projekt folgte 1989 ein zweites Forschungsprojekt – METKIT – für die Einführung von Messtechnik in der europäischen Software Industrie. Der Mauer war gerade gefallen und ein deutscher Software Ingenieur bei Zeiss in Jena hat mit mir Kontakt aufgenommen. Er hatte von mir viel gelesen und wollte mit mir zusam-

menarbeiten. Im Frühjahr 1990 habe ich ihn in Jena besucht und ihn als freie Mitarbeiter gewonnen. Er sollte dort für das METKIT Projekt Messwerkzeuge entwickeln, die wir auch an Industriekunden anbieten könnten. Das war der Anfang einer langjährigen Zusammenarbeit aus der eine Reihe Managementwerkzeuge hervorgingen – PC-Org für die Erfassung der betrieblichen Anwendungsprozesse, PC-Calc für die Projektaufwandschätzung, PC-Man für die Planung und Überwachung der Projekten und PC-Mess für die Sammlung und Auswertung der Produkt- und Prozess-Metrikdaten. Alle vier PC-Tools galten als unser Beitrag zum EU-Forschungsprojekt. PC-Calc und PC-Man haben wir später an einige Industriekunden vermittelt, u.a. Daimler Benz und Merz Chemie. Wir waren der einzige Partner in diesem Forschungskonsortium, der einen Industrienutzen vorweisen konnten. Wir haben die Werkzeuge auch als Lehrmittel in den Metrik-Lehrgängen verwenden. Dennoch wurde die SES zum Schluss von der Kommission bestraft weil wir unsere Ziele nicht erreicht hatten.

Außer Dr. Rothhardt in Jena als Toolentwickler ist mir auch gelungen Prof. Dr. Horst Zuse, Deutschlands bekanntester Softwaremetrik-Experte als wissenschaftlicher Berater für das Projekt zu gewinnen. Dr. Zuse hatte schon zwei Bücher zu diesem Thema verfasst und genoss einen internationalen Ruf als Metrik-Fachmann. Für das METKIT Projekt hat er mehrere Kurse entwickelt und eine Studie über die Anwendung der Softwaremetrik in der Industrie durchgeführt. Mit Professorin Agnes Kapocsi von der Londoner South Bank University und Dr. Zuse von dem TU-Berlin haben wir ein umfangreiches Experiment in der Auswirkung der Codekomplexität auf den Wartungsaufwand durchgeführt. Im Rahmen dieser kontrollierten empirischen Studie haben wir die originale, die automatisch restrukturierte und die manuell reimplementierte Version eines alten COBOL Programmes korrigiert, geändert und erweitert, um die dazu erforderlichen Aufwände zu vergleichen. Erwartungsgemäß gab es eine schwache Korrelation zwischen Komplexität und Aufwand bei der Fehlerkorrektur und eine starke Korrelation bei der Erweiterung. Was besonders auffiel war, wie die Komplexität bei jedem Eingriff in den Code zunimmt. Über die Ereignisse dieses Laborexperiments haben wir auf der internationalen Metrik Tagung in Chicago sowie auf der internationalen Maintenance Konferenz in San Diego berichtet [SnKa90].

Das METKIT Projekt hat mich inspiriert mich näher mit dem Thema Softwaremessung zu befassen. Es folgten mehrere Veröffentlichungen zu dem Thema, auch in deutscher Sprache [Sned91]. Zusammen mit Dr. Rothhardt habe ich einige Kurse über Softwaremessung entwickelt und als öffentliche Seminare angeboten [SnRo96]. Auf dieser Weise gewannen wir auch die ersten Messungskunden, Firmen die ihre Soft-

waresysteme messen wollten. Man könnte mit Recht behaupten, die Firma SES hat die Software-Messtechnik in Deutschland eingeführt, zumindest was die praktische Anwendung anbetrifft. Insofern haben wir das Ziel des METKIT Projektes erfüllt. Später fasste ich die Erkenntnisse aus dem METKIT Projekt in einem Beitrag für das Journal of Software Maintenance zusammen [Sned95].

Wir waren die einzige Partner in dem METKIT Konsortium, der einen praktischen Erfolg vorweisen konnte. Neben der Schulungsunterlagen, den Fachartikeln und dem Laborexperiment haben wir vier industriereife Messwerkzeuge produziert. Damit haben wir allerdings über das Ziel hinweggeschossen. Man darf laut den EU-Richtlinien keinen direkten wirtschaftlichen Nutzen aus der EU-finanzierte Forschung ziehen. Die Kommission wollte von meiner Firma € 55.000 zurück haben. Angeblich hatten wir nicht alle wissenschaftliche Abnahmekriterien erfüllt um die Fördergelder in Anspruch zu nehmen. Diese Rückforderung habe ich ohne Erfolg angefochten. So blieb diese Schuld noch viele Jahre danach wie ein Fallbeil über meinem Kopf.

6.8.3 Das DOCKET Reverse Engineering Forschungsprojekt

Erstaunlicherweise bekam ich trotzdem in dieser Zeit ein weiteres Esprit Forschungsprojekt – DOCKET – für die Reverse Engineering bestehender IT-Systeme. Die Projektleitung hatte die Technische Universität von Manchester. Der leitende Professor dort – Paul Lazell - hatte ein globales Unternehmensmodell konzipiert und wollte, dass die Projektpartner dieses Modell mit Daten aus einem bestehenden IT System für die Verwaltung der Studentenwohnheime füllen. Die Engländer sollten Daten aus Gesprächen mit den Anwendern – Expertenbefragung - beziehen, die Griechen sollten Daten aus der Analyse der Anforderungstexte gewinnen durch Natural Language Processing, die Holländer sollten Daten aus den Entwurfsdokumenten entnehmen, wir sollten zusammen mit dem Italienischen Forschungsinstitut CRIAI in Portici südlich von Neapel Anwendungswissen aus dem Source Code sowie aus den Testfällen gewinnen. Wir hatten schon etliche Werkzeuge zu diesem Zweck, wir mussten sie nur anpassen und mit dem Anwendungsbeispiel – das Studenten Wohnheim-Managementsystem – anwenden. Ich hatte das Glück mit einer wunderschönen italienischen Forscherin von dem CRIAI Instituts zusammenzuarbeiten. Dies hatte zu Folge dass ich zu viel Zeit in Portici verbrachte, Zeit, die ich für die Abwicklung meiner Projekte in Deutschland dringend nötig hatte. Anderseits brauchte ich diese Ablenkung um die Folgen meines geschäftlichen Zusammenbruchs in Deutschland zu überwinden.

Unser Teilprojekt war jedenfalls ein Erfolg. Wir haben mit Hilfe der statischen und dynamischen Analyse ein Entwurfsmodell aus dem Source-Code gewonnen.

Darüber habe ich auf dem ersten IEEE Workshop für Reverse Engineering berichtet [Sned93]. Wir hatten zur gleichen Zweit ein ähnliches Reverse Engineering Projekt für die UBS in Zürich. Das Team von Erika Nyary hat dort den Natural Source Code analysiert und daraus Daten für eine Repository gewonnen, aus der sie anschließend mehrere graphische Dokumente produzierten, Dokumente die ich heute noch in meinem Lehrgang über Reverse Engineering verwende. Jedes einzelne Team im Docket Projekt brachte ein Ergebnis. Das Problem war, dass die Ergebnisse nicht zusammenpassten. Was aus dem Source Code herauskam, war ganz anders als das, was aus den Anforderungsdokumenten herauskam und das war wiederum was Anderes, als das was aus dem Interviews mit dem Benutzer hervor ging. Jede semantische Ebene der Software hatte eine eigene Sicht auf das System und die Sichten waren nicht konsistent. Das lag daran, dass jede Ebene eine andere Ontologie hatte, bzw. andere Begriffe verwendete. Es gab keine Möglichkeit, die Sichten zusammenzuführen ohne die Begriffe anzupassen. Dies wäre wiederum zu Aufwendig gewesen, also wurde das Projekt abgebrochen. Aus dem „Global Model“ wurde nichts. Es blieb vom Projekt nichts übrig als einige Erinnerungen an schöne Abende am Meer in Portici.

6.9 Vom Praktiker zum Forscher

Die drei ESPRIT Projekte dauerte vom 1987 bis 1993, gerade in der Zeit des großen Umbruchs. Sie waren für mich die Rettung. Mit dem Fördergeld konnte ich die SoftOrg Entwicklung in Ungarn abwickeln und einen neuen Anfang in der Schweiz als Software ReEngineer beginnen. Ich hatte erkannt dass eine große geschäftliche Möglichkeit in der Aufarbeitung der bestehenden Altsoftware lag. Die Herausforderung der Zukunft war die Bewältigung der Vergangenheit [Sned87]. Dank der Unterstützung der EU konnte ich diesen neuen Geschäftszweig aufbauen. Diese Unterstützung half mir auch den Übergang von dem Mainframe zur PC zu schaffen. Ab 1990 wurden alle neuen Werkzeuge für den PC gemacht mit Windows als Trägersystem. Um möglichst viel von dem alten Werkzeugcode wiederzuverwenden haben wir den MicroFocus COBOL Compiler auf dem PC-Arbeitsplatz eingesetzt und viele alte Mainframe-Cobol Module hinter einer graphischen Oberfläche gekapselt. Nur ganz neue Testwerkzeuge wurden in C++ implementiert. Die Benutzeroberfläche wurde mit Delphi von Borland programmiert. So entstand in der Schweiz bei der UBS sowie in Jena mit dem ehemaligen Zeiss Mitarbeiter eine neue Toolgeneration von der ich noch lange leben sollte. Das war die Erbe der Softorg Entwicklung.

Durch die EU-Forschungsprojekte kam ich weiterhin dazu, zahlreiche neue Papers zu verfassen und auf internationalen Konferenzen zu präsentieren. Im Jahre 1991 habe ich ein Paper zum Thema „Reengineering Economics“ in dem neuen Journal für

Software Maintenance veröffentlicht [Sned91]. Daraufhin wurde ich von einer Arbeitsgruppe im U.S. Defense Department aufgefordert, an der Verfassung ihrer Reengineering Richtlinie mitzuwirken und im Jahr 1996 wurde ich von der IEEE Computer Society auf der ICSM Tagung in Monterey für meine Pionierleistungen auf dem Gebiet der Software Reverse- und Reengineering ausgezeichnet. In der Zeit von 1985 bis 1990 habe ich fünf neue Bücher im Rudolf-Müller Verlag herausgebracht, in 1986 Softwareentwicklung, in 1987 Softwaremanagement, in 1988 Softwarequalitätssicherung, in 1989 Softwarewartung und in 1990 Softwaresanierung. Ich wollte damit alle Software-Lebenszyklusphasen abdecken und meine Ansätze dazu dokumentieren. Die europäischen Förderprojekte haben mir sehr geholfen in der Welt der Wissenschaft Fuß zu fassen und später eine Karriere als Hochschullehrer zu beginnen. Komischerweise begann diese Karriere unweit von Portici an der Universität Benevento. Die alten italienischen Beziehungen sind mir doch noch zu Gute gekommen. Vom sozialistischen Ungarn blieb mir nur eine Auszeichnung als „Kiváló dolgozó“ – Held der Arbeit –übrig. Die SoftOrg Werkzeuge, in die so viel Zeit, Geld und Energie aus meinem besten Jahren eingeflossen ist, wurden auf Magnetbändern gesichert und lagern heute noch in dem Dachkammer meines Hauses in Budapest. Teile des Source Codes wurden in den neuen PC-Werkzeugen übernommen und funktionieren weiter als gekapselte Bausteine in einer DOS-Box. Software Systeme sind vergänglich, der Code selbst ist unsterblich. Er lebt so lange als es Maschinen gibt, die ihn ausführen und Menschen gibt, die ihn verändern können.

7 Als Reengineer in der Schweiz

7.1 Ein Anruf aus der Schweiz

Im August 1989 wurde mir klar, dass es mit der Softorg Entwicklung nicht weiter gehen kann. Das sozialistische System war in Auflösung In diesem Monat wurde die Grenze zur Österreich geöffnet, die finanzielle Unterstützung der staatlichen Institute eingestellt und der Weg für die Gründung privater Gesellschaften innerhalb der bestehenden Institutionen freigemacht. In dem Szamalk Institut waren meine Mitarbeiter schon dabei, eigene Firmen zu etablieren. In den SzKI Institut wurde dies zur Diskussion gestellt. Ich habe eingesehen, dass ich die Finanzierung der weiteren Entwicklung der Softorg Werkzeuge selber übernehmen muss. Dazu war ich jedoch kaum in der Lage. Die monatlichen Wartungsgebühren deckten nur ein Bruchteil der Entwicklungskosten ab. Die SES hatte zum diesen Zeitpunkt zur Glück zwei EU-Forschungsprojekte, so dass wir aus Brüssel eine finanzielle Unterstützung hatten, wir müssten davon aber eigene Mitarbeiter bezahlen. In dem Testprojekt hatten wir zwei deutsche Mitarbeiter und in dem Metrikprojekt zwei – Dr. Zuse von der TU Berlin und Dr. Rothhardt von Jena. Für die Ungarn blieben kaum Gelder übrig. Größere Projekte waren nicht in Sicht – das Thyssen-Stahl Projekt war abgeschlossen, ebenso das Continental Reifen Projekt, Bertelsmann fiel aus, nur bei der Bundesbahn und der Bremer Lagerhausgesellschaft lief es weiter. In der Bahn war der Herr Schäfer mit einem deutschen Mitarbeiter und in Bremen der Herr Kroeger mit einem weiteren deutschen Mitarbeiter beschäftigt. In Bremen gab es noch ein Projekt an dem die Ungarn beteiligt waren.

Allerdings waren die wenigen Projekte nicht ausreichend um die Toolentwicklung zu finanzieren. Es war mir klar, dass ich einen anderen Weg einschlagen musste. Aber welchen? Mit der Finanzierung der Softorg-Entwicklung durch die ungarische Außenhandelsgesellschaft konnte ich nicht mehr rechnen. Sie war selbst dabei abgewickelt zu werden. Außerdem war der Mainframe nicht mehr der geeigneten Plattform für Entwicklungswerkzeuge. Auf dem Markt setzten sich die neuen PC-Rechner immer mehr durch. Die neuen CASE Werkzeuge von Knowledgeware und IEF hatten alle den PC als Trägersystem. Die amerikanischen Anwender hatten angefangen ihre Software bis zum Source Code auf dem PC-Arbeitsplatz zu entwickeln, um anschließend den Source Code zwecks Integration und Test auf den Mainframe zu übertragen.

In Deutschland bot SoftLab mit PET-Maestro bereits eine ähnliche Lösung an [Oest85].

Diese Entwicklung hatten wir Ende der 80er Jahren begonnen nachzuvollziehen. Das Team von Erika Nyary hatte schon eine PC-Version von SofSpec entwickelt. Der erste Prototyp PC-Spec war bereits bei der Bundesbahn im Einsatz. Die Weiterentwicklung konnte jedoch von der SzKI nicht finanziert werden. Das Institut war in der Auflösung und das Team von Frau Nyary wollte eine eigene Firma gründen. Ich musste die Finanzierung der Weiterentwicklung selber übernehmen.

In dem Szamalk Institut sah es ähnlich aus. Das Test Team von Frau Erdös drängte mich, sie direkt als freie Mitarbeiter zu beschäftigen um die Entwicklung der Testwerkzeuge voranzutreiben. Das Entwicklungsteam für SoftDoc und SoftGen waren im Begriff einen eigenen Weg zu gehen. Gabor Jandrasics und Sari Elek hatten schon eine eigene Firma innerhalb des Instituts gegründet. Ich habe mich sogar daran beteiligt. Sie waren dabei das neueste Reengineering Werkzeug – SoftRecon zu entwickeln. Das Werbematerial war schon vorbereitet und verteilt. Ich warb dafür in dem SES Newsletter und in den öffentlichen Seminaren. Die Themen Reverse- und Reengineering waren im Kommen und ich hatte schon einen gewissen Vorsprung vor den anderen Anbieter auf diesem Gebiet. Ich brauchte nur dringend ein Vorzeigeprojekt. Es sollte bald kommen.

Im September saß ich im Kellerbüro der SES in Ottobrunn und brüte über die Zukunft der SES als ein Anruf aus Zürich kam. Es war der IT Leiter der Schweizerische Bankgesellschaft – der Herr O. Er bat mich sofort nach Zürich zu kommen über ein mögliches Projekt zu sprechen. Mein alter Freund und Konkurrent, Dr. Thurner hatte mich empfohlen. Der IT Leiter stellte mir eine merkwürdige Frage, ob ich mit schwierigen Frauen als Chef auskommen konnte. Diese Frage konnte ich zu dem Zeitpunkt gar nicht einordnen. Ich wusste nicht dass in der Schweiz zu diesem Zeitpunkt jede stark auftretende Frau al schwieriger Fall betrachtet wurde. Ich versicherte ihm, dass ich gewohnt war in Ungarn mit starken Frauen zusammenzuarbeiten.

Am nächsten Tag bin ich nach Zürich geflogen und wir einigten uns gleich auf eine Zusammenarbeit. Ich war bereit ein Testprojekt zu machen und einen Teil der Kosten selber zu tragen. Wenn alles gut ginge, würde mir die Bank meine vollen Kosten in Höhe von Sfr. 80.000 erstatten. Sollte es nicht gelingen, würde die Bank lediglich meine Unkosten in Höhe bis zu Sfr. 40.000 erstatten. Dadurch war meine Firma an dem Risiko beteiligt. Das gefiel ihm und ich bekam innerhalb weniger Tage einen Auftrag. Dies sollte das erste von vielen Reverse- und Reengineering Projekte für die

SBG sein. Ich habe mein Risiko an die ungarischen Partnerfirmen weitergegeben. Sie sollten wie früher die Institute 75% von dem erhalten was ich vom Kunden bekomme. Ihre Reisekosten mussten sie selber decken. Ich habe ihnen lediglich ein Darlehen für den Anfang gewährt [Sned91].

7.2 Das Promega Migrationsprojekt

7.2.1 Promega Projekthintergrund

Promega war die Bezeichnung für eine lose Sammlung alter Batchysteme in dem SBG. Heute würde man solche Systeme als Backend bezeichnen. Das Frontend für diese Systeme wurde schon längst auf dem UNISYS Rechner in der neuen COBOL/DELTA/JSP Umgebung neuentwickelt. Aus welchen Gründen auch immer, man ließ die Batchprogramme auf einen uralten UNIVAC Rechner aus den 70er Jahren und übertrug die Daten von dem Unisys Rechner auf dem Univac Rechner mit Bändern. Dort wurden die Daten weiterbearbeitet und zahlreiche statische Berichte erstellt. Man hat den alten Rechner praktisch als statistische Auswertungsmaschine verwendet.

In Promega waren 12 verschiedene Applikationssysteme mit unterschiedlicher Größenordnung zusammengefasst. Es gab in diesen Systemen circa 300 einzelne Programme und über 800 sequentielle und index-sequentielle Dateien. Performanzkritische Programme waren in der Assembler Sprache SPURT implementiert. Die meisten Programme waren in COBOL-68 mit Hilfe von einem Entscheidungstabellengenerator namens „DETCAB“ implementiert. Über die Jahre sind die Programme immer größer geworden, manche enthielten mehr als 10.000 Zeilen. Gesteuert wurden die Programme von über 500 Job Control Prozeduren in einer uralten UNIVAC Job-Control-Sprache [Sned91a].

Das Besondere an dem UNIVAC Rechner war seine Speicherstruktur. Die Daten waren in 30 Bit Wörtern gespeichert. Ein Byte bzw. ein Ziffer wurde mit 6 Bit dargestellt. Der neuere Unisys Rechner hatte 36-Bit Wörter mit 6 x 6 Bit Zeichen. D.h. die Daten mussten immer physisch übersetzt werden, um sie von dem einen Rechner auf den anderen zu übertragen. Das war umständlich und hat viel Zeit gekostet. das Betriebssystem für den Univac Rechner hieß OMEGA und wurde von der Firma Unisys nicht länger gepflegt. Unisys hatte auch der Support für die Univac Hardware aufgekündigt. Folge dessen wurde die Promega Umgebung auf den neuen Unisys-1100 Rechner emuliert, was zu irre lange Laufzeiten führte. Es war demnach höchste Zeit, das alte System abzulösen.

Hinzu kam, dass acht Wartungsprogrammierer, meistens ältere Mitarbeiter mit der Erhaltung der Promega Systeme beschäftigt waren. Junge Entwickler in der Bank wurden mit den Sprachen COBOL-74 und DELTA sowie mit der Jackson Strukturierten Programmierung- JSP – ausgebildet. COBOL-74 war die Hauptsprache, DELTA war die Sprache für die Verbindung mit der Umgebung bzw. die Ein- und Ausgabe Operationen – die TP Operationen und Datenbank Zugriffe. Die JSP diente der Strukturierung der Abläufe mit den normierten Konstrukten – Sequenz, Auswahl und Wiederholung. Die Kluft zwischen dieser neuen Programmiertechnik und der GOTO gesteuerten Ablauflogik der alten Promega Programme war enorm. Es war klar, dass das junge Personal für die Weiterentwicklung der Promega Programme nicht in Frage kam.

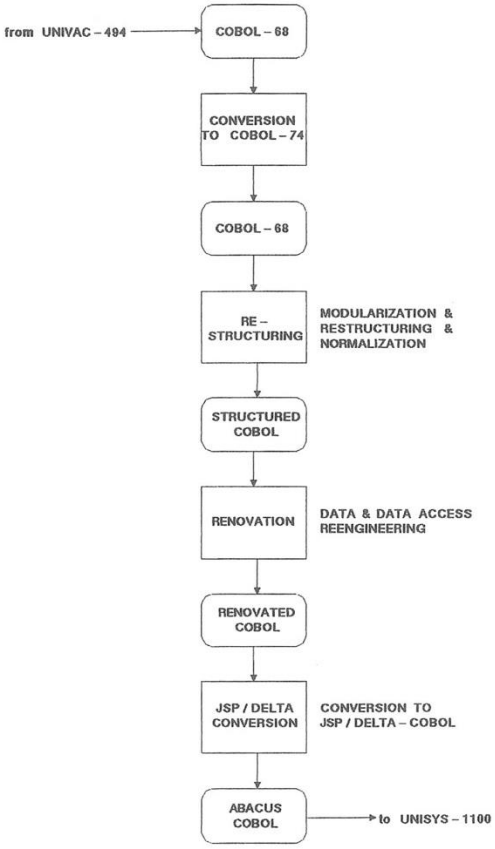


Figure 2: COBOL Transformation Process

Der IT Leiter der SBG stand vor der Entscheidung, die alten Batchprogramme in der neuen Umgebung neu zu entwickeln oder sie auf irgendeiner Weise automatisch zu übertragen. Die Neuentwicklung hätte viel gekostet und lange gedauert. Einzelne geschäftskritische Applikationen wurden trotzdem neu entwickelt. Der Nutzen war groß genug, um die hohen Kosten auszugleichen. Das Gros der alten Applikationen sollten jedoch automatisch konvertiert werden. So ging er auf die Suche nach einer Softwarefirma, die dazu in der Lage wäre. Dr. Thurber, Vater der DELTA Sprache

und Hoflieferant der Bank empfahl mich. So kam es, dass der IT Leiter mich anrief und nach Zürich einlud.

7.2.2 Promega Projektziele

Die Ziele des Promega Migrationsprojektes waren fünffach:

- Erstens ging es um die Konversion sieben der 12 Promega Applikationen, einschließlich deren Daten und Jobsteuerung. Die verbliebenen 5 Systeme sollten entweder neu entwickelt oder weggeworfen werden.
- Zweitens sollten die konvertierten Programme den Standards der neuen JSP/DELTA Entwicklungsumgebung angepasst werden, d.h. die COBOL IO-Operationen durch DELTA Makros ersetzen, die eingebetteten Satzbeschreibungen in Copy Members ausgliedern und eine zentrale Steuerung der Ablauflogik einbauen.
- Drittens müssten sämtliche SPURT Assembler Programme – insgesamt 27 Hauptprogramme und 123 Subroutinen in DELTA/COBOL umgesetzt werden.
- Viertens sollten alle 800 Dateien restrukturiert und in DMS-1100 Datenbanksätze umgewandelt werden.
- Fünftens waren die über 400 OMEGA Job Control Prozeduren in die DS-1100 Job Control Sprache umzusetzen.

Der Erfolg des Projektes wurde daran gemessen, zu welchem Grade alle fünf Ziele erfüllt wurden. Es genügte nicht nur einen Teil der Ziele zu erfüllen. Sie müssten alle erfüllt werden, und zwar in der vorgesehenen Zeit von 18 Monaten. Nachher sollte der alte UNIVAC Rechner verschrottet werden.

7.2.3 Promega Projektorganisation

Das Projekt begann mit einem Pilotprojekt, ein sogenanntes „Proof of Concept“ Projekt. Dieses fand im Herbst 1989 statt. Mir wurde das kleinste Teilsystem mit nur 7 Programmen gegeben und ich hatte drei Monate um es zu konvertieren – einschließlich Daten- und Jobsteuerung aber ohne Assembler. Das Pilotprojekt galt als Test meiner SoftRecon Reengineering Werkzeuge, ob sie wirklich das bringen, was ich versprochen habe. Ein Team von fünf Ungarn, die nach der Auflösung der Institute ihre eigene Firmen gegründet hatten, kamen mit mir nach Zürich und wir haben die drei Monate bis Jahresende Tag und Nacht gearbeitet, um die Werkzeuge auf Vordermann zu bringen und die Ziele der Probekonversion zu erreichen.

Bis Ende Juni 1990 wurden alle Migrationswerkzeuge nochmals durchgetestet. Am Schwierigsten war es mit dem Assembler Konverter. Dieses Tool wurde extra für die SBG entwickelt und musste vor Ort getestet werden. ES brauchte die ersten sechs Monate bis er einwandfrei lief. Das gleiche traf für die Datenkonversionswerkzeuge zu. Sie mussten den besonderen Speichertechnik der Univac Maschine angepasst werden. Es bedarf viele Tests bis sie endlich Einsatzbereit waren. Die Datenkonversion lag in den Händen der ehemaligen SZKI Mitarbeiter unter der Leitung von Erika Nyári. Die Programmkonversion einschließlich der Assembler und der JCL Konversion wurde von dem Team der ehemaligen Szamalk Mitarbeitern unter Leitung von Gabor Jandrasics durchgeführt. Ich habe mich um die COBOL Restrukturierung gekümmert. Ein weiteres Team alter Szamalk Mitarbeitern um Frau Erdős baute die Modultestwerkzeuge um die Modulergebnisse zu validieren und die Testüberdeckung zu messen. Die Verantwortung für den Abnahmetest lag bei dem SBG Testteam in Lausanne. Dieses Team sollte die Ausgabedateien aus der Produktion des Altsystems mit den Ausgabedateien und Datenbanken des konvertierten Systems vergleichen und die Abweichungen an uns melden. Dafür hatten sie eigene Werkzeuge. Zu diesem Zweck wurden über 800 Bänder mit Daten aus der Produktion gesammelt.

Die ungarischen Mitarbeiter würden alle an dem Erlös des Projektes beteiligt. Später als es richtig los ging wurden sie nach konvertierten Anweisung bezahlt. Die SBG zahlte Sfr 4,- pro Anweisung. Davon bekamen die Ungarn 75%, bzw. Sfr 3,- pro Anweisung. Das war für sie damals viel Geld. Sie haben sich sehr angestrengt und bis Dezember war das Probeprojekt erfolgreich abgeschlossen. Wir hatten alle Ziele erfüllt und die Bank war zufrieden. Wir bekamen den Auftrag, das ganze Promega zu konvertieren.

Dieses Pilotprojekt war für mich aus anderer Hinsicht auch eine Lehre in Volkswirtschaft. Die gleichen Menschen, die zwei Jahre zuvor bei Thyssen die Aufgaben vor sich hingeschoben haben und unbefriedigende Ergebnisse ablieferte, waren jetzt hoch motiviert. Alle Aufgaben wurden rechtzeitig erfüllt und mit für die Bank-Projektleiterin ausreichende Qualität. Ich habe gelernt, dass man das bekommt, wofür man bezahlt – Projektarbeit muss nach Leistung bezahlt werden.

7.2.4 Promega Projektablauf

Das Promega Migrationsprojekt wurde in vier Teilprojekten aufgeteilt:

- die Job Control Konversion
- die Assembler Konversion

- die COBOL Konversion und
- die Datenkonversion.

Die Job Control Prozeduren haben die Ressourcen zugeteilt, die Programme angestoßen und den Ablauf überwacht. Dazu verwendete sie eine einfache Assembler-ähnliche Sprache. Diese Sprache musste in die viel kompliziertere OS-1100 Job-Steuerungssprache des Unisys Rechners übersetzt werden. Diese Aufgabe übernahm ein erfahrener Assembler-Programmierer aus dem SzKI Team. Sie wäre nicht allzu schwierig gewesen, wenn die Utility Routinen nicht gegeben hätte. Mit den Utility Routinen wurden Dateien selektiv kopiert, sortiert und wieder zusammengeführt. Die Routinen enthielten eine Logik mit „und“, „oder“ Bedingungen. Sowas gab gar nicht auf dem Zielrechner, dem Unisys 1100. Daher mussten aus diesen JCL Dienstprogrammen kleine COBOL Programme generiert werden, die von den Kontrollprozeduren auf dem Unisys aufgerufen wurde. Es gab zum Schluss 108 solche parametrisierte COBOL Dienstprogramme.

Die Konversion der Univac Assembler Programme war die schwierigste Aufgabe im ganzen Projekt. In dem alten Promega System gab es 27 Assembler Hauptprogramme und 123 Assembler Subroutinen mit insgesamt 150.000 Assembler Anweisungen. Dafür wurde ein Tool verwendet, das für die Übersetzung von IBM-Assembler in COBOL konzipiert wurde. Natürlich musste das Tool der Univac Assembler Sprache „SPURT“ angepasst werden. Die Datenstrukturen waren aufgrund der 30-Bit Wörter ganz anders. Da alle Datenreferenzen auf der Basis der physischen Datenlokation mit Basisadresse und Absetzung, war die Datenadressierung sehr heikel. Erschwert wurde dies durch die Overlay-Technik, die überall verwendet wurde. Ein ehemaliger Systemprogrammierer von der Szamalk mit Erfahrung in der Umsetzung von Assembler Code auf der russischen R-Maschine hat diese Aufgabe übernommen und meisterhaft gelöst.

Die Konversion der COBOL-68 Programme setzte voraus, dass die Programme erst restrukturiert wurden. Die Restrukturierung bestand hauptsächlich darin, die viele GOTO Verzweigungen zu entfernen und die übergroße Programme in mehrere Prozeduren, in COBOL Sections, aufzuteilen. GOTO Verzweigungen innerhalb einer Prozedur waren erlaubt aber nicht zwischen Prozeduren. Die Prozeduren wurden von einer zentralen Steuerungsroutine aufgerufen. Alle Datei- Ein- und Ausgabe Operationen wurden in IO-Prozeduren versetzt, eine für jede Datei bzw. Datenbank. Hierfür wurde das Tool SoftRecon verwendet, das wir ursprünglich für die Restrukturierung der Thyssen COBOL Programme entwickelt hatten. Das Tool hatte sich inzwischen in zwei weitere Sanierungsprojekte in Deutschland bewährt.

Nach der Restrukturierung gingen die COBOL Programme durch eine für dieses Projekt speziell entwickeltes Tool, welche die alten COBOL IO-Anweisungen und CALL-Anweisungen durch DELTA Makros und die zentrale Steuerung mit PERFORM Anweisungen durch JSP Makros ersetzte. Alle internen Datenstrukturen wurden in COBOL Copy-Strecken ausgelagert. Die Univac-spezifische Anweisungen wurden durch die entsprechende Unisys Anweisungen ersetzt. Das Resultat war ein JSP/DELTA/COBOL Programm sowie es die Bank-Konvention vorschrieb. Auf dieser Weise wurden 169 Programme mit circa 199.000 Codezeilen umgesetzt.

Die Datenkonversion erfolgte ebenfalls in zwei Schritte, erstens mit einer Datenrestrukturierung und zweitens mit der eigentlichen Datenumsetzung. Bei der Datenrestrukturierung wurden redundante Dateien und Sätze eliminiert, Datenredefinitionen in separate Sätze verlagert und die tief verschachtelte Datenstrukturen refaktoriert. Das Ganze zielte auf die Normalisierung der Daten. Das wurde nicht in allen Fällen konsequent zu Ende geführt, aber die neuen Datenstrukturen waren jedenfalls viel näher an einer relationalen Datenbank als die alten.

Im zweiten Schritt wurden die restrukturierten Daten von der Univac 6-Bit Byte Darstellung in die Unisys 9-Bit Byte Abbildung transformiert. Dazwischen wurden die Daten in ASCII Zeichenformat abgebildet, d.h. wir haben zunächst eine Univac Datei gelesen und in eine ASCII Datei umgesetzt. dann wurde die ASCII Datei gelesen und in einer Unisys Datei geschrieben. Dabei wurden die Daten in das Zielformat ausgegeben. die neue Bytelänge führte zu Verschiebungen im Satzformat. Als Folge mussten die Satzbeschreibungen angepasst werden. Besonders schwierig war die Umsetzung der Index-sequentiellen Dateien wegen der Positionierung der vielen Zugriffsschlüssel. Diese mussten mit den Schlüsseldefinitionen in dem Programmcode genau abgestimmt werden. Die Verschiebung der Daten und die fehlende Abstimmung der neuen Daten mit den Datenbeschreibungen im Programmcode waren die Hauptfehlerursache. Das lag auch an der Arbeitsteilung. Die Verantwortung für die Datenkonversion lag bei den alten SzKI Mitarbeitern, die für die Programmkonversion bei den alten Szamalk Mitarbeitern. Wie in alten Zeiten, gab es die üblichen Kommunikationsschwierigkeiten zwischen den Beiden. Nichtsdestotrotz konnten alle 519 Promega Dateien erfolgreich umgesetzt werden.

Das letzte Teilprojekt war der Test. Der Modultest basierte auf einer Simulation der Eingabedateien und der Steuerungsparameter. Sie wurden von einem Testtreiber generiert. Nach jedem Programmdurchgang wurde die Ausgabesätze abgefangen und mit den Sätzen aus der Produktion bzw. aus dem letzten Test abgeglichen. Gleichzei-

tig wurde durch einen Testmonitor due Testüberdeckung aller Programmzweige registriert. So war es möglich zu erkennen, welche Pfade durch das Programm durchlaufen wurde. Durch den Abgleich der Ausgaben konnten wir viele Programmfehler finden und entfernen, ehe die Programme an den französischen Testteam gingen. Nachher wurde es viel schwieriger die Fehlerursache zu lokalisieren, weil der Bezug zu dem einzelnen Modul fehlte.

7.2.5 Promega Projektabschluss

Im Mai 1991, genau 16 Monate nach dem Projektbeginn wurde das letzte Promega Programm auf der Unisys Maschine abgenommen. Sämtliche Programme, JCL-Prozeduren und Dateien waren erfolgreich umgesetzt. Natürlich wurde dieses Ereignis groß gefeiert und ein Bericht darüber erschien in der UBS Newsletter. Das Projekt hatte alle Erwartungen erfüllt und wurde dank unserer ungarischen Reengineering Ingenieure, der französischen Tester und der strengen, kompetenten Leitung von Frau von N. sogar zwei Monate vor dem geplanten Endtermin fertig. So etwas passiert nur selten in der IT Welt. Es gab ein großes Abschlussfeier für alle Beteiligte.

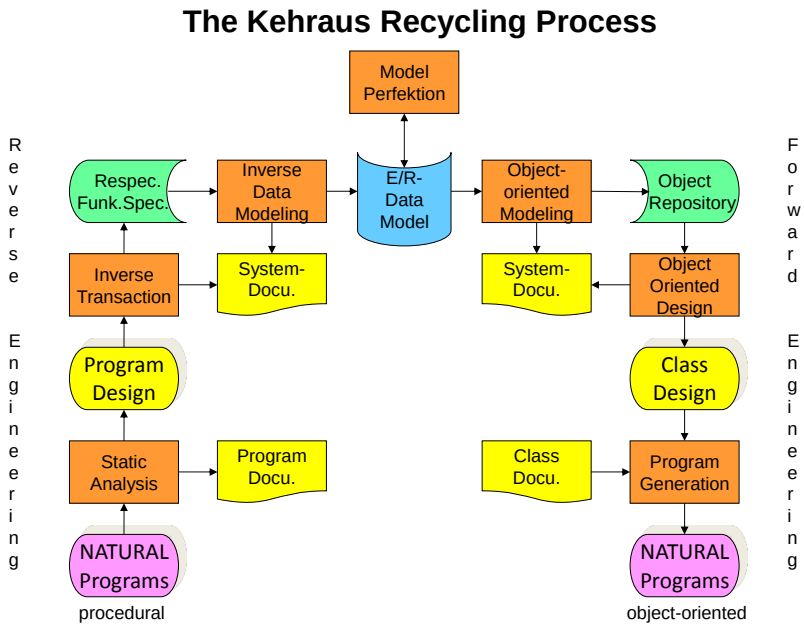
7.3 Das Kehraus Reverse Engineering Projekt

Noch ehe das Promega Projekt beendet wurde, war ich zu einer anderen Stelle in der Bank gerufen. Am anderen Ende des Software Lebenszyklus waren schon Bemühungen im Gange, die Unisys-1100 Software ABACUS mit mehr als 25.000 Delta/JSP/COBOL Programme abzulösen. D.h., während wir am unteren Ende der Migrationskette dabei waren, die letzten nicht Unisys Programme auf den Unisys zu übertragen war am oberen Ende der Migrationskette eine andere Stelle damit beschäftigt, die Unisys Programme auf einen IBM-Rechner zu übertragen. Dies war der erste von drei Versuchen, von dem Unisys auf dem IBM Mainframe umzusteigen. Die Bank war mit Recht um die Zukunft der Firma Unisys im Mainframe Geschäft besorgt. Der Tag war abzusehen, an dem sie zumindest aus diesem Geschäft aussteigen würde. Vorher sollte die SBG ihre Kern-banking Systeme auf einen anderen Mainframe Rechner hinüber gerettet haben.

7.3.1 Gescheitere Versuch mit einer 4GL Sprache

Zu diesem Zeitpunkt waren 4GL Sprachen sehr in Mode. Die Software AG beherrschte mit ihrer Sprache Natural einen großen Teil des europäischen Marktes. Ich habe selber für ihre Mitarbeiter und Kunden Kurse gehalten. Mein Hauptthema war die Aufwandsschätzung von Entwicklungsprojekte, die eine 4GL Sprache benutzten.. Zu diesem Zweck habe ich eine eigene Schätzmethode entwickelt – die Data-Point Methode, die ich im Frühjahr 1990 in der Online Zeitschrift veröffentlichte. Die IBM

hat ihre Function-Point Methode propagiert, die Software AG sollte auch eine eigene Schätzmethode haben. Bei einer Projektschätzung kommt es nicht darauf an, was man zählt – die 4GL Systeme hatten viele Größen, u.a. Daten, Datenflüsse, Entitäten, Beziehungen, Module, Objekte, Anweisungen und Codezeilen. Wichtig ist, dass was immer man zählt, repräsentativ für die Größe der Software ist und dass sie die konsistent gezählt wird damit die Projekte vergleichbar sind. Als ich nach Zürich ging, hatte ich schon einige Natural Projekte relativ gut geschätzt [Sned90]. Der technische Direktor der Software AG – Peter Pagé – hatte mich sogar in seinem Büro in Darmstadt empfangen um mich für die gute Zusammenarbeit zu bedanken. Eigentlich war dies für mich eher eine Demütigung. Die ganze Zeit bin ich mit den Softorg Werkzeuge als Konkurrent zur Software AG in der deutschen Industrie aufgetreten. Jetzt musste ich für meinen ehemaligen Widersacher arbeiten. Seine wohlgemeinte Wörter „if you can't beat them, join them" brachte mir heim wo ich gelandet war, nämlich als Verlierer im technologischen Wettkampf.



Jedenfalls hatte die SBG Management in Erfahrung gebracht, dass ich auch mit der Natural/Adabas Entwicklungsumgebung vertraut war. Das Projekt mit der Software AG ist schlecht gelaufen. Die Mitarbeiter dort kannten zwar Natural gut, aber mit den alten COBL/DELTA/JPS Programmen im ABACUS System kamen sie

schlecht zurecht. Sie waren gewohnt auf der Basis einer strukturierten Anforderungsspezifikation zu entwickelt. Hier in diesem Projekt bekam sie nur die alten Programme als Vorgabe. Sie kamen nicht voran und was sie erreicht haben, war nicht brauchbar. Die neuen Natural Programme verhielten sich anders, als die alten ABACUS Programme und lieferten inkorrekte Ergebnisse. Außerdem waren die Laufzeiten völlig inakzeptabel. Bei den größeren Programmen stieß Natural auf ihre Grenzen. Der Natural Adressraum war begrenzt – also wurde das Projekt abgebrochen. Es blieben jedoch circa 5.000 Natural Programme ohne Verwendung zurück. Die Bank hatte dafür mehrere Millionen gezahlt, sie wollte diese Summe nicht einfach abschreiben. Es musste möglich sein, etwas aus diesem gescheiterten Projekt zu retten. So entstand die Idee, die fertigen Dialogprogramme nach zu dokumentieren um sie in einer anderen Sprache neu zu implementieren. Das Projekt sollte „Kehraus“ heißen.

7.3.2 Natural Nachdokumentation

Kehraus war im Gegensatz zu Promega ein klassisches Reverse Engineering Projekt. Es ging darum, aus dem vorliegenden Natural Code ein objektorientiertes Modell abzuleiten, um dieses als Basis für eine Neuimplementierung zu nutzen. Da das Szamalk Team, jetzt hießen sie SoRing Kft. immer noch durch das Promega Projekt gebunden war, habe ich das Projekt mit dem SzKI Team, jetzt Softing Kft. begonnen. In diesem Team von Frau Nyary waren zwei Mitarbeiter mit der graphischen Datenverarbeitung vertraut. Sie wurden benötigt, um die graphische Dokumentation zu erstellen. Dazu kam Frau Erdös als Natural/Adabas Expertin. Sie hatte für das Bremer Lagerhaus gerade einen Natural Parser geschrieben um deren Programme von Natural in COBOL zu migrieren. Der Projektleiter für Kehraus war seitens der Bank ein junger Informatiker, der gerade von der Universität Linz kam. Später sollte er einer der Gründer der Firma ANECON in Wien werden. Wir haben uns hier in Zürich kennengelernt. Die Vorarbeit in diesem Projekt wurde hauptsächlich von mir und Frau Erdös geleistet. Wir haben ihr Natural Parser verfeinert und aus dem Natural Code Beziehungstabellen generiert, pro Programm:

- eine für die Programmstruktur,
- eine für die Datenstrukturen,
- eine für die Schnittstellen,
- eine für die Geschäftslogik und
- eine für die Programmflüsse.

Aus diesen Tabellen wurden diverse Dokumente ausgegeben. Darüber hinaus wurde über eine Batchschnittstelle der Strukturdaten auf das Tool PC-Spec übertragen, wo sie graphisch abgebildet wurden. Das Team der Frau Nyary ist dazu gestoßen

und hatte die Ergebnisse der Natural Codeanalyse graphisch aufgearbeitet. Ihr Tool PC-Spec hatte inzwischen als Folge des DOCKET Forschungsprojektes eine graphische Ausgabe die wir in das Bankprojekt einbringen konnten. Zum Glück passte dieses Natural Reverse Engineering Projekt ganz gut zum DOCKET Reverse Engineering Forschungsprojekt und zu dem was wir bisher auf dem Gebiet geleistet hatten [Sned88].

Das Ergebnis der Reverse Engineering war eine Natural Programm-Repository in der alle Entitäten und Beziehungen der Natural Programme gespeichert waren. Von der Repository konnten verschiedene Abbildungen ausgegeben werden, darunter Programmbaum-, Datenbaum- und Datenflussdiagramme (siehe Abbildung 7.?). Damit wurde Kehraus zu einem klassischen Reverse Engineering Projekt – Code rein und Dokumentation raus. Später habe ich auf dem IEEE Case Konferenz in Toronto darüber berichtet [Sned95].

7.3.3 Transformation von einem prozeduralen in ein Objektmodell

Das Reverse Engineering war nur die erste Phase des Projektes. In der zweiten Phase folgte das Forward Engineering. Auf Grund der Datenstrukturen wurden Objekte gebildet und anhand der Datennutzung durch die Geschäftsregel die Regel den Objekten zugeordnet. Auf dieser Weise wurden Klassen gebildet. Das Resultat war ein Klassenbaum mit Beziehungen zwischen Knoten. Die Beziehungen stellten Assoziationen zwischen den Klassen dar, bzw. Aufrufe fremder Methoden. Die früheren Natural Variabel wurden die Attribute der neuen Objektklassen. Es folgte daraus ein eigenes kleines Objektmodell für jedes Natural Programm mit Klassen- und Sequenzdiagrammen. Dies war unser erstes prozedural zu objektorientiert Reverse Engineering Projekt. Es sollte nicht das letzte sein. Über diesen Ansatz hat Frau Nyary auf der internationalen Maintenance Konferenz in Orlando berichtet [Nyar92].

Natürlich gab es viele Redundanzen in den Modellen, weil die gleichen Datenobjekte und die gleichen Regel in mehreren Programmen vorkamen. Ich hatte vor diese Redundanzen in einem dritten Schritt – ein Modelloptimierungsphase – zu bereinigen, dazu es ist aber leider nie gekommen. Die Mittel für dieses Projekt wurden gestrichen. Das SBG Management hat sich es anders überlegt. Statt die Core Banking Systeme auf der Basis der bereits erstellten Natural Programme neu zu implementieren, sollten sie von Grund auf als objekt-orientierte Systeme neu konzipiert und mit der OO-Sprache Smalltalk implementiert werden. Dazu wurde eine neue Entwicklungsabteilung unter der Leitung eines bekannten Professors von der Universität Zürich aufgebaut. Man versprach sich von der neuen Technologie einen großen Durchbruch in der Anwendungsentwicklung.

7.4 Weitere Projekte in Zürich

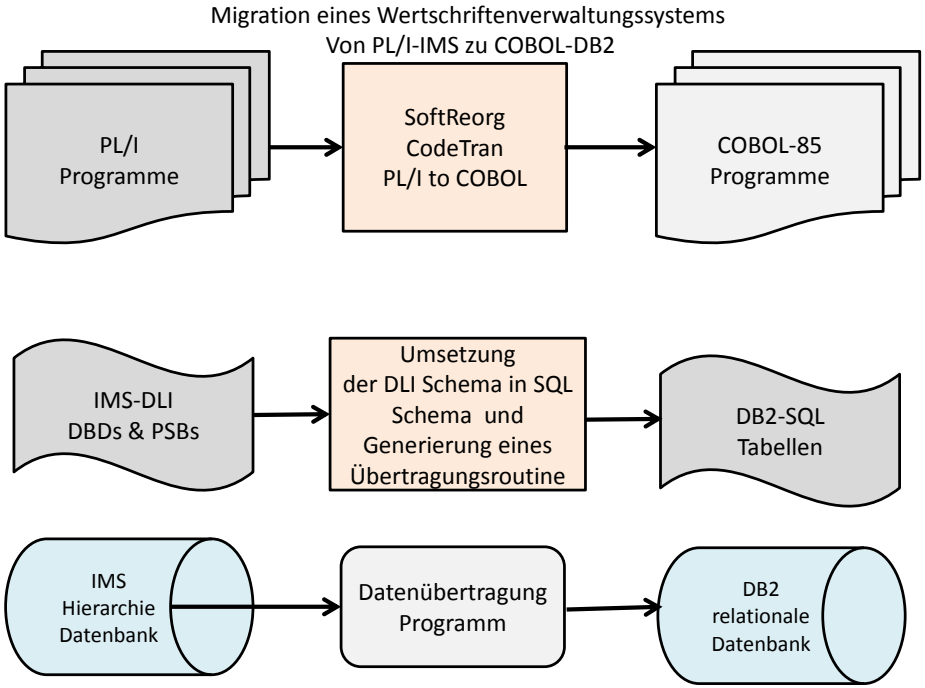
Nach Abschluss des Promega Projektes war unser Ruf in der Bank sowie in der Schweiz gefestigt. Ich bekam immer mehr Einladungen in Zürich auf Konferenzen aufzutreten und über Software Reengineering zu sprechen. Die Reengineering Welle in Europa erreichte seinen Höhepunkt und ich bin oben darauf geritten. In dieser Zeit erschien auch mein Buch „Software Sanierung“, als erstes Buch in deutscher Sprache über das Thema Reengineering [Sned90]. Viele Anwenderbetriebe, vor allem in der Schweiz wo sie mit der IT früher angefangen haben, befanden sie sich als Gefangener ihrer eigener Legacy Systeme. Sie kamen ohne die Systeme nicht aus, konnten aber die Systeme kaum aufrechterhalten. Sie stießen an die Grenze ihrer Änderbarkeit und Erweiterbarkeit. Dennoch ließ das Flut an neuen Anforderungen nicht nach. Hinzu kam, dass die laufenden Systeme total veraltet waren. Sie brauchten eine neue Migrationsstrategie um daraus zu kommen [Sned95].

Hinzu kam dass seit Ende der 80er Jahren die neue Objekttechnologie auf dem Vormarsch war. Sie war das Gebot der Stunde, aber mit ihren alten prozeduralen Systemen hatten die Anwenderbetriebe keine Möglichkeit sie zu nutzen. Sie konnten nicht ohne weiteres umsteigen so lange die alten Systeme noch im Betrieb waren. Der Druck die alte Software abzulösen war enorm aber keiner wusste wie. Es schien ein unlösbares Problem zu sein. In diese Lücke bin ich mit den neuen Werkzeugen hinein gestolpert. Wie mit der CASE-Technologie in den 80er Jahren bin ich nun in den 90er Jahren auf der Welle der Reengineering Technologie hochgeschwommen [Sned91].

7.4.1 PL/I zu COBOL Konversion

In der UBS bekamen wir weitere Projekte. Ein Projekt war ein altes PL/I-IMS System von der IBM in COBOL mit DB-2 umzusetzen. Da wir bei den alten SzAMALK Mannschaft jede Menge PL/I Know-how hatten, war dies für uns kein Problem. Wir mussten nur die Tools erweitern. Wir hatten ohnehin begonnen, alle Tools auf den PC umzustellen. Ich musste wegkommen von den verbleibenden Mainframe Rechner in Budapest – ihre Nutzung war seit dem Systemwechsel für mich zu teuer geworden. Außerdem war es nur eine Frage der Zeit, bis sie verkauft oder verschrottet werden würden. Ich habe deshalb im Alter von 52 Jahren wieder begonnen, selber zu programmieren. Seit Ende der 70er Jahren hatte ich nicht mehr selber programmiert. Die Toolentwicklung hatte ich den Ungarn überlassen. Das brachte mich in ein Abhängigkeitsverhältnis. Um da rauszukommen, musste ich wieder selbst programmieren. Die neuen PC Rechner mit dem Windows-3.1 Betriebssystem gaben mir die Gelegenheit dazu. In der SBG war ohnehin eine Bewegung im Gange, die Entwicklung auf PCs zu versetzen. Die COBOL Programme sollten auf dem PC Arbeitsplatz ge-

schrieben und kompiliert werden, dann erst sollte der Entwickler sie auf den Mainframe zum Testen übertragen. Dr. Thurner war im Begriff das Konzept eines PC Entwicklungsplatzes für die Bank auszuarbeiten – der ABACUS-Arbeitsplatz. Im Falle unserer Werkzeuge sollten sie sowohl auf dem PC als auch auf dem Mainframe ausführbar sein. Dies war für mich die Gelegenheit, wieder in die Programmierung einzusteigen.



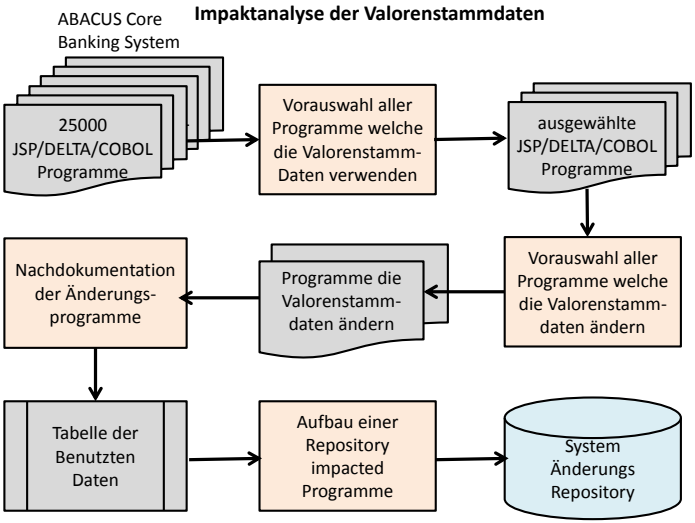
Das erste PC Programm, das ich schrieb war in der Micro-Focus COBOL Sprache. Später sollte ich auch Borland C++ Programme schreiben. Java gab es zu diesem Zeitpunkt noch nicht. Meine ersten Programme waren für die Konversion der IMS Datenbankbeschreibungen in DB-2 relationale Tabellen. Dabei habe ich gleich die COBOL Programme generiert, um die alten Daten aus IMS zu holen und in die neue DB-2 Datenbanktabellen zu speichern. Frau Erdös, die inzwischen selbstständig war, hat mir dabei geholfen. Das Team von der SoRing Kft., die neugegründete Firma von Gabor Jandrasics, entwickelte ein neues PC Tool, um die PL/I Programme in COBOL umzusetzen. PlITran soll sie heißen. Das Vorgänger Tool – Pli2COB - war in PL/I

geschrieben und lief nur auf dem IBM Mainframe. Wir nutzten diese Gelegenheit uns weiter von dem Mainframe zu lösen. Unser Ziel war alle Tools auf dem PC zu haben.

Das PL/I-IMS zu COBOL-DB2 Konversionsprojekt war wieder ein guter Erfolg für uns und brachte weitere Kredite bei der Bank ein. Ein Projekt, das für 18 Monate vorgesehen war, haben wir unter 12 Monate durchgezogen und für nur ein Viertel der Kosten, die IBM dafür verlangt hat. Auch in diesem Reengineering Projekt haben wir nach umgesetzter Anweisung abgerechnet, Sfr. 4,- pro getestete Anweisung. Dieser Zahlungsmodus hat sich bewährt. Da die Mitarbeiter daran beteiligt waren, waren sie hoch motiviert, diese sonst langweilige Übersetzungsarbeit durchzusetzen. Jetzt wann immer ein Problem mit Altsoftware in der Bank aufkam, wurden wir beauftragt das Problem zu lösen.

7.4.2 Impaktanalyse des ABACUS Systems

Als die Bank entschied, ihre Valorenstammdaten von der Unisys DMS-1100 Datenbank in eine IBM-DB2 Datenbank zu versetzen, wurde ich beauftragt herauszufinden, welche der 25.000 ABACUS COBOL Programme die Daten benutzen und wie – in Retrieval oder Update Modus. Sie wollten wissen, welche Programme sie ändern mussten. Da die Ungarn alle in anderen Projekten eingebunden waren, habe ich dieses Projekt selber übernommen.



Das Projekt war eine klassische Impactanalyse [Arno90]. Zuerst habe ich die gesamte Source-Bibliothek gescannt, um zu entdecken, welche Programme die Valou-

renstammdatei überhaupt öffnen. Es waren wenige als 5.000. Diese habe ich anschließend durch mein COBOL Parser geschickt, um die READ und WRITE Anweisungen auszuweisen. Die so erfassten Anweisungen habe ich in einer großen Tabelle gesammelt, aus der hervorging, welche Datensätze von welchen Programmen geändert, gelöscht oder hinzugefügt wurden. Ich ging so weit um auch die einzelnen geänderten Felder aufzulisten. In weniger als einen Monat war die Aufgabe gelöst. Die Bank konnte die Daten migrieren und die betroffenen Programme entsprechend anpassen.

7.4.3 PL/I Nachdokumentation

Inzwischen kamen auch Aufträge von Schweizer Firmen außerhalb der SBG – nämlich von Swissair und von der Züricher Versicherung. Für andere Schweizer Banken durfte ich laut meinem Vertrag nicht arbeiten. Von der Swissair kam ein Projekt die Datenbank für ihr Flugplanungssystem zu konvertieren und von der Züricher Versicherung ein Projekt ihre PL/I Programme nach zu dokumentieren. Sie wollten sich aus der Abhängigkeit von bestimmtem Programmieren herauslösen und dachten mit der Nachdokumentation könnten andere Entwickler die Wartung und Weiterentwicklung übernehmen. Wegen der mangelnden Kommentierung und der nichtsprechenden Datennamen ist dies nur zum Teil gelungen. Wir sind hier auf die Grenzen der Reverse Engineering gestoßen. Es waren die gleichen Grenzen auf die wir in dem EU-Forschungsprojekt DOCKET gestoßen sind. Die Lösung wird durchleuchtet aber der Bezug zum fachlichen Problem bleibt verborgen.

Die Züricher Versicherung hatte auch ein System auf dem AS400, dass sie renovieren wollte. Es handelte sich um AS400 COBOL Programme mit CL Control Prozeduren und index-sequentielle Dateien. Mit unserer inzwischen ausgereiften SoftRecon Tools war es für uns leicht, den Code zu rekonstruieren. Neu war in diesem Projekt das Testwerkzeug, welches ich für die Validation veränderter Programme gebaut habe. Es war schon immer ein Problem gewesen restrukturierten Code zu testen, bzw. die funktionale Äquivalenz zum alten Code nachzuweisen. Mit dem Tool CobRetest konnte ich Datenzuweisungen in den Source Code einbauen die gewisse Abläufe verursachen. Es wurden danach die Ablaufpfade in dem alten unstrukturierten Programm sowie in dem neuen restrukturierten Programm aufgezeichnet um die Pfade miteinander zu vergleichen. Durch den Pfadabgleich gelang es die funktionale Äquivalenz der beiden Programmversionen – alt und neu - nachzuweisen. Das war für uns ein großer Fortschritt. Jetzt konnten wir reengineered Programme auf dem PC-Arbeitsplatz testen und waren nicht mehr vom Rechner des Kunden abhängig [Sned94].

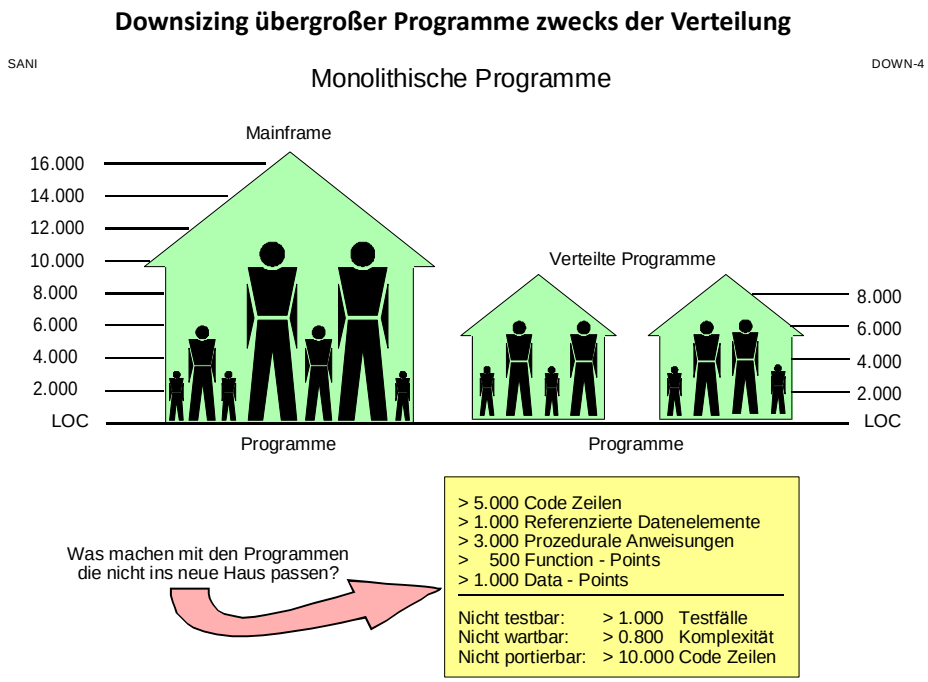
7.5 Ein missglückter Abstecher nach Deutschland

Gerade zu dieser Zeit, als wir in der Schweiz eine Unterbrechung in der Projektarbeit hatten, kam ein großes Reengineering Projekt in Deutschland auf mich zu. Die Firma Ploenzke, eine renommierte deutsche Beratungsfirma in Wiesbaden hatte einen mittelgroßen Industriekunden in Darmstadt, der von einen alten UNISYS Mainframe Rechner auf einen AS400 Rechner umsteigen wollte. Durch die öffentlichen Seminare und das Newsletter der SES waren unsere Leistungen auf dem Gebiet der Altsystemrenovierung dem Ploenzke Management bekannt. In Deutschland gab es zur diesen Zeit kaum andere Firmen mit Reengineering Erfahrung. Case Consult war gerade im Entstehen. Deswegen wollte Ploenzke uns auf jeden Fall in das Projekt einbeziehen. Sie haben nicht nur mich sondern auch meine Projektleiter Gabor Jandrasics, Erika Nyary und Kati Erdös nach Wiesbaden eingeladen um uns für das Projekt zu gewinnen. Ihre Strategie war hier schon durchschaubar. Sie wollten die Ungarn unter ihre Kontrolle bringen und mich später ausspannen. Das ist ihnen auch gelungen.

Die Gesamtleitung sollte in den Händen der Ploenzke bleiben, wir sollten die Verantwortung für die Einzelprojekte übernehmen. Die Firma von Herrn Jandrasics – SoRing – sollte die Unisys-COBOL Programme in AS400 Programme umsetzen. Die Firma von Frau Nyary – Softing – sollte die Unisys Datenbanken in AS400 Dateien umsetzen und auch die Unisys Link Programme in AS400 COBOL konvertieren. LINK war eine Unisys 4GL Sprache für die Online Datenverarbeitung. Ich sollte mit meinem SES Team die konvertierten Programme testen. Ein viertes Team von der Anwenderfirma sollte die konvertierte Software abnehmen und installieren. Vorher sollte ich die gesamten Projektkosten im Namen der Firma Ploenzke schätzen, denn es sollte ein Festpreisprojekt sein. Aufgrund einer Analyse des Codes und der Daten bei der ich die Function-Points und Data-Points gezählt habe, und unter Berücksichtigung unserer bisherigen Projekterfahrungen kam ich auf einen Betrag von DM 6,5 Million. Dies hatte der Kunde akzeptiert und das Projekt konnte im Sommer 1993 beginnen.

Der Schlüssel zum Erfolg in diesem wie auch in jedem Migrationsprojekt waren die Werkzeuge. Für die Restrukturierung und Konversion der COBOL Progaramme hatten wir CobRecon, der inzwischen auf dem PC lief. für die Konversion der Daten hatte Frau Nyary ihr Tool DatRecon. Für die Link Transformation musste sie ein neues Tool entwickeln. Das war eine ziemlich große Aufgabe für zwei ihrer Leute und hat bis Ende des Jahres gedauert. Dennoch hatte es dazu keine Alternative gegeben. Es wäre undenkbar gewesen, die mehr als 500.000 Link Anweisungen manuell zu übersetzen. Für den Test mussten wir auch neue Werkzeuge für die AS400 entwi-

ckeln. Ich habe das Team von der ungarischen Akademie der Wissenschaften unter Endre Somos engagiert um einen Modultester zu bauen und die Testüberdeckung zu messen. Ich hatte mich vertraglich verpflichtet sowie damals in dem Siemens Projekt die Module zu mindestens 85% Zweigüberdeckung zu testen. Frau Erdős hatte ihre alten SoftInt Werkzeuge für den Integrationstest auf den AS400 übertragen. Dazu musste sie sie in COBOL neu implementieren. Auf dem Mainframe sind sie in PL/I gewesen. Das war eine sehr aufwendige Arbeit und hat mindestens sechs Monate gedauert. Das Migrationsprojekt begann also mit einer ausgedehnten Tooling Phase.



Unter den COBOL Programmen des Kunden gab es ein Monsterprogramm mit mehr als 35.000 Codezeilen – das Faktura Programm. Es war vom Anfang an klar, dass dieses Programm nicht so auf dem AS400 übertragen werden konnte. Es musste in mehrere Programme zerlegt werden. Ich habe mir die Sache persönlich angenommen und machte es zum Forschungsthema = „Program-Downsizing“ [Guen92]. Das Downsizing des Faktura Programmes wurde zum eigenständigen Projekt im Projekt. Herr Jandrasics stellte einen jungen ungarischen Informatiker zur Verfügung, der mir dabei helfen sollte. Der Entwickler des Programmes – ein gewisser Herr K. - war ein

älterer Programmierer mit einem photogenischen Gedächtnis. Man konnte ihn fragen, wo etwas in dem Programm geschieht und er wusste sofort, an welcher der tausend Zeilen. Ich bin vorher so jemandem nie begegnet. So, wie die Versicherung in Zürich in der Abhängigkeit zu einem einzigen launischen Entwickler gefangen war, war dieser Firma in Darmstadt in der Abhängigkeit des Herrn K. Er bezog ein hohes Gehalt, fuhr ein Firmenauto und bekam seine Urlaubsreisen bezahlt. Er herrschte über die Faktura Applikation wie ein König über sein Königreich. Er bestimmte alles, was darin passierte und ließ keinen anderen dran. Es wagte auch niemand seinem Reich zu betreten. Der neuen IT-Leitung konnte er die Stirn bieten. Sie war von ihm völlig abhängig. Dies war auch u.a. einer der Gründe für die Migration. Die IT Leitung wollte sich aus dieser Abhängigkeit befreien.

Ich habe viel Zeit mit Herrn K. verbracht. Am Ende ist es uns gelungen das Monsterprogramm in 15 Einzelprogramme mit einer zentralen Steuerung zu erlegen. Das geschah zuerst nur aufs Papier. Ich zeichnete in Diagrammen auf, wie die neue Architektur des Programmes auf der AS400 aussehen sollte. Dann wurde der junge ungarische Mitarbeiter damit beauftragt, die Zerlegung des Codes einschließlich der Restrukturierung der Einzelteile durchzuführen und das neue Programm auf der AS400 durchzutesten. Der Test war besonders langwierig. Er hat mehr als 60% des Gesamtaufwandes gekostet, obwohl er größtenteils automatisiert war. Die Refakturierung und Konvertierung des Faktura Programms mit Regressionstest dauerte fast ein Jahr. Es war eine hervorragende Fallstudie in Refakturierung, eine Studie, die ich später sehr oft zitiert habe [Sned95].

Inzwischen ging die Restrukturierung und Konvertierung der anderen Batchprogramme in Budapest weiter. Das SoRing Team setzte die Programme um und mein Team hat sie in den IBM Rechenzentrum in Budapest getestet. Wir hatten dort einen AS400 Rechner gemietet. Am Anfang war der Test sehr aufwendig. Es dauerte 3 bis 4 Tage pro Programm bis die erforderliche Testüberdeckung erreicht war. Erst im Spätherbst 1993 als die Testwerkzeuge von Frau Erdös endlich fertig waren, konnte der Test beschleunigt werden. Mit den neuen Datengenerierungs- und Validierungswerkzeugen dauerte der Test eines Programms maximal 2 Tage. Jetzt war der Engpass die Fehlerkorrektur. Die Tester haben die Fehler an das Reengeneering Team gemeldet. Dieses Team wurde nach der Zahl der ungetesteten und kompilierten Anweisungen bezahlt, nicht aber für die Zahl der fehlerfreien Anweisungen. Wie später auch die Inder, hatten sie wenig Interesse daran, ihre kostbare Zeit mit Fehlerkorrekturen zu verbringen. Dies war ein Mangel in der vertraglichen Vereinbarung. Als Folge mussten die Tester oft eine Woche oder mehr auf eine korrigierte Version des Programmes warten. Sie gingen dazu über die Fehler selber zu korrigieren. Manche

Programme wurden auch mehrfach getestet, weil immer neue Fehler auftauchten. Der Test zog sich langsam hin.

AS400 Teststatusbericht aus dem Jahr 1993

PLOENZKE/SES AS-400 TEST STATUS SITE: BUDAPEST
AUTHOR: SNEED BERICHT DATE: 02.04.93
SUBJEKCT: STATUS OF COBOL PROGRAM TEST IN BUDAPEST PAGE: 1 OF 2

PROGRAM	TESTER	STATUS	NR LINES	NR STMTS	NR BRCHS	NR FILES	NR FKPT	NR ERRS	CONF RATE	CORR RATE	COVR RATE
GFL000	FUELE	TESTED	1986	544	292	9	160	11	0,91	1,00	0,91
(P3053)	(12)	30.10									
GFL001	FUELE	TESTED	4385	1231	483	34	162	9	0,89	0,99	0,73
(P3001)	SZABO	04.12									
GFL002	SMIKAL	TESTED						5			
(P3024)	SZABO	18.12									
GFL003	SMIKAL	TESTED	455	93	54	3	15	1	0,92	1,00	0,92
(P3060)	(4)	30.11									
GFL004	SMIKAL	TESTED	1334	417	189	8	55	11	0,96	1,00	0,91
(P3061)	(8)	19.11									
GFL005	FUELE	TESTED						8			
(P3093)	SZABO	27.11									
GFL012	FUELE	TESTED	1744	589	305	10	95	6	0,94	1,00	0,73
(P3067)	(7)	03.02									
GFL013	PAP	TESTED	1820	619	320	10	95	5	0,95	1,00	0,75
(P3068)	(6)	04.02									
GFL014	FUELE	TESTED	2006	703	379	10	95	4	0,95	1,00	0,66
(P3069)	(5)	27.01									
GFL015	FUELE	TESTED	1994	696	370	10	95	4	0,95	1,00	0,65
(P3070)	(4)	28.01									
GFL016	SMIKAL	TESTED	1993	679	349	10	95	4	0,95	1,00	0,66
(P3071)	(5)	03.02									
GFL017	SMIKAL	TESTED	1702	586	292	10	19	3	0,95	1,00	0,75
(P3072)	(4)	03.02									
GFL018	SMIKAL	TESTED	930	264	124	5	35	1	0,94	1,00	0,91
(P3063)	(3)	21.01									
GFL019	PAP	TESTED	1189	375	192	6	55	2	0,93	1,00	0,79
(P3081)	(2)	04.02									
GFL020	SMIKAL	TESTED	1190	373	192	6	55	2	0,93	1,00	0,91
(P3082)	(3)	04.02									

Ein weiterer Grund für das Hinziehen des Tests war die Vereinbarung über die Testüberdeckung. Im Vertrag waren wir dem Kunden gegenüber verpflichtet, bei jedem getestetem Programm mindestens 85% Zweigüberdeckung zu erreichen. Andererseits war der Kunden verpflichtet, uns adäquate Testdaten zu liefern. Bald stellte sich heraus, dass die Daten, die die Kunden aus der Produktion abzogen, alles andere als adäquate waren. Sie haben nur einen Bruchteil des entsprechenden Codes abgedeckt. Das lag daran, dass die meisten Programme Kopien von anderen Programmen waren, die der Entwickler nur geringfügig änderte um der jeweiligen Aufgabe gerecht zu werden. Ein großer Teil des Codes wurde für die spezifische Aufgabe gar nicht genutzt. Also konnten wir diesen Code mit den gelieferten Testdaten unmöglich erreichen. Wir müssten die gelieferten Testdaten manuell editieren, bzw. anreichern, um weitere Datenkombinationen zu erzeugen. Dies stellte sich als äußerst aufwendige Verfahren heraus. Der Test kam nicht voran und die Testrechnerkosten bei dem IBM-Budapest häuften sich. Es müsste etwas geschehen.

Nach Gesprächen mit der Ploenzke Projektleitung und dem Kunden wurde entschieden, dass wir dem Kunden helfen sollten, adäquate Testdaten aufzubereiten. Derweil sollten wir den Test in Budapest einstellen, Das heißt, wir mussten ein Tool erstellen, um die Testüberdeckung auf dem Unisys Mainframe in Darmstadt zu messen. Dies habe ich zusammen mit Frau Erdös gemacht, aber es dauerte zwei Monate. Es war März bis wir den Test in Darmstadt beginnen konnten. Danach konnten wir mit Hilfe der zuständigen Fachabteilungen den Testdatenbestand in den Zustand gebracht, in dem er vom Anfang an hätte sein sollen. Wir konnten den alten Code auf dem Mainframe weitgehend abdecken und den Teil, den wir nicht erreichen konnten, haben wir entfernt. Er war überflüssig. Ende Mai waren wir mit der Aufbereitung der Testdaten endlich fertig. Wir hatten damit 6 Monate verloren, was hieß, das Projekt war sechs Monate im Verzug.

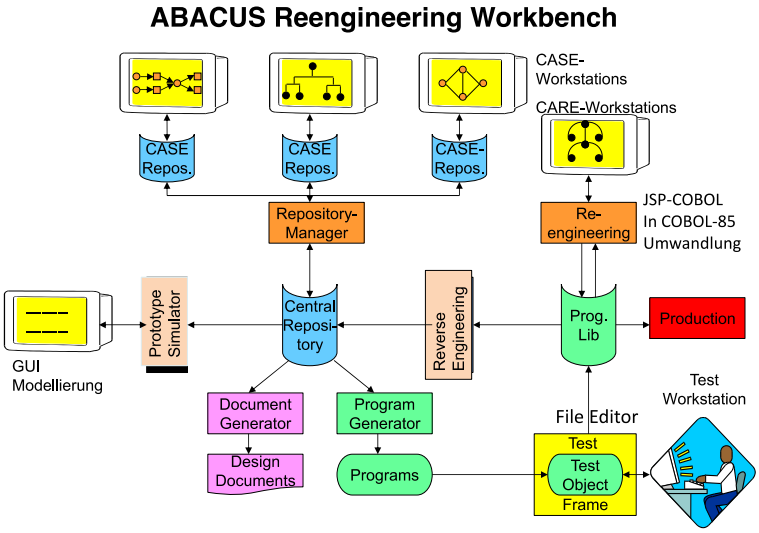
In Anbetracht des heranrückenden Endtermins – das Projekt sollte bis Ende des Jahres 1994 abgeschlossen sein – hat Ploenzke entschieden, den Systemtest in Budapest auszulassen und gleich mit dem Abnahmetest in Darmstadt zu beginnen. Die eingesetzten Programme wurden gleich nach der Kompilierung auf den AS400 Produktionsrechner eingespielt und von den Fachabteilungen getestet. Das Team von Herrn Jandrasics musste fortan in Darmstadt sitzen und die Fehler an Ort und Stelle korrigieren. Ich wurde aus dem Projekt entlassen und ging zurück zur UBS in Zürich. Nach dem Projekt bei Thyssen Stahl in 1988 war dies meine zweitgrößte Niederlage. Ich hätte das mit der Testüberdeckung niemals in den Vertrag bringen dürfen. Der Kunde hatte es nicht verlangt. Ploenzke war mit Recht auf mich sauer.

Die AS400 Systeme sind Juni 1995 endlich zum Einsatz gekommen aber mit sechsmonatiger Verspätung und 2 Millionen Mark Mehrkosten. Dies hatte auch ein Nachspiel. Ich bekam einen Brief von der Ploenzke Anwalt, meine Firma, die SES solle die Summe von einer Million DM an die Ploenzke überwiesen. Sie wollten den Verlust mit mir teilen. Ich hatte 6,5 Millionen für die Migration geschätzt und auf dieser Basis wurde ein Festpreis mit dem Kunden vereinbart. Am Ende hat das Projekt aber 8,5 Millionen DM gekostet und ich sollte die Verantwortung dafür mittragen. Nicht nur hatte ich das Projekt unterschätzt, ich hatte auch mit dem Test in Budapest das Projekt verzögert. Das mit der minimalen Testüberdeckung hätte ich in den Vertrag nie einbringen sollen, und wenn, hätte ich dafür sorgen müssen, dass ich das auch einhalten kann. Ploenzke hatte Recht. Ich war schuld. Einen Prozess hätte ich sicherlich verloren. Zum Glück wurde Ploenzke kurz danach von einem großen amerikanischen Softwarehaus übernommen, welches in den deutschen Markt einsteigen

wollte und die Sache mit den zwei Millionen DM fiel unter den Tisch. Ich hatte Glück gehabt.

7.6 Das große ABACUS Migrationsprojekt

Als ich geschlagen von Deutschland wieder in die Schweiz zurückkam, wartete ein neues Projekt auf mich. Es sollte mein größtes Reengineering Projekt sein. Nach dem misslungenen Versuch der Software AG, das Kern-Banking System der UBS mit Natural/ADABAS zu reimplementieren, hat die Bank nochmals versucht, das ganze objektorientiert mit Smalltalk neu zu entwickeln, Dabei sollten die Bankanwendungen auf mehrere Unix Rechner verteilt werden. Eine neue Abteilung mit 200 Mitarbeitern unter Leitung eines renommierten Hochschulprofessors wurde zu diesem Zweck ins Leben gerufen. Die Mitarbeiter der Abteilung waren größtenteils junge Akademiker, direkt von der Hochschule. Dieses Projekt war leider von Anfang an mit Schwierigkeiten behaftet. Die jungen Entwickler hatten keine ausreichende Spezifikation ihrer Aufgabe. Sie bekamen nur die alten COBOL/JSP/DELTA Source Programme vorgelegt und sollten daraus ein neues objektorientiertes System schaffen. Der Informationsaustausch zwischen den alten COBOL Entwicklern und den jungen Smalltalk Entwicklern hat nie so recht funktioniert. Einerseits konnten sie sich wegen der unterschiedlichen Technologien und Denkweisen nur schwerlich verständigen, anderseits waren die alten COBOL Entwickler nicht besonders bereit ihr Wissen preiszugeben. Die Anwender und Analytiker waren zudem wenig hilfreich. Sie hatten keine Ahnung mehr, was in der Software geschieht. Die Details hatten sie schon längst vergessen.



Mein Protagonist bei der Bank hatte schon geahnt, dass aus dem Smalltalk Projekt nichts wird. Er war jedoch dafür nicht zuständig. Er war Leiter Operations und nicht Leiter der Entwicklung. Wohlweislich bereitete er ein Plan-B. vor. Er wusste, dass der Unisys Mainframe bald aus dem Betrieb genommen wird. Unisys hatte schon angekündigt, dass sie sich aus dem Mainframegeschäft zurückziehen würde. Er musste deshalb bald die Bank-Software auf einen anderen Mainframe versetzen. Die neuen Applikationen liefen schon auf dem IBM Mainframe.

7.6.1 Der ABACUC Migrationsarbeitsplatz

Die Voraussetzung für die Übertragung der ABACUS COBOL Programme auf der IBM war, dass sie in ANSI-Standard COBOL geschrieben sind. Die Delta Makros konnten bleiben, aber die JSP Strukturierungs-Anweisungen mussten weg. Die Grundstruktur des Codes mit den Steuerungskonstrukten – Sequenz, Auswahl und Wiederholung – konnte bleiben, aber sie musste durch die neuen COBOL-85 Kontrollanweisungen implementiert werden. Unsere Aufgabe war es, ein Tool zu entwickeln welches die JSP-COBOL Programme automatisch in COBOL-85 Programme umsetzt. Das Tool sollte sowohl auf dem Unisys Mainframe als auch auf dem PC Arbeitsplatz – damals Windows 3.1 laufen.

Das Transformationswerkzeug sollte ein Baustein in einer allgemeinen PC-basierten Entwicklungsumgebung sein. Das Konzept für diese Umgebung stammte von Dr. Thurner, der von der UBS beauftragt wurde, eine Entwicklungs- und Testumgebung für den PC mit Anschluss an dem Mainframe zu konzipieren – der sogenannte ABACUS-Arbeitsplatz. Zu dieser Umgebung gehörten ein Syntax-getriebener Editor, eine Repository und ein COBOL Compiler sowie die damaligen Office Tools. Ich sollte mit meinen Leuten den Code-Transformator, den Code-Dokumentor, den Modultester und einen Testdaten-Editor liefern. Frau Nyary hat mit ihrem Team die Entwicklung des Testdateneditors übernommen. Mit diesem Tool sollte es möglich sein Dateien vom Mainframe herunterzuladen, zu verändern und wieder hochzuladen.

7.6.2 Die Umsetzung 25.000 JSP-COBOL Programme

Die Entwicklung des Transformators habe ich an Frau Erdös, die Entwicklung des Testdaten-Editors hatte ich an Erika Nyary delegiert. Ich selber habe mich auf den Dokumentor und den Modultestramen konzentriert. Frau Erdős übernahm einen Prototyp des Code-Transformators von mir. Statt eine Spezifikation zu schreiben, habe ich ein halbwegs funktionierendes Programm geschrieben, das die Aufgabe teilweise erfüllt. Einer der alten Szamalk Programmierer hat mir dabei geholfen. Das Prototyp-Programm sollte nur das „Proof of Conzept“ liefern. Wir konnten damit sowohl auf dem PC Arbeitsplatz als auch auf dem Unisys Mainframe einige Beispielprogramme in Standard COBOL-85 umsetzten. Es war fehlerhaft und schrecklich langsam, aber es zeigte, wie die Aufgabe zu lösen wäre. Frau Erdös konnte es übernehmen und binnen weniger Monate ein fehlerfreies Produkt machen. Mit diesem Tool konnte die Bank in den folgenden Jahren 25.000 ABACUS Programme transformieren und auf die IBM Anlage übertragen. Aus dem Smalltalk Projekt ist nie etwas geworden. Nach drei Jahren wurde es aufgegeben. Der produzierte Smalltalk Code blieb – wie der Thyssen Manager es treffend formulierte – auf Halden.

7.6.3 Auf den Erfolg folgt die Verabschiedung

Nichtdestotrotz war die ABACUS Migration ein großer Erfolg für die Bank. Sie konnten ihre Kern-Banking Systeme auf die IBM Anlage übertragen und die Unisys Mainframe abschalten. Leider war dieser Erfolg umstritten. Einige einflussreiche Bankmanager hätten sich eine andere Lösung gewünscht – eine verteilte Losung in einer modernen Programmiersprache, was immer das Wort „modern“ bedeutet. Die Übertragung der UBS COBOL Programme auf den IBM Rechner öffnete den Weg für die geplante Fusion der UBS mit dem Vereinsbank. Dieses Merger war schon lange im Gespräch, es fehlten jedoch bisher die technischen und kaufmännischen Voraussetzungen. Nach der Übertragung der Kern-Banking Systeme auf den IBM-

Host waren die technische Voraussetzungen erfüllt. Die beiden IT-Abteilungen wurden vereinigt. Das war aber für uns der Anfang vom Ende. In der neuen Organisation hatten wir keinen Platz mehr. In der politischen Auseinandersetzung um die Neubesetzung der Posten zog unser Schirmherr, der Herr O., trotz oder gerade wegen seines letzten Erfolges den Kürzeren. Es war wie damals in den 80er Jahren mit Herrn B. bei Bertelsmann. Beide waren starke Persönlichkeiten und haben dadurch viele Feinde gemacht. Am Ende waren viele Hunde des Wolfes Tod. Auch Frau von N. musste, wie man sagt, andere Aufgaben übernehmen. Eine neue Management-Generation ist angetreten. Mir wurde von dem Nachfolger von Herrn O. in Februar 1996 mitgeteilt, dass sie mich nicht mehr brauchen würden. Meine Arbeitsbewilligung für die Bank und auch die der Ungarn wurde nach 7 Jahren nicht mehr erneuert, Ich wurde aus dem Paradies verstoßen. Mag sein, dass ich in der einen oder anderen Aufgabe versagt hatte, aber meine Gesamtleistung war positiv. Die Testwerkzeuge, die wir gebaut hatten, wurden noch viele Jahre weiter benutzt.

7.7 Die Inder kommen

Nach dem Abschied von der UBS bin ich auf andere Schweizer Firmen ausgewichen. Ich durfte jedoch jetzt nur noch zeitweise ein- und ausreisen. Für die Züricher Versicherung habe ich noch AS400 Testwerkzeuge geliefert und die Tester dort in ihrer Nutzung geschult. Für Swissair habe ich zusammen mit Frau Nyary ihre Flugplanungsdaten von einem alten hierarchischen Datenbanksystem in eine relationale Datenbank migriert. Für eine Privatbank habe ich Tools geliefert um ihre Daten von einen netzartigen Datenbanksystem in eine Relationale Datenbank zu migrieren. Ich habe auch angeboten, ihre COBOL Programme zu migrieren, wir waren ihnen aber scheinbar zu teuer. Wir – die SES – verlangten weiterhin SFR 4,- pro umgesetzte Anweisung inclusive Test. Die Vertreter der Inder, die jetzt in der Schweiz auftraten, verlangten nur Sfr 1,- pro umgesetzte Anweisung ohne Test, testen sollten die Anwender selber. Die Inder hatten den großen Auftrag für die Migration der COBOL Programme erhalten – ich sollte die Mitarbeiter der Bank in Regressionstesttechniken schulen. Diese Schulung war ein einmaliges Erlebnis. Die Bank hatte inzwischen die ersten umgesetzten Programme von Indien zurückbekommen – die firmeneigene Entwickler sollten diese testen und warten. Sie waren nicht gerade glücklich über diese Rolle und mit den umgesetzten Programmen waren sie überhaupt nicht zufrieden. Vor allem waren sie auf ihre Manager wütend. Als ich zur Testschulung antrat, haben sie Papierflugzeuge auf mich geworfen. Ihre ganze Wut auf ihre Manager ha-

ben sie auf mich entladen. Die Atmosphäre war äußerst angespannt. Ich musste die Schulung abbrechen.

Eine zweite Enttäuschung war für mich die Schweizerische Bundesbahn. Als ich noch bei der UBS war, hatten sie mich nach Bern eingeladen, um eine Probemigration durchzuführen. Die SBB hatte die gleiche alte Unisys Entwicklungsumgebung wie die UBS – JSP-COBOL Programme mit DELTA Makros. Ihr Ziel war auch das Gleiche: sie wollten ihre Systeme auf eine IBM-Anlage übertragen. Dazu mussten sie ihren Code in Standard ANSI-COBOL umsetzen. Die JSP-Anweisungen mussten durch Standard COBOL Anweisungen ersetzt werden. Es wäre das gleiche Verfahren gewesen, wie bei der UBS. Wir hatten inzwischen die Werkzeuge um die Transformation voll automatisch durchzuführen. Wir haben für sie sogar mehrere Programme probeweise transformiert. Mein Angebot war alle COBOL Programme in strukturiertem ANSI-COBOL umzusetzen und zu testen für Sfr 4,- pro Anweisung. Die Bahn, bzw. die IT-Leiterin der Bahn, entschied den Auftrag an die Inder zu erteilen. Sie versprachen ein Band mit flag-free kompilierten Programmen zu liefern. Die Bahn soll sie selber testen. Dafür verlangten sie nur Sfr 1,- pro Anweisung. Das war für mich eine herbe Enttäuschung. Ich war aufgrund unserer Erfahrung bei der UBS sicher, wir würden den Auftrag bekommen.

Inzwischen habe ich erfahren, dass die Inder in die UBS eingezogen waren. Dort, wo wir früher gesessen sind, saßen jetzt Inder. Als ich bei der zweiten großen Schweizer Bank eigeladen war um dort einen Testseminar zu halten wurde mir endgültig klar, dass ich auf dem Gebiet der Reengineering nichts mehr zu suchen habe. Das Hotel am Züricher See, wo die Bank mit untergebracht hatte, war voll mit Inder. Beim Frühstück kam es mir vor, als ob ich ein Tourist in Neu Delhi wäre. Die Inder hatten die Wartung der Altsysteme in PL/I für diese Bank völlig übernommen. Für mich als Software Reengineer war in der Schweiz nichts mehr zu holen. Die Inder hatten mich vertrieben [Eber01].

8 Migrationsprojekte am Ende des Jahrhunderts

Zwischen meinem Exil aus der Schweiz in Frühjahr 1996 und meinem Neuanfang als Gastarbeiter in Wien im Frühjahr 1998 gab es zwei Jahre in den ich an diversen Projekten in Deutschland mitgewirkt habe. Manche dieser Projekte, wie das FISCUS Projekt, hatte ich schon vorher begonnen als ich noch in der Schweiz war. Inzwischen hatte sich in meiner Firma vieles geändert. Nach der Einstellung der SoftOrg Entwicklung sind die beiden ehemaligen Bundeswehroffiziere ausgeschieden. Die anderen deutschen Angestellten müsste ich auch freisetzen weil wir ohne SoftOrg keine Arbeit mehr für sie hatten. Im Jahre 1995 ist Frau Molnar in den Ruhestand gegangen und zog sich nach Budapest zurück. Die Büroleitung übernahm ein junger Betriebswirt – Oliver Foshag – der bei mir als studentische Hilfskraft eingestiegen war. Diese Zeit wurde von dem drohenden Jahrtausendwechsel geprägt. Viele Anwender wurden gezwungen ihre alte Software anzupacken und Sanierungen die sie lange Zeit vor sich hingeschoben hatten in Angriff zu nehmen. Besonders erwähnenswert war das Fiscus Projekt, das SKA Assembler Migrationsprojekt, das Multisprachen Projekt für die Bayerische Versorgungskasse, das COBOL Migrationsprojekt für das deutsche Auswärtige Amt, die Sparkassen Kapselungsprojekte und die Assembler Migration für die Bayerische Kommunalverwaltung. In allen dieser Projekte ging es darum, ein bestehendes System zu sanieren bzw. abzulösen. Das wir nicht mehr Projekte hatten lag vor allem an der Konkurrenz aus Indien. Ende der 90er Jahren sind die Inder auf dem deutschen Softwaremarkt erschienen und haben es für die heimischen Anbieter auf dem Gebiet der Software-Reengineering schwer gemacht.

Neben diesen Migrationsprojekten habe ich in dieser Zeit begonnen Softwaresysteme für Kunden zu vermessen. Mein erster Messungskunde war die niederländische PPT im Jahre 1995. Ich musste nach Gröhningen reisen um dort drei verschiedene Billing Systeme zu analysieren um deren Größe und Qualität zu vergleichen. Die Ergebnisse meiner Studie habe auf dem Schipol Flughafen vor der versammelten Projektmannschaft vorgetragen. Die Projektleiterin war mit dem Ergebnis zufrieden, vor allem damit, dass es mir gelungen ist die Qualität der Software in wenigen Zahlen auszudrucken [Kobi01].

8.1 Das FISKUS Projekt

8.1.1 Der erste Anlauf

FISKUS war ein großes Vorhaben der Bundesländer, ihr in die Jahre gekommenes Steuerverwaltungssystem durch ein einziges bundeseinheitliches System abzulösen. Ich spielte nur eine kleine Nebenrolle in diesem gigantischen Projekt der Projekte. Die Hauptrolle spielte Peter Hruschka der das Buch „Wien wartet auf Dich“ übersetzt hatte. Meine Aufgabe war es, die Datenmigration zu planen und die Qualität gewisser gelieferten Ergebnisse zu sichern. Dieses Projekt hatte ein hohes politisches Profil. Es waren demzufolge viele renommierte Software Hauser und viele profilierte Software Berater beteiligt, darunter viele Autoren von dem Objektspektrum und Redner von der jährlichen OOP Konferenz. Das Projektmitarbeiter-Verzeichnis ließ sich wie ein Who-is-who in der deutschen Informatik. Viele führende Experten der neuen Objekttechnologie angeführt von Dr. Hruschka von der Atlantic Guild und Martin Rösch von Rösch Consulting waren an Bord. Man würde meinen, mit so einer Mannschaft könne nichts schief gehen. Dies war schon die zweite Auflage des Projektes – der erste Anlauf hatte schon 1990 begonnen. Ein renommiertes deutsches Softwarehaus wurde beauftragt, eine Lösung vorzuschlagen und anschließend zu implementieren. Die Lösung wurde in Form eines monolithischen, strukturierten Systementwurfs geliefert. Aufs Papier schien alles gut zu sein, aber die ersten Versuche, den Entwurf in lauffähige Programme umzusetzen, scheiterten an die Inkompatibilität der Verfahren. Jedes Bundesland hatte andere Regel für die Steuerbewertung und ein anderes Verschlüsselungsschema für die Daten. Nichts passte zusammen. Statt gleich Programme zu generieren, hätte der Auftragsnehmer mit einem Metamodellierung der Daten und Prozesse beginnen sollen. In so einem Projekt haben eigentlich die Daten Vorrang.

Da die Projektverantwortlichen kalte Füße bekamen und außerdem behauptet wurde, die angewandte prozedurale Technologie sei ohnehin überholt, wurde entschieden, das Projekt abzubrechen und einen neuen Anlauf zu nehmen, diesmal verteilt und objektorientiert. Nicht nur das System, auch das Projekt wurde verteilt und zwar auf verschiedenen Auftragnehmern. Was ein einzelnes Software Haus nicht schafft hatte, sollten jetzt mehrere schaffen.

8.1.2 Der zweite Anlauf

Mein Auftraggeber war eine gewisse Frau Dr. G. vom Finanzministerium Bade-Württemberg. Sie kam aus der ehemaligen DDR und war den anderen Beamten aus der alten BRD um einiges überlegen. Später sollte sie mehrere Bücher schreiben und eine renommierte Projektberaterin werden. Ich hatte Glück für sie arbeiten zu dürfen. Für die Datenmigration hatte ich Ansprechpartner in dem Finanzämter München und Erfurt. Für die Qualitätssicherung waren meine Partner in Freiburg im Breisgau sowie im Finanzamt Hessen in Wiesbaden.

8.1.3 Die FISCUS Datenmigration

Was die Datenmigration anbetraf, war ich bemüht wegen der unterschiedlichen Datenbanksysteme in den einzelnen Ländern eine gemeinsame Zugriffsschicht zu schaffen. Die Sicht der Systeme auf die Daten sollte völlig unabhängig von den physischen Speicherungsstrukturen sein. Es sollte möglich sein, sowohl auf hierarchische und netzartige als auch auf relationale und objektorientierte Datenbanken zuzugreifen. Die Datenbanken sollten gekapselt werden. Die Client-Programme sollten eine Standard ODBC – Open Database Connection – Schnittstelle benutzen. Auf der Serverseite sollte für jeden Datenbanktyp eine eigene Zugriffsschale bereitgestellt werden. Für das Land Bayern mit seinem Siemens Rechner habe ich eine SESAM Zugriffsschicht, für Baden-Württemberg mit seinen IBM Rechner einen DB2 Zugriffsschicht – beide in C++ -- entwickelt. Als Verbindungstechnologie habe ich den Common Request Broker Orbix von der IONA Technologies mit der Sprache IDL verwendet.

Zu diesem Zeitpunkt hatte ich begonnen mich mit der Objekttechnologie intensiv auseinander zu setzen. Bisher hatte ich auf dem PC nur in COBOL programmiert. Die Fiscus Zugriffskomponente habe ich in C++ implementiert. Es dauerte lange, bis ich diese neue Programmiersprache ausreichend beherrschte. Seit dem habe ich großes Verständnis für Programmierer, die ab 50 eine neue Programmiersprache erlernen mussten. Das menschliche Gehirn ist nicht so flexibel, wie manche Optimisten behaupten.

Nach einigen Versuchen haben meine Zugriffsschichten endlich prototypmäßig funktioniert. Es wurde in Stuttgarter Finanzministerium ein Probeprojekt damit gemacht. Ich hatte schon begonnen über das Thema ein Buch mit dem Titel „Objektorientierte Softwaremigration“ zu schreiben. Zwei Jahre später erschien es in Addison-Wesley Verlag. Die Kapselungstechnik – sowohl Datenkapselung als auch Programmkapselung – spielte darin eine große Rolle. Später habe ich erkennen müssen, dass Kapselung nur eine vorübergehende Lösung ist, denn das alte Datenbanksystem bzw. die alten Programme bleiben, und müssen weiter gewartet bleiben. Jedenfalls habe ich aus dem CORBA Integrationsteilprojekt vieles gelernt das mir später beim Umstieg auf die SoA Technologie geholfen hat.

8.1.4 Die FISCUS Qualitätssicherung

Mein anderes Teilprojekt – die Qualitätssicherung der Anwendungskomponente – ist nie so weit gekommen. Ich begann das Teilprojekt textbuchmäßig mit einer Risikoanalyse. Ich habe ein Formular für die Bewertung und Gewichtung der Projektrisiken von einem amerikanischen Fachartikel zu diesem Thema übernommen und an die

anderen Projektleiter verteilt, mit der Bitte sie auszufüllen. Ich habe unterschieden zwischen organisatorischen, personellen, konzeptionellen und technischen Risiken. Für jedes Risiko musste der Projektleiter die Wahrscheinlichkeit, die Auswirkung und die mögliche Einschränkung schätzen. Die Ergebnisse der Umfrage waren verheerend. Aus der Sicht der Teilprojektleiter gab es so viele schwerwiegende Risiken, dass man das Projekt sofort hätte einstellen müssen. Stattdessen lief dieser Anlauf noch zwei Jahre weiter und verschlang mehrere Millionen Mark bis das Finanzministerium entschied es endlich abzubrechen und einen neuen Ablauf zu nehmen. Bis es so weit war lagen die Ergebnisse meiner Risikoanalyse in der Schublade. Man wollte keine pessimistische Stimmung in dem Projekt verbreiten.

FISCUS-Risikoanalyse (Interne Kommunikation)

RISIKOANALYSE TOP 10 CHARTS					
Projekt-Name:		Interne Kommunikation			
Autor:		Dr. Winklmayer/Hr. Ferken/Hr. Weiser			
Firma/Abt.:		OFD Freiburg			
Erfassungsdatum:		24.08.1995			
Textreferenz:		WinWORD 6.0: Risk583t.doc			
Zusätzliche Hinweise:					
Risiko Rang	**Risiko-beschreibung**	**Wahrschein-lichkeit in %**	**Risiko-aussetzung**	**Risiko-faktor**	**Gegenmaßnahmen**
1	Mangelnde Qualifikation der Mitarbeiter	50	70	35	Schulung, Neueinstellungen, Fremderfahrung
2	Fehlende Erfahrung/Schulung	50	50	25	Reihenfolge: OO allgemein, CORBA, OO-Sprachen, Programmierung ORB's, Schulung Services
3	Instabilität der Produkte	40	50	20	Testen, Qualitätssicherung, Alternativprodukte
4	Verfügbarkeit von Produkten und Funktionalität	35	30	11	Alternativprodukte
5	Performance-Grenzen	30	30	9	Gutes Design, Ausnutzung der Verteilungseigenschaften
6	Mangelnde Normierung	15	60	9	Einwirkung auf Hersteller, Test
7	Herstellungsabhängigkeit	30	20	6	Einschränkung der Plattformen
8	Komplexitätsabhängige, lastabhängige Performance-Probleme	30	20	6	Organisatorische Maßnahmen, dynamische Verteilung
9	Viele Lieferanten mit unterschiedlichen Interessen	20	30	6	Abnahmetest, Delegationsprinzip
10	Unverträglichkeit der Teilkomponenten	20	30	6	Testen, Prototypen, Druck auf Hersteller

Inzwischen hatte ich eine Qualitätsrichtlinie für das Projekt verfasst, in dem ich alle abzuliefernden Ergebnisse identifizierte und in welcher Form sie zu liefern waren, beschrieb. Von den Anforderungsspezifikation bis zum Abnahmetestprotokoll wurde alles exakt beschrieben mit Musterbeispielen und Checklisten, alles zugeschnitten auf die Objekttechnologie mit Anwendungsfällen, UML-Diagrammen, C++ Codebeispielen und Objekttestskripten. Ich hatte auch begonnen, die passenden Prüf- und Test-

werkzeuge zu sammeln und zu integrieren. Leider gab es kaum Ergebnisse zu prüfen. Das Projekt ist nicht über die Anforderungsphase hinaus gekommen. Die Länder konnten sich über die Anforderungen nicht einigen. Diese zweite Anlauf wurde nach zwei Jahren beendet und damit auch mein Auftrag. Ich war sehr traurig darüber, denn das FISCUS Projekt war ein großes Spielfeld nicht nur für mich sondern auch für die anderen Zuliefern. Man konnte alle möglichen neuen Technologien ausprobieren mit der Gewissheit, dass sie nie zum Einsatz kommen würden. Die Stakeholder, in diesem Fall die Bundesländer, hatten kein echtes Interesse dran das Projekt umzusetzen.

8.1.5 Das FISCUS Projekt wird aufgegeben

Nachdem ich aus dem FISCUS Projekt ausgeschieden bin, wurde ein neuer Ablauf vorgenommen. Statt mit einer Vielzahl einzelnen Auftraggeber zu arbeiten, hat das Finanzministerium eine eigene Firma in Wiesbaden gegründet, die das ganze Projekt stemmen sollte. Jetzt war alles in einer Hand. Ich weiß nicht, ob es wirklich besser gelaufen ist. Jahre später, es war 2004, habe ich aus der Zeitung entnommen, dass das Fiscus Projekt begraben wurde und zwar mit einer Verlust von rund 900 Millionen Euro [FAZ12]. Die Bundesländer sollten ihre alten Steuerverwaltungssysteme behalten oder sie selber migrieren. Ganz um sonst war das Projekt jedoch nicht. Wie 20 Jahre davor, das integrierte Transportsteuerungssystem der Bundesbahn – I.T.S. - diente das FISCUS Projekt als Schule der Nation. Etliche deutsche Softwarehäuser haben dort ihre Kenntnisse über die neue verteilte IT vertieft. Zahlreiche Fachartikel und mehrere Doktorarbeiten sind aus dem Projekt hervorgegangen, nicht zu schweigen von den vielen Softwarewerkzeugen die entwickelt wurden. Die Technologien die auf Kosten der Steuerzahler speziell für dieses Projekt entwickelt worden, sind später anderen Projekten in der Industrie zu Gute gekommen. Wie damals das I.T.S. Projekt, diente das FISCUS Projekt dazu, die deutsche Softwareindustrie zu fordern. Insofern wurde das Geld nicht ganz verloren. Ich habe jedenfalls viel von dem Projekt profitiert. Es war für mich der Einstieg in die Objekttechnologie.

Einige Jahre später hat Prof. Peter Mertens von der Uni Erlangen-Nürnberg ein viel zitierten Beitrag in der Wirtschaftsinformatik über gescheiterte Projekte im öffentlichen Bereich veröffentlicht [Mert12]. FISCUS war ganz vorne dran. Es sieht so aus, als ob große Softwareprojekte im öffentlichen Bereich nur geringe Erfolgschancen haben. Woran das liegt wird noch erforscht. Es liegt bestimmt nicht an der Fähigkeit der Beteiligten noch an ihren Arbeitswillen. Ich habe in diesem Projekt viele qualifizierten und engagierten Mitarbeiter im öffentlichen Dienst kennengelernt. Die Ursachen der Probleme liegen eher in der Organisation der Projekte.

8.2 Das SKA Assembler Migrationsprojekt

Die Schweizer Kreditanstalt hatte eine Tochtergesellschaft in Deutschland mit Sitz in Frankfurt. Diese hatte eine Menge alter IBM Assembler Programme. Für den herannähernden Jahrtausendwechsel müssten die Entwickler dort sich in diese alten Programme einarbeiten um die Datumsfelder und Datumsfeld-Operationen zu verändern. Dies stellte sich als problematisch heraus, weil es nur noch einen einzigen Entwickler gab, der mit Assembler vertraut war. Für die Veränderung der Assembler Programme hätten sie externe Kapazität zu teurerem Geld reinholen müssen. Stattdessen haben sie entschieden, diese Programme – es waren nur 19 an der Zahl mit knapp über 10.000 Assemblerzeilen – in COBOL umzusetzen und anschließend der COBOL Code Jahr 2000 gerecht zu richten. COBOL Entwickler hatten sie genüg.

Assembler Metrics before Conversion

PROCEDURE SIZE METRICS	
NUMBER OF STATEMENTS	7521
NUMBER OF MODULES (CSECTS)	9
NUMBER OF BRANCHES	2933
NUMBER OF GO TO BRANCHES	1254
NUMBER OF SUBROUTINE CALLS	207
NUMBER OF MODULE CALLS	19
NUMBER OF DATA REFERENCES	4961
NUMBER OF FUNCTION-POINTS	55
NUMBER OF LINES OF CODE	9994
NUMBER OF COMMENT LINES	1093
COMPLEXITY METRICS	
DATA COMPLEXITY	0,743
DATA FLOW COMPLEXITY	0,892
INTERFACE COMPLEXITY	0,421
CONTROL FLOW COMPLEXITY	0,617
DECISIONAL COMPLECITY	0,509
INTERNAL COMPLEXITY	0,636
QUALITY METRICS	
MODULARITY	0,339
PORTABILITY	0,214
MAINTAINABILIT	0,364
TESTABILITY	0,472
CONFORMITY	0,702
INTERNAL QUALITY	0,418

COBOL Metrics after Conversion

PROCEDURE SIZE METRICS	
NUMBER OF STATEMENTS	8271
NUMBER OF MODULES (SECTIONS	39
NUMBER OF BRANCHES	2673
NUMBER OF GO TO BRANCHES	1142
NUMBER OF PERFORMS	357
NUMBER OF MODULE CALLS	19
NUMBER OF DATA REFERENCES	5174
NUMBER OF FUNCTION-POINTS	55
NUMBER OF LINES OF CODE	10759
NUMBER OF COMMENT LINES	1579
COMPLEXITY METRICS	
DATA COMPLEXITY	0,744
DATA FLOW COMPLEXITY	0,811
INTERFACE COMPLEXITY	0,529
CONTROL FLOW COMPLEXITY	0,506
DECISIONAL COMPLECITY	0,421
INTERNAL COMPLEXITY	0,602
QUALITY METRICS	
MODULARITY	0,396
PORTABILITY	0,389
MAINTAINABILITY	0,398
TESTABILITY	0,467
CONFORMITY	0,715
INTERNAL QUALITY	0,473

Von unseren Reengineering Projekten bei der UBS in Zürich hatten sie schon gehört. In Zürich hätte ich für sie nicht arbeiten dürfen aber jetzt war ich von der UBS weg. Sie luden mich nach Frankfurt ein, um die Programme anzuschauen. Der Einladung bin ich gefolgt und habe die Assembler Programme an Ort und Stelle in einem Tag mit dem Assembler Audit analysiert. Ich zählte die Codezeilen, Anweisungen,

Data-Points und Function-Points und machte ihnen am selben Tag ein Angebot, die Programme zu einem festen Preis in COBOL umzuwandeln. Ich wusste, dass wir das richtige Werkzeug und das richtige Personal hatten um das Problem für sie zu lösen. Kati Erdös hatte gerade mehrere alte Assembler Programme von der UBS in Zürich umgesetzt und war jetzt frei für dieses Projekt.

In diesem Falle passte das Werkzeug „ASMRecon“ sehr gut zu dem IBM-Assembler Code. In nur wenigen Tagen in Frankfurt konnte Frau Erdös mehr als 80% der Assembler Anweisungen umsetzen. Den Rest hat sie in den folgenden zwei Monaten manuell übersetzt und alle 19 Programme getestet. Ich habe die konvertieren COBOL Programme mit COBRecon restrukturiert und einen großen Teil der GOTO Verzweigungen entfernt. In diesem Projekt habe ich zum ersten Mal die Qualität des Codes vor und nach der Migration gemessen. Die Zahlen belegten, dass die Programme weniger komplex und qualitativ hochwertiger waren. Die Komplexität sank von 0,636 auf 0,602. Die Qualität stieg von 0,418 auf 0,473. Dies zeigte, dass signifikante Verbesserungen zum Code von einer automatischen Transformation nicht zu erwarten sind. Die Steuerungskomplexität ist durch die Entfernung der GOTO Verzweigungen von 0,617 auf 0,506 herabgesetzt und die Modularität ist von 0,339 auf 0,396 gestiegen, aber Kontrollflusskomplexität und Modularität sind nur einzelne Eigenschaften und nicht die Gesamtsicht. Wir waren vorsichtig, zu viel an dem Code zu verbessern. Mit jeder Verbesserung steigt das Fehlerrisiko. Unser Hauptziel war dass die neuen COBOL Programme mit den alten Assembler Programmen funktional äquivalent waren. Dieses Ziel haben wir auch erreicht. Die 39 COBOL Module lieferten die gleichen Ergebnisse wie die ursprünglichen 19 Assembler Programme nur etwas langsamer, wie es sich herausgestellt hat. Das ist unumgänglich wenn man maschinennahe Programme in eine höhere, abstraktere Sprache umsetzt [Sned96].

8.3 Migrationsplanung für die bayerische Versorgungskasse

Während Frau Erdös in Frankfurt mit den alten Assembler Programmen der SKA beschäftigt war, trat ich mit Frau Nyary ein neues Migrationsprojekt bei der Bayerischen Versorgungskasse an. Dieses Projekt stellte eine besondere Herausforderung, weil diese staatliche Versorgungsanstalt für bestimmte privilegierte Stände im Dienst des Bayerischen Königshauses fünf größere Applikationen hatte, in denen drei verschiedene Sprachen vorkamen – hauptsächlich Assembler Code, aber auch COBOL und PL/I. Meine Firma war zu dieser Zeit die einzige Firma in Deutschland mit Werkzeugen für alle drei IBM Sprachen. Deshalb kam die Betriebsleitung auf uns zu. Sie wollten erfahren ob eine automatisierte Migration für sie in Frage käme. Unser neuer Büroleiter – der Herr Foshag – hat dieses Geschäft eingefädelt. Wir haben, wie

immer, mit einer Vermessung des Codes angefangen. Die fünf Applikationen enthielten des Codes angefangen. Diese Applikationen enthielten insgesamt

```
 1.272 Assembler Module mit 1.050.327 Anweisungen
 1.070 Assembler Makros mit    36.758 Datenanweisungen
    67 COBOL Module     mit    22.813 Anweisungen
    44 COBOL Copies     mit     5.441 Datenanweisungen
    81 PL/I Module      mit    39.469 Anweisungen
    37 PL/I Includes .  mit     2.112 Datenanweisungen
_____________________________________________________
 2.571 Module            mit 1.156.920 Anweisungen insgesamt
```

Die mittlere Komplexität des Assembler Codes lag bei 0,521 also etwas über den Durchschnitt. Die mittlere Qualität des Assembler Codes lag bei 0,322, also weit unter dem Soll. Der Assembler Code wäre demzufolge besonders schwer zu pflegen.

Bei dem COBOL Code sah es nicht ganz so schlimm aus. Wir konnten mit COBRecon die meisten GOTO Verzweigungen entfernen. Die mittlere Komplexität der 67 Programme betrug 0,498, die mittlere Qualität 0,491, Die niedrigere Qualität war vor allem eine Folge der zu großen Module, der zu tief verschachtelten Prozeduren und der festdrahteten Daten.

Der PL/I Code war einfach schlecht implementiert. Die mittlere Komplexität war mit 0,447 akzeptable, aber die mittlere Qualität mit 0,398 war viel zu niedrig. Der PL/I Code war voll mit GOTO Verzweigungen was für PL/I ungewöhnlich ist.

Jedenfalls war es schwierig mit dem alten Code umzugehen. Nichtdestotrotz konnte Frau Nyary mit ihrem Team sämtliche Programme nachdokumentieren und eine zentrale Repository aufbauen. Zu jedem Programm, auch zu den Assembler Programmen wurde ein word .doc File erstellt mit den Ein- und Ausgabedaten, den Unterprogrammen und der Entscheidungslogik in Form von Pseudo Code. Der Kunde bekam also eine komplette technische Spezifikation seiner Programme.

Der nächste Schritt wäre es gewesen, die ersten Applikationen abzulösen. Der Kunde stand vor der Wahl

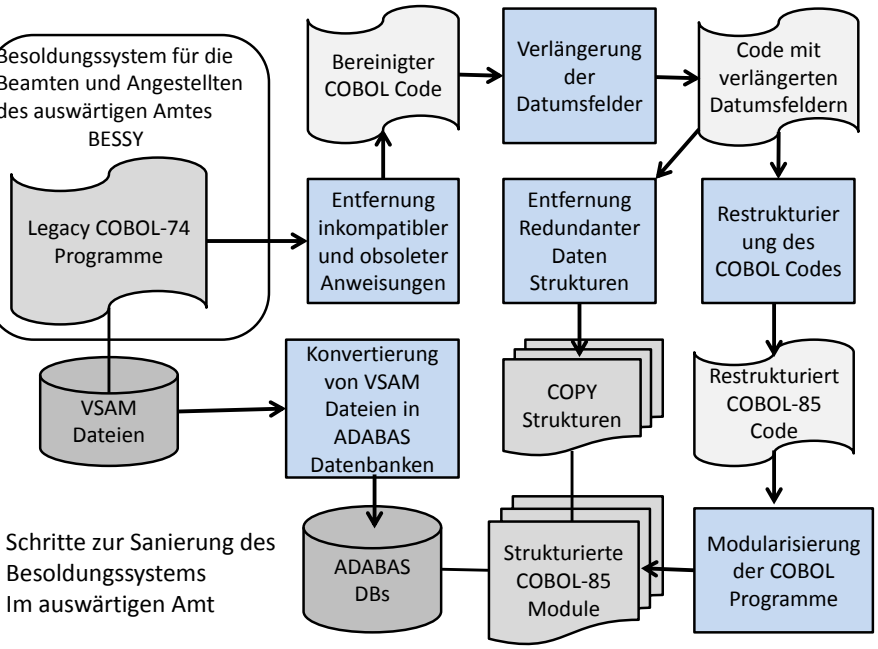
- die Assembler und PL/I Systeme alle automatisch in COBOL umzuwandeln und die bestehenden COBOL Programme nachzubessern oder
- die Assembler und PL/I Systeme manuell aufgrund der Nachdokumentation in COBOL zu reimplementieren oder
- alle Systeme mit einer neuen Sprache wie Natural von der Software AG neu zu entwickeln oder

- die individuelle Anwendungssysteme durch Standard Software von der SAP abzulösen.

Ich habe auch hier eine Risikoanalyse und eine Schätzung für alle drei Alternativen durchgeführt. Nach langem hin und her hat sich der Kunde endlich für die SAP Losung entschieden. Wir waren damit aus dem Spiel. Ich hatte getan was ich konnte, um den Weg zur Entscheidungsfindung zu bahnen. Die Entscheidung war völlig richtig. Es wäre unwirtschaftlich gewesen die alte Software in das neue Jahrhundert mit hineinzuschleppen. Bis sie zusammen mit SAP die Ablösung begannen, war ich schon längst in Wien.

8.4 COBOL Reengineering für das deutsche Auswärtige Amt

Das deutsche Auswertige Amt hatte schon Ende der 70er Jahren ein System namens BESSY in COBOL-74 entwickelt auf der Basis einer ADABAS Datenbank, um die Beamten und Angestellte des Auswärtigen Dienstes einschließlich des Goethe Instituts im In- und Ausland zu bezahlen. Später kamen Natural Programme hinzu, um die ADABAS Datenbanken in Online-Betrieb zu pflegen.



Die 67 COBOL-74 Batchprogramme mit 324 Kilo Codezeilen waren vom Anfang an nicht optimal implementiert. Sie waren viel zu groß, das Durchschnittprogramm hatte mehr als 4000 Codezeilen, manche sogar bis zu 10.000. der Datenfluss war sehr komplex. Die Programme griffen nicht nur auf mehrere ADABAS Datenbanken, sie verarbeiteten auch zahlreiche sequentielle und index-sequentielle Dateien. Die Entscheidungslogik war wegen der vielen Ausnahmen für das Personal im Ausland äußerst verzwickt und mit GOTO Anweisungen implementiert. Ein großer Teil der prozeduralen Anweisungen waren GOTO Anweisungen. es war daher sehr schwer die Programme zu ändern und es wurde vom Jahr zu Jahr immer schwieriger. Eine kleine Änderung nahm mindestens 5 Personentage in Anspruch und die Änderungshäufigkeit nahm immer mehr zu. Die Tarife für die Vergütung wurden ständig geändert. Die zwei Entwickler, die für die Pflege des Codes verantwortlich waren. waren total überfordert. Einer musste nebenbei auch die National Programme pflegen.

Jetzt kam das heranrückende Jahr 2000 Problem und außerdem noch der bevorstehende Standortwechsel nach Berlin. Das Amt musste sich von der alten DOS-VSE Maschine trennen. Mit der COBOL-74 Dialekt war dies nicht so einfach. Es gab viele IO-Anweisungen die auf diese Maschine zugeschnitten waren, z.B. die Druck Operationen. Ein Wechsel auf COBOL-85 war unvermeidlich. Die Datumsfelder waren alle nur 6-stellig. Sie mussten wegen dem neuen Jahrtausend auf 8 Stellen erweitert werden. Gerade in solchen Lohn und Gehaltssysteme spielt der jeweilige Datum eine zentrale Rolle.

Es gab also zwingende Gründe den COBOL Code zu sanieren und es blieb nicht allzu viel Zeit dazu. Es musste schnell gehandelt werden. Es wurden sechs Sanierungsziele mit unterschiedlicher Priorität festgelegt:

- als erstes sollten die alten inkompatiblen Anweisungen entfernt und der Code konvertiert werden,
- als zweites sollten die verbliebenen VSAM Dateien durch ADABAS Datenbanken abgelöst werden,
- als drittes sollten die Datumfelder von 6 auf 8 Stellen erweitert werden,
- als viertes sollten die redundanten COBOL Datenstrukturen in COPY Strecken ausgelagert werden,
- als fünftes sollten die COBOL Programme durch Refaktoring aufgeteilt werden.
- als sechstes sollte die Entscheidungslogik vereinfacht und restrukturiert werden.

Nach jeder Stufe sollte das System voll operationsfähig sein, d.h. nach jeder Stufe musste es durch einen Systemtest durch das Wartungspersonal und einen Abnahmetest durch die Sachbearbeiter in der Personalabteilung gehen.

8.4.1 COBOL Sprachaktualisierung

Die Sprachaktualisierung wurde mit Hilfe des COBRecon Tools automatisch durchgeführt. Dazu gehörte

- die Entfernung der Datums und Condition-Code Register für die Kommunikation mit der übergeordneten Jobsteuerungsprozedur. Dieser Steuerungsmechanismus wurde auf andere Weise gelöst
- die Entfernung obsoleter COBOL Anweisungen
- die Anpassung der File Selection Anweisungen
- die Veränderung der Drucksteuerung.

Nach dieser Sprachaktualisierung, die nur drei Tage für die Vorbereitung und einen Tag für die Durchführung gedauert hat, konnten die 67 COBOL Programme mit dem neuen COBOL-2 Compiler kompiliert werden. Nach dem Test mit Datenabgleich zu den alten Dateien gingen die konvertierten Programme in die Produktion.

8.4.2 VSAM Dateikonversion

Schon vor unserem Reengineering Projekt wurden die VSAM Dateien von der Software A.G. in ADABAS Datenbanken konvertiert. Es blieb unsere Aufgabe, die COBOL IO-Anweisungen - wie OPEN, WRITE, REWRITE, DELETE und CLOSE – durch entsprechende ADABAS Exec Makros zu ersetzen. Dafür hatte ich ein Mustermakro für jede IO-Operation. Das Reengineering Tool musste nur die variablen Namen in dem Makro durch die ADABAS Tabellen, Satz- und Schlüsselnamen ersetzen und das Makro in den Code einfügen. Nachdem ich die Mustermakros entworfen hatte, dauerte es nur wenige Tage die alten VSAM Dateioperationen zu ersetzen. Danach gingen die Programme wieder durch den Testprozess. Diesmal musste der Inhalt der neuen ADABAS Tabellen mit den alten VSAM-Dateien verglichen werden. Bis auf einige Verschiebungen passten die Daten auf Anhieb.

8.4.3 Vergrößerung der Datumsfelder

Die Vergrößerung der Datumsfelder war viel aufwendiger als die bisherigen Transformationen. Deshalb habe ich sie an das Team von Frau Nyary delegiert. Zuerst müssten die Datumsfelder überhaupt erkannt werden. Über die Namen war dies nicht immer möglich, man musste auch die Verwendung der Felder analysieren um sie einzuordnen. Dies ist dem Reengineering Team mit Hilfe der Source-Analyse Werk-

zeuge letztendlich gelungen, aber das Problem mit den kurzen Datumsfelder blieb in den Datenbanken. Beim Lesen und Schreiben der Daten mussten die Datumsfelder immer umgesetzt werden. Die Erweiterung der Datenbankfelder erfolgte später.

8.4.4 Auslagerung redundanter Datenstrukturen

In dem BESSY System des Auswärtigen Amtes wurden die gleichen Dateien von verschiedenen Programmen verarbeitet. Jedes Programm hatte eine eigene Beschreibung der Daten, je nachdem, wie es die Daten verwendet hat. Es konnte vorkommen, dass ein Feld das als Nummer in dem einen Programm als Zeichenfolge im anderen Programm definiert war. Im Falle einer Veränderung der Datenstruktur - wie dies bei der Datumserweiterung der Fall war - kam es oft zu Fehlern. Die unterschiedlichen Datensichten waren nicht synchronisiert. Aus diesem Grund sollte es pro Datei bzw. Datenbanksatz nur eine Beschreibung geben, die von allen Programmen geteilt wird. Diese eine Beschreibung wurde in ein eigenes Quelltext abgelegt, das bei der Kompilierung in das jeweilige Programm inkludiert wird. Das Problem mit unterschiedlichen Typen und Strukturen für die gleichen Felder wurde mit Redefinition in der einen Datenbeschreibung gelöst. Mit dem SofReCon Tool wurde diese Generierung einer einheitlichen Datensicht automatisch bewerkstelligt.

8.4.5 Refaktorierung der COBOL Programme

Die COBOL Programme in BESSY waren gute Beispiele für die Evolutionslehre. Sie begannen als eine überschaubare Anreihung einzelne Codeblöcke, mit jeder Menge Sprünge hin und her zwischen den Blöcken. Ursprünglich hatte ihre Anreihung einen Sinn gehabt. Später, im Laufe der Zeit ging dieser Sinn verloren. Es wurden immer mehr Ergänzungen zu den Abrechnungsregeln gemacht und neue Codeblöcke wurden willkürlich eingefügt. Programme, die nach der ersten Entwicklung weniger als 3000 Anweisungen hatten, hatten nach 10 Jahre Weiterentwicklung mehr als 6000 Anweisungen. Sie sind um 100% oder mehr gewachsen. Entsprechen schwer war es sie zu verstehen, zu ändern und zu korrigieren. Das hätte mit der Sprache nichts zu tun gehabt. Die Sprache COBOL bietet genug Möglichkeiten, den Code aufzuteilen. Nur diese Möglichkeiten wurden in der Eile des Änderungsdienstes nie wahrgenommen. Also blieb es bei mir übrig, die Programme aufzuteilen. Zum Glück hatte ich schon die Erfahrung mit „Program Downsizing“ in dem Ploenzke Projekt gemacht.

Wegen der vielen GOTO Verbindungen zwischen den Codeblöcken war dies alles andere als einfach. Ich musste SECTIONS in dem Code bilden und die Verzweigungen zwischen den SECTIONS kappen. Die GOTO Sprünge wurden durch PERFORM Aufrufe ersetzt. In einigen Programmen habe ich übergroße Codeabschnitte in Unter-

programme ausgelagert. Das Problem hier war die Daten. Da im COBOL der Datenaustausch zwischen Module nur über eine Parameterliste geht, musste ich neue Datensätze definieren, die ich als Transportbehälter für Parameter verwendet habe. Diese Container wurden von einem Unterprogramm zu anderen weitergereicht. Jedes Unterprogramm erhielt eine eigene Sicht auf den Inhalt des Containers und konnte den Inhalt verändern. Die Datensichte habe ich aus der Datenverwendungsanalyse entnommen. Später wurde dieser Container-Ansatz zum Doktrin der verteilten Systementwicklung. Damals war er unbekannt [Roß15].

Auf dieser Weise ist es mir gelungen aus den 67 COBOL Programme 93 Programme abzuleiten und aus den zwei oder drei Abschnitten - Sections - per Programm wurden 50 und mehr. Die Anzahl der elementaren Codeblöcke blieb jedoch gleich. Sie wurden bloß umverteilt.

8.4.6 Restrukturierung der COBOL Programme

Der letzte Schritt in diesem Reengineering Prozess war die Restrukturierung der Ablauflogik. Die GOTO Verbindungen zwischen Sections hatte ich bereits bei der Refaktorierung gekappt. Jetzt wollte ich versuchen, die GOTO Verzweigungen innerhalb der Sections zu entfernen. Ich habe damit angefangen, bin aber leider nicht sehr weit gekommen. Die Lösung dieser Aufgabe war nur bedingt automatisierbar. Hinzu kam, dass die verbleibenden COBOL Entwickler sie ablehnte. Ihnen waren ihre GOTO Sprünge lieber als die tief verschachtelte IF und LOOP Strukturen. Also habe ich aufgegeben dieses Ziel weiter zu verfolgen. Am Ende scheitern die meisten guten Ansätze in der Softwareentwicklung an den Menschen.

Nichtdestotrotz wurde das BESSY Reengineering Projekt eins meiner erfolgreichsten. Im Jahre 1999 auf der Internationale Maintenance in Oxford habe ich einen Beitrag mit dem Titel „Salvaging an Ancient Legacy System at the German Foreign Office“ berichtet [SnNy99].

8.5 Assembler Migration bei der bayerischen Kommunalverwaltung

Auf einem Seminar in München habe ich einen Manager von der bayerischen Anstalt für kommunalen Datenverarbeitung – AKDB - kennengelernt. Die AKDB bietet den bayerischen Kommunen IT-Dienstleistungen an, u.a. Systeme für die Verarbeitung ihrer Gebühren. Sie waren inzwischen bei der Software AG gelandet und setzten hauptsächlich Natural/Adabas für die Online-Verarbeitung und COBOL für die Batchverarbeitung ein. Eins der Batchprogramme blieb jedoch in der IBM-

Assemblersprache, ausgerechnet das System für die Abrechnung der Wassergebühren. Der Jahrtausendwechsel näherte sich und es gab in der AKDB keine Assembler Entwickler mehr, die die Assembler Programme hätten anpassen können. Die neueren Batchprogramme waren in COBOL implementiert. Demzufolge hat das AKDB Management entschieden, die Wassergebührenabrechnung noch vor den Jahrtausendwechsel von Assembler in COBOL umzusetzen damit sie die Datumsfelder noch rechtzeitig verändern konnten. Die Aussendung der Wasserrechnungen wurde über das Datum gesteuert.

Automatisierter Assembler to COBOL Transformation

```
COB   Z226U26.                                                          00011350
ASM  *         CLI   T025F041,0                                         00011360
ASM  *         BE    Z226U263               Backward Jump ===>>       00011370
COB       IF T025F041 = X-0                                             00011380
COB           GO TO Z226U263 .                                          00011390
ASM  *         L     R10,SAVA                                          00011400
COB       MOVE SAVA TO R10 .                                            00011410
ASM  *         L     R5,WT025A                                         00011420
COB       MOVE WT025A TO R5 .                                           00011430
ASM  *Z226U262 DS    0H                                                00011440
COM  *_________________________________________________________________00011450
COB   Z226U262.                                                         00011460
ASM  *         C     R5,WT025                                          00011470
ASM  *         BNL   Z226U29                                           00011480
COB       IF R5 NOT < WT025                                             00011490
COB           GO TO Z226U29 .                                           00011500
COM  *                                     Forward  Jump ===>>        00011510
ASM  *         CLC   GEBA1,T025F03          GLEICHER TARIF ?           00011520
ASM  *         BNE   Z226U261                                          00011530
COB       IF GEBA1 NOT = T025F03                                        00011540
COB           GO TO Z226U261 .                                          00011550
COM  *                                      Forward  Jump ===>>       00011560
ASM  *         MVC   FEHLTEXT(L'Z22FEHL1),Z22FEHL1      KEINE BESTAB.  00011570
COB       MOVE Z22FEHL1 TO FEHLTEXT(L-Z22FEHL1).                        00011580
ASM  *         MVC   FEHLTEXT+L'Z22FEHL1-4(L'T025F03),T025F03           00011590
COB       MOVE (L-T025F1:03) TO FEHLTEXT(L-Z22FEHL1-.                   00011600
ASM  *         MVI   FEHLTEXT+L'Z22FEHL1,C'1'                          00011610
COB       MOVE X-1:1) TO FEHLTEXT(L-Z22FEHL1:.                          00011620
ASM  *         BAL   R14,Z22PROT                                       00011630
```

In diesem System gab es ein großes Hauptprogramm mit über 30.000 Assembler Anweisungen und 8 Unterprogramme mit weiteren 15.000 Anweisungen. Für die Berechnung der Wassernutzung und der Umsetzung der Gebühren wurden große Tabellen im Kernspeicher verwendet, die den gleichen Speicherplatz belegten, d.h. die Programme nutzten eine Overlay Technik mit Pointers um die Daten zu adressieren. So wurden auch die Tabellen von dem Hauptprogramm an die Unterprogramme übergeben. Die Felder in den Tabellen wurden über eine Basisadresse mit Absetzung in den prozeduralen Anweisungen identifiziert. Die kleinste Verschiebung hätte alles

umgeworfen. So kam es auch, dass seit Jahren die falschen Tarife für den Wasserverbrauch in einigen Gemeinden benutz wurde, ohne dass es jemand gemerkt hatte.

Unser Werkzeug für die automatische Umsetzung der Assembler Befehle war in diesem Umfeld nur bedingt brauchbar. Die prozeduralen Anweisungen konnten 1:1 in COBOL Anweisungen umgesetzt werden, aber die komplexen Datenstrukturen mussten manuell neu definiert werden. Auch die prozeduralen Anweisungen mussten manuell nachgebessert werden. Als ich bemerkt habe, wie schwierig die Aufgabe war, habe ich mich zurückgezogen und die Arbeit der Frau Erdös überlassen. Sie war gerade mit der SKA Assembler Konversion in Frankfurt fertig geworden. Die COBOL Übersetzung für dieses Projekt konnte sie in Budapest erledigen, aber die Test- und Korrekturarbeit musste sie in dem AKDB Büro in München ausführen und das war der Löwenanteil der Zeit.

Das wäre nicht so schlimm gewesen aber es kam ein weiteres Problem hinzu. Der Auftritt der Inder auf dem deutschen Markt hatte Konsequenzen für uns. Die Preise für solche Sanierungsprojekte wurden überall nach unten gedrückt. Auch die AKDB drohte mit den Indern und druckte damit den Preis auf nur 2 DM pro Anweisung. In der Schweiz hatte ich Sfr 4,- pro Anweisung verlangt. Ich musste deshalb auf meinen Anteil verzichten und Frau Erdös das ganze Einkommen überlassen um die Reise- und Unterhaltskosten zu bezahlen. Es blieb ihr selbst nur wenig übrig von den DM 90,000 die wir als Firma bekommen haben. Anderseits brauchte sie eine starke Motivation, um diese vertrackte Codemasse allein in so kurzer Zeit zu bewältigen. Wir hatten nur ein Jahr mit der Konversion fertig zu werden. Ich habe deshalb von meinem Verdienst in anderen Projekten was dazu beigesteuert. Es hat gewirkt. Das große Hauptprogramm wurde nicht nur konvertiert, sondern auch in sieben kleinere Programme zerlegt. Am Ende haben wir 15 Programme konvertiert und getestet.

Der Test der konvertierten Programme hat sich als besonders langwierig herausgestellt. Der Kunde gab uns seine Testdaten für verschiedene Kommunen von Großstädten bis zu Landkreis. Mit diesen Daten konnten wir bei weitem nicht alle Pfade durch den Code erreichen. Frau Erdös musste zusammen mit dem zuständigen Entwickler bei der AKDB die Daten anreichern, um in mühselige Kleinarbeit auch die seltenen Pfade zu durchlaufen. Hinzu kam die Aufdeckung jeder Menge alter Fehler, die schon immer in den Code gewesen sind. Diese mussten auch in den COBOL Programmen korrigiert werden. Die eigentliche Konvertierung konnte in den ersten 4 Monaten vollendet werden, aber der Test hat sich über 6 Monate hingezogen. Zum Schluss folgte ein Paralleltest von 2 Monaten.

Ich wollte mehr Geld für den zusätzlichen Testaufwand aber der Kunde blieb stur. Wir bekam nur, was wir ursprünglich vereinbart haben. für Frau Erdös reichte es. Für die SES blieb nichts übrig. Dies sollte mein letztes Assembler Konvertierungsprojekt sein. Es gab einfach keine Möglichkeit mit solchen Projekten zu verdienen. Dies hat mich bewogen die Firma SES nach 20 langen Jahren aufzugeben.

8.6 Anwendung der Softwarekapselungstechnik bei der Sparkasse Informatik

8.6.1 Kapselung einer IDMS Datenbank

Nach unserem Abzug aus der Schweiz ist Herr Jandrasics mit seinem Team aus der alten Szamalk Institut einen eigenen Weg gegangen. Über die Privatbank in Zürich, für die ich gearbeitet habe, gewann ich einen Auftrag für ihre Tochtergesellschaft in Hamburg, die ich an seiner Firma SoRing weitergeleitet habe. Auch dort ging es um die Ablösung eines vernetzten Datenbanksystems – IDMS – durch eine ORACLE relationale Datenbank. Die COBOL Programme mussten wegen der Datenzugriffslogik, die in COBOL festverdrahtet war, alle umgeschrieben werden. Herr Jandrasics's Leute waren mit dieser Problematik vertraut und konnten das Projekt in eigener Regie durchführen. Ich wurde nur einmal dazu gerufen, als es sich herausstellte dass die neuen Batchprogramme viermal so viel Zeit brauchten als die alten. Die Batchläufe liefen die ganze Nacht durch und waren in der Früh noch immer nicht fertig.

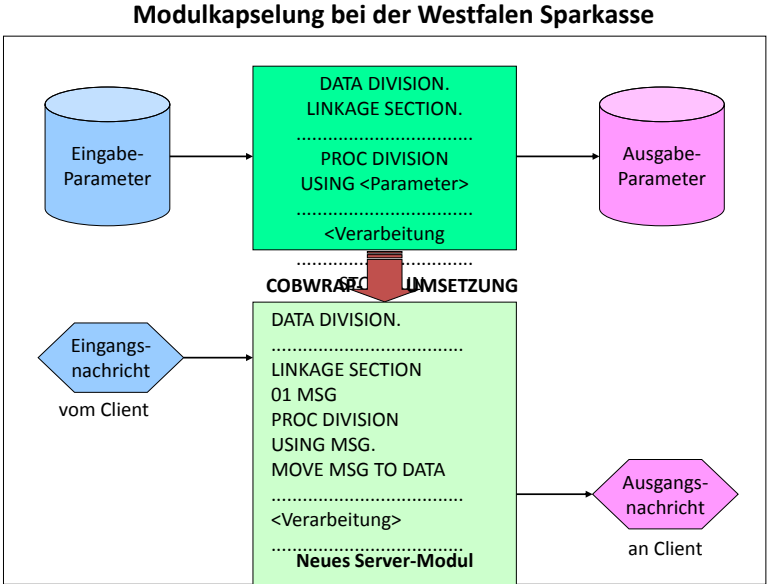
Der Kunde hatte einen ORACLE Experten aus Kalifornien geholt. Er hat dieses und jenes versucht ohne signifikante Verbesserung und fragte am Ende, warum die Deutschen überhaupt in dem antiquierten Batchmodus noch arbeiten. Sie sollten ihre Batchprogramme in Online Programme umsetzen. Dann würden sie zu Oracle passen. Für den Kunden war das ein schlechter Scherz. Er war überhaupt nicht bereit seinen Betrieb umzustellen um dem Datenbanksystem gerecht zu werden. Ich habe einen anderen Weg vorgeschlagen. Sie sollten die alten Datensätze als Tabellen mit nur zwei Spalten definieren – ein Schlüsselfeld und ein Blob-Feld. Das Blob-Feld würden wir genau so behandeln, wie bisher die alten IDMS-Sätze. Für die Online-Abfrageprogramme würden wir die Blobs über eine Zugriffsschicht in einzelne Felder wieder zerlegen. Das hat auch funktioniert, die Batchläufe waren danach nur 50% langsamer als bisher. Der Kunde gab sich damit zufrieden und das Projekt lief weiter zu einem erfolgreichen Abschluss.

Aufgrund dieser guten Referenz konnte die SoRing einen noch größeren Auftrag in Hamburg ans Land ziehen. Sie erhielt einen Auftrag, sämtliche Applikationen der

Hamburg-Mannheimer Versicherung von einer maßgeschneiderten 4GL Sprache in Standard-COBOL umsetzen. Das Projekt dauerte mehrere Jahre.

8.6.2 Erfolgreiche Kapselung eines COBOL Buchungssystems

Ich war inzwischen bei der Sparkassen Informatik in Bonn als Migrationsberater gelandet. Es gab dort viele Konzepte und Richtlinien für die diversen Sparkassenbetriebe zu schreiben. Viele der anderen Beratungshäuser haben nur Konzepte verfasst und es den Kunden überlassen sie auszuprobieren. Sie wussten es würde eventuell Jahre dauern bis der Kunde dahinter kam, dass das alles Luftschlösser waren. Ich bestand deshalb darauf, alles was wir zu Papier brachten auf Tauglichkeit zu prüfen und zwar in einem echten Projekt. Die Chance bekam ich für meine Konzepte über Softwarekapselung bei der Sparkasse Westfalen in Münster. Sie hatten ein riesen Buchungssystem auf dem Mainframe mit 94 einzelnen COBOL Module auf sechs Hierarchieebenen verteilt. Ihr Ziel war dieses Programm als Server für ihre neue verteilte Java Clients zu verwenden.



Damals war der Begriff „Wrapping“ in aller Munde. Mit der Übergang zur Objekttechnologie empfiehl IBM ihren Kunden ihre alte, prozedurale Applikationen zu kapseln bzw. zu wrappen nach dem Motto „Don’t discard it, wrap it“ [Diet88]. Die alten Systeme sollten wie Objekte behandelt werden. Sie bekamen eine Schnittstelle zuge-

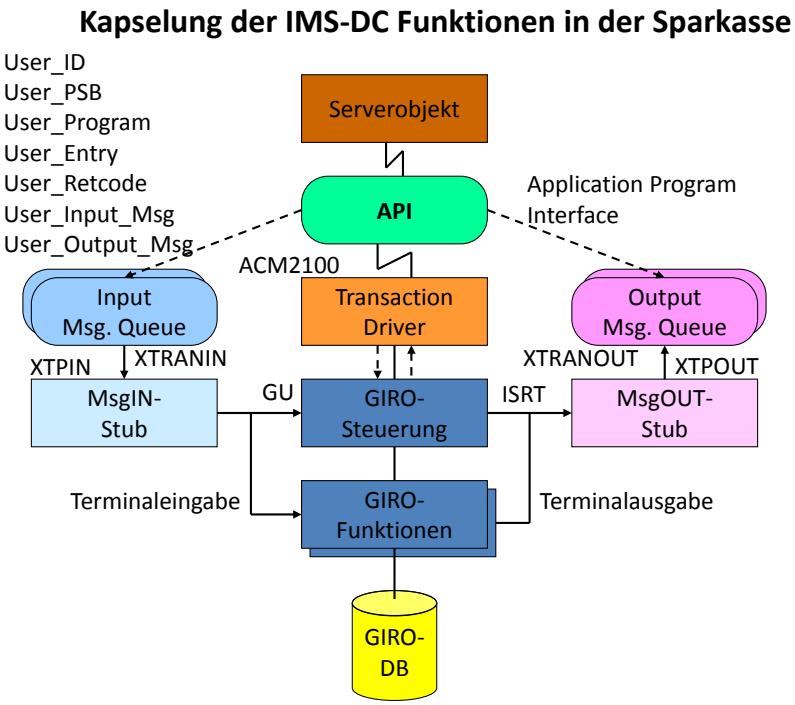
passt über die die anderen Objekte auf sie zugreifen konnten. Ich habe mein Konzept für die Kapselung von COBOL und PPL/I Programme bereits 1996 zu der internationalen Maintenance Conference in Monterey, Kalifornien vorgestellt, an dem gleichen Ort, wo in der gleichen Konferenz in der ich 1983 das Prüfstand Testsystem vorgestellt hatte [Sned96]. Ein Jahr später habe ich darüber einen Artikel in dem deutschen Objektspektrum veröffentlicht [Sned96a]. Dadurch ist die Sparkassen Informatik auf mich aufmerksam geworden. Nun boten sie mir die Gelegenheit, mein Konzept unter Beweis zu stellen.

Mit Hilfe eines jungen ungarischen Mitarbeiters habe ich ein Tool entwickelt, um aus dem COBOL Programm ein Wrapper Programm zu generieren, das einfließende CORBA Nachrichten empfängt und in COBOL Modulaufrufe umsetzt. Umgekehrt konnte es COBOL Ausgangsparameter in CORBA Nachrichten versetzen und diese an das Client-Programm zurücksenden. Für das Buchungssystem der Westfalen Sparkasse hat es einwandfrei funktioniert. Ihre Java-Client-Applikationen konnten Aufträge an ihr zentrales Buchungssystem senden und Ergebnisse als CORBA Nachrichten zurückbekommen. Das Wrapper Tool erlaubte auch dein Aufruf einzelner Unterprogramme in der großen Buchungshierarchie. Das große Buchungsobjekt hatte somit mehrere Eingänge mit unterschiedlichen Parametern. Dies war ein Vorgriff auf der kommenden Web Service Technologie. Auf der negativen Seite hat der Test der Buchungsservice sich lange hingeschleppt. Hier habe ich bitter erfahren müssen, wie groß der Testaufwand für gekapselte Services ist. Ich musste jeden möglichen Eingang mit allem möglichen repräsentativen Parameterwert testen. Es hat Monate gedauert. Im Vergleich hat die Kapselung selbst nur ein paar Wochen gedauert. Ich musste einsehen, dass die Schere zwischen Entwicklung und Test sich immer breiter öffnete. Dies gab mir die Motivation später das Webtest Tool zu entwickeln.

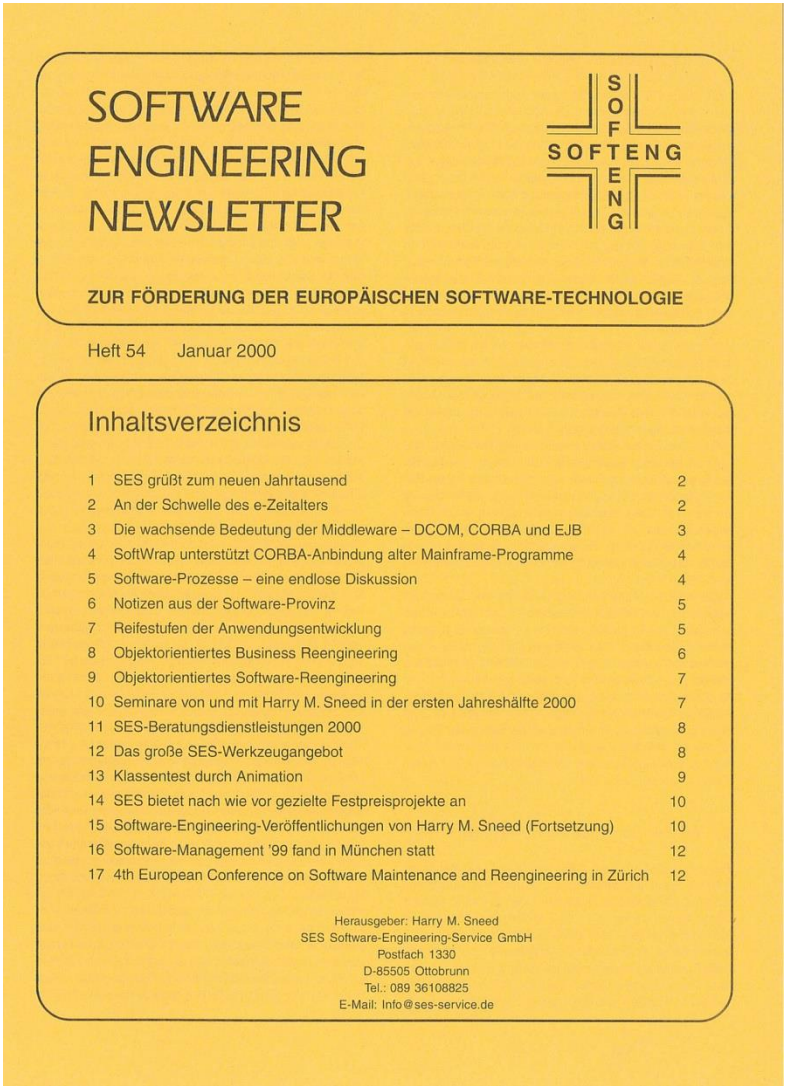
8.6.3 Gescheiterte Kapselung eines Assembler Buchungssystems

Beflügelt von dem Erfolg in Münster bin ich weiter zum nächsten Projekt in Hannover gezogen, zur dvA – das Rechenzentrum der norddeutschen Sparkassen. Der IT-Leiter dort war mein früherer Werkstudent aus den 80er Jahren. Mit seinen Veröffentlichungen über die Function-Point Schätzung hat er einen Namen gemacht. Leider waren die Bedingungen dort nicht so günstig. Die dvA Programme waren alle in Assembler und liefen unter IMS. Die dvA hatte als ich gekommen bin, schon zweimal versucht die Assembler Programme zu migrieren, einmal in C++ mit dem Wunderwerkzeug Fermat von der Universität Durham in England, dessen Entwickler Dr. Martin Ward ich von meiner Zeit als Dozent in Durham gut kannte [Ward99] und einmal mit den Indern von Case Consult [Borc97]. Beide Projekte sind gescheitert. Nun war die norddeutsche Sparkasse im Begriff neue Java-Beans Applikationen für

ihre Bankzweigstelle in den neuen Ländern zu entwickeln – die Hostprogramme sollten bleiben. In den Filialen selber sollten die Buchungen und Kreditanträge mit PC-Rechner erfasst werden. Anschließend sollten sie an Unix-Rechnern in der Kreiszentrale ergänzt und weitergeleitet werden an den großen Host-Rechner in Hannover. Dort sollten die Buchungen in der IMS-Datenbank durchgeführt und die Daten für die Kreditanträge bereitgestellt werden. Ich sollte die Schnittstelle zu den Host-Programmen schaffen.



Ich saß mutterseelenallein in dem neuen IT-Gebäude der dvA in Hannover und versuchte mit in die Assembler Logik einzuarbeiten. Es waren Assembler Programme von der übelsten Sorte mit vielen Overlays und physikalischen Adressierungen. Hinzu kam die IMS-Steuerungslogik mit der ich nur wenig vertraut war. In allen bisherigen Kapselungsprojekten hatte ich mit dem CICS-Monitor zu tun gehabt. Mit IMS hatte ich mich schon lange nicht mehr befasst. So kam ich nur langsam voran. es stellte sich auch heraus, dass die Assembler Source Code Wrapping gar nicht automatisierbar war. Ich musste in den Code manuell eingreifen und jeden IMS Entry Point mit dem SPF-Editor umändern.



SOFTWARE
ENGINEERING
NEWSLETTER

SOFTENG

ZUR FÖRDERUNG DER EUROPÄISCHEN SOFTWARE-TECHNOLOGIE

Heft 54 Januar 2000

Inhaltsverzeichnis

Herausgeber: Harry M. Sneed
SES Software-Engineering-Service GmbH
Postfach 1330
D-85505 Ottobrunn
Tel.: 089 36108825
E-Mail: Info@ses-service.de

Als die Zeit immer knapper wurde, habe ich von Gabor's Firma Rudi Majnar ausgeliehen. Er war neben Frau Erdös unser beste Assembler Experte. Mit Rudi's Hilfe ist es gelungen, vor dem Endtermin in Dezember 1998 vier Programme manuell hinter einer CORBA-IDL Schnittstelle zu kapseln. An einem Programm konnte ich sogar demonstrieren, dass es möglich war Nachtrichten von einem Unix-Vermittlungsrechner in ferner Jena zu erhalten, auf die IMS-Datenbank in Hannover zuzugreifen und eine Buchung durchzuführen. Das Proof of Concept wurde somit

erbracht, aber sonst nichts [SnMa98]. Die Aufgabe wäre es gewesen, alle 800 Assembler Programme automatisch zu kapseln und für die Client-Systeme in den Zweigstellen Schnittstellen zu generieren. Dies stellte sich – zumindest für mich – als unmöglich heraus. Was für COBOL und PL/I funktioniert hatte, war auf Assembler nicht übertragbar. Das hätte ich wissen müssen und das Projekt gar nicht annehmen dürfen. So habe ich mein Ruf bei der Sparkassen Informatik verdorben. Das dvA Wrapping Projekt war mein deutsches Waterloo. Mein Beratungsvertrag wurde nicht mehr erneuert und ich hatte in Deutschland keine Aufträge mehr. Meine Karriere in Deutschland war damit beendet. Ich war frei für Wien.

9 Der Ruf nach Wien

9.1 Rückkehr nach Wien

Ich hatte 1977 mein erstes Testprojekt für das Spardat in Wien bekommen, sogar noch vor dem Siemens Projekt in Budapest. Im Rahmen dieses Projektes habe ich einige Tage in Wien verbracht. Tagsüber habe ich den Test im Spardat Rechenzentrum im 3. Bezirk vorbereitet. Nachts habe ich draußen in dem Vorort Stockerau getestet. Damals fand ich die Stadt Wien recht interessant, sie hatte eine reiche Geschichte mit vielen imposanten alten Bauwerken, die Amerikaner immer bewundern. Als ich dann Wien verließ in Richtung Budapest, hätte ich nie geahnt dass ich hier eines Tages für längere Zeit zurückkehren würde. In den nächsten 20 Jahren habe ich die Stadt nur tageweise besucht um kurze Lehrgänge zu halten. Das sollte sich 1998 ändern.

Mein letzter Auftraggeber in Deutschland war, neben der deutschen Börse, die Westdeutsche Landesbank in Düsseldorf. Der Entwicklungsleiter dort, ein gewisser Herr B, hat mir einen Auftrag erteilt verschiedene Wertpapierverwaltungssysteme zu analysieren und zu vergleichen. Mit ihm bin ich eigentlich gut zurechtgekommen. Er schätzte meine Werkzeuge und meine Systemmesstechnik. Leider hatte er Pech mit der Entwicklung eines neuen Weltpapiersystems gehabt. Der Auftragnehmer, ein renommiertes internationales Softwarehaus hatte ihm viel versprochen und nichts gehalten. Wie immer wurde der Aufwand für die Entwicklung eines neuen Systems unterschätzt. Das Projekt musste nach mehreren Terminverschiebungen abgebrochen werden. Als Folge musste Herr B. die Bank verlassen.

Inzwischen war ich in Hannover beim Sparkassenrechenzentrum gewesen. Nachdem das Projekt dort für mich zu Ende ging, stand ich plötzlich ohne Auftrag auf der Straße. Obwohl überall auf die Jahrtausendwende hingearbeitet wurde – das war im Frühjahr 1998 das dominierende Thema - bekam ich kein weiteres Projekt. Ich habe bei mehreren Anwenderfirmen mit Codeanalysen versucht einen Auftrag zu bekommen, aber ich blieb ohne Erfolg. Das geplante Migrationsprojekt für die Bayerische Versorgungskasse wurde zu Gunsten einer Standardsoftware gestrichen. Lediglich das Assembler Konvertierungsprojekt von der bayerischen kommunalen Datenverarbeitung blieb mir übrig, aber dies war für mich eher ein Verlustprojekt. Deshalb habe ich es an meine ungarische Assembler Spezialistin – Frau Erdös - delegiert und suchte

für mich selbst was anders. Noch musste ich das SES Büro in Ottobrunn erhalten und für das Gehalt des Büroleiters und einer Hilfskraft aufkommen. Ich brauchte also einen größeren Auftrag um die Firma SES über Wasser zu halten. Zu diesem Zweck habe ich mich an einige der größeren deutschen Softwarehäuser gewandt aber sie haben abgewinkt. Ich war ihnen schon zu alt. Immerhin war ich schon 58.

Das war im Juni 1998. Ich musste auf mein eigenes Gehalt verzichten und zusehen, wie ich über die Runde kam. Ich war Drum und Dran, das Büro zu schließen und die Firma aufzugeben. Dann kam aus heiterem Himmel ein Anruf von meinem früherem Kunden, dem Herrn B aus Düsseldorf. Er war inzwischen Berater für ein Wertpapierprojekt der österreichischen Banken in Wien. Er meinte, sie könnten mich dort für die Qualitätssicherung ihrer Software gut gebrauchen. Das Geld für die Reise habe ich von meinem Sohn Stephan geliehen, der inzwischen seine Lehre als Systemelektroniker beendet hat und als Website Entwickler für eine schwedische Internetfirma arbeitete.

Das Büro der Wiener Firma, die Software Data Service, war in der Aspenbrückengasse im 2. Bezirk, nicht weit vom Donaukanal. Dort hat mich der Herr B. dem Geschäftsführer der Softwarefirma vorgestellt, ein jovialer Wiener mit dem Aussehen des heutigen Bürgermeisters Häupl. Herr P. war gerade dabei, eine große Entwicklungsmannschaft aufzubauen. Das Wertpapierverwaltungssystem, um das es ging hieß GEOS – Global Entity Order System. Es waren schon über 150 Mitarbeiter dabei, das neue System für die Banken zu bauen – Konzeptler, Entwickler und Tester. Herr B. hatte mich als Toolentwickler und Qualitätsberater empfohlen. Herr P. hat mir sofort ein Angebot gemacht, ich konnte sofort anfangen. Es war nicht gerade mein Wunsch in Wien zu arbeiten und erst recht nicht in einem C++ Laden, aber ich hatte kein anderes Alternativ. Die Stadt liegt zwischen München und Budapest und war insofern für mich mit dem Büro in Ottobrunn und den Mitarbeitern in Budapest eine praktische Lösung. So habe ich das Angebot sofort angenommen.

Von Anfang an hatte ich bei der SDS zwei Haupttätigkeiten – die eine als Qualitätsberater und das andere als Toolentwickler. Als Qualitätsberater sollte ich die Qualität der produzierten Dokumente und Codekomponente prüfen und darüber an die Geschäftsführung berichten. Dass die Geschäftsführung auch die Systemgröße und die Mitarbeiterproduktivität messen wollte stellte sich später heraus. Als Toolentwickler sollte ich Testwerkzeuge für die statische und dynamische Analyse beisteuern, insbesondere ein Testwerkzeug um einzelne Module los gelöst vom Gesamtsystem zu testen. Da die SDS kein Standardwerkzeug kaufen konnte lag daran, dass sie keine Standardprogrammiersprache verwendete. Sie benutzte zwar C und C++, hatte

aber eine eigene Makrosprache daraus gemacht. Alle Versuche mit den damals verfügbaren Testwerkzeugen – Cantata, Testbed und CTest - sind an der firmeneigener Makrosprache gescheitert. Wer einmal von dem Standard abkommt, den bleibt nichts anders übrig als alles selber zu machen, einschließlich der eigenen Werkzeuge [KaBa03]. Die SDS hatte zwar eine eigene Werkzeuggruppe aber sie war mit der Pflege und Weiterentwicklung ihrer Makroprogrammiersprache voll ausgelastet. Sie hatten dazu einen Precompiler entwickelt und er war noch nicht ausgereift. Außerdem musste sie auch das Werkzeug für die betriebseigene Spezifikationssprache CMF – Concept Management Facility – weiterentwickeln. Für Testwerkzeuge hatten sie zu dem Zeitpunkt werde die Kapazität noch das Knowhow. Also übernahm ich diese Aufgabe. Ich musste froh sein, in meinem Alter überhaupt eine Aufgabe zu bekommen. Ich fand ein Hotel in der Praterstrasse und begann jede Woche mit der Bahn nach Wien zu pendeln.

9.2 Als Test-Tool Entwickler im GEOS Projekt

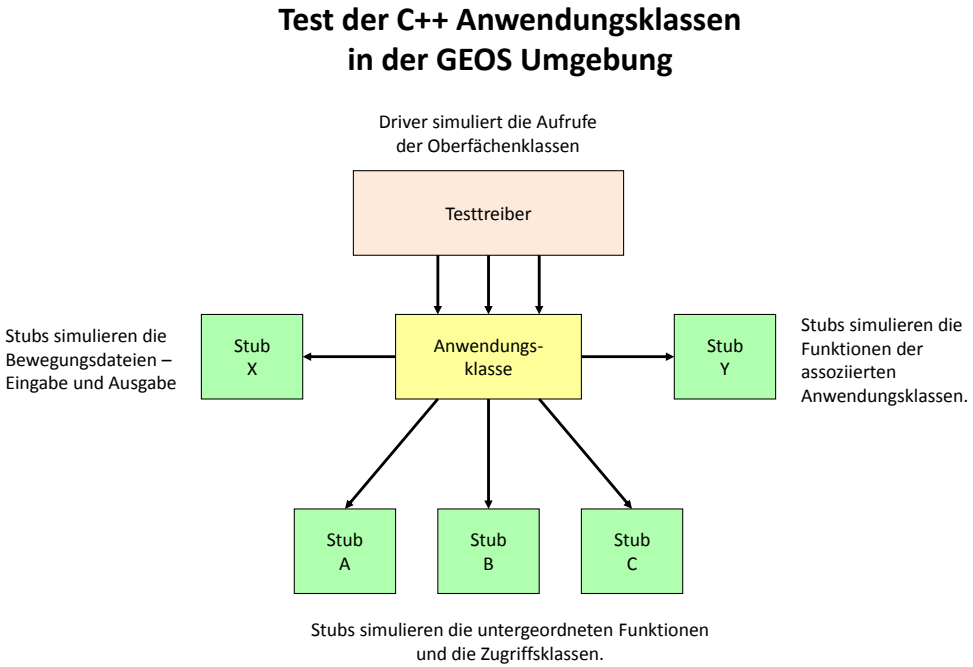
9.2.1 CPPTest

Leider hatte ich kein Glück mit dem ersten Tool, dem Modultestrahmen. Mein Plan war einen Testtreiber und die Test Stubs aus dem Modul-Source-Code zu generieren. Dafür war es notwendig, einen Parser für den C-Code zu entwickeln. Mit dem reinen C-Code hatte ich keine besonderen Probleme, dafür gab es genügend Vorgängerprodukte und ein Haufen Fachliteratur. Probleme machten die lokale Spracherweiterungen, die von der Standard C-Syntax abwichen. Da gab es eigene IF Anweisungen, die mit einem Semikolon endeten, eigene Funktionsaufrufe und eigene Datentypen. Beim Parsen musste man unterscheiden, ob es sich um eine Standard C-Anweisung oder um eine Makro C-Anweisung handelt. Am schwierigsten war es die Datentypvereinbarungen zu erkennen. Manchmal verwendeten die Programmierer Standard C und manchmal Makro C. Es hat eine Weile gedauert, aber am Ende ist es mir doch gelungen, den Makro-C Code zu parsen und einen Syntaxbaum mit Anweisungstabelle aufzubauen.

Für jede öffentliche Funktion, d.h. jede Funktion, die von außen aufrufbar war, habe ich eine Aufrufsfunktion in dem Testtreiber eingefügt. Die zu übergebenden Parameter konnte der Tester direkt zuweisen oder er konnte sie mit Werten aus einer Testdatentabelle belegen. Dadurch war es möglich, beliebige Eingänge im Testobjekt anzuspringen und von dort aus einen Pfad durch das Objekt zu verfolgen. Ich habe den Source-Code instrumentiert, d.h. Durchlaufzähler in jeden Ablaufzweig eingebaut, um die Pfade zu registrieren. Die Durchlaufzähler hielten in einer Trace-Tabelle

fest, welche Zweige wie oft durchlaufen wurden. Dadurch konnte der Tester den Verlauf des Testes rekonstruieren.

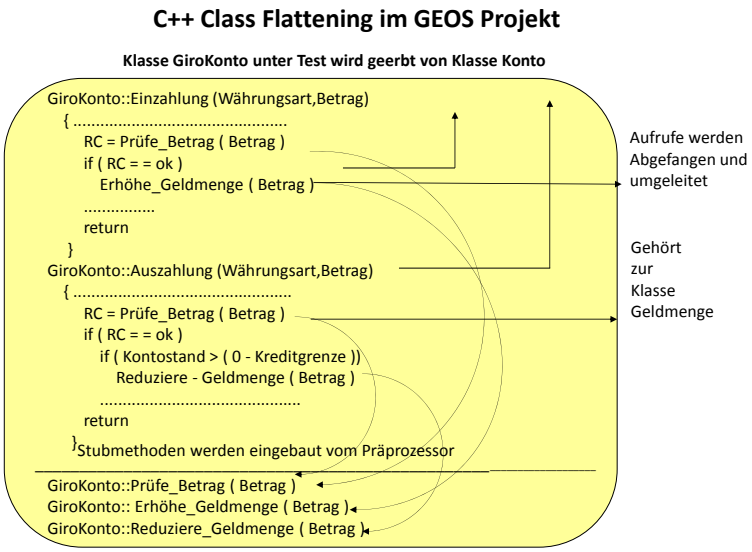
Schwierigkeiten traten auf, wo der Modul unter Test andere Module aufrief. Diese aufgerufenen Module durften in den Modultest nicht einbezogen werden. da sie ihrerseits weitere Module aufgerufen haben. Hätte ich damit begonnen die aufgerufenen Module in den Test einzubeziehen hätte ich damit geendet, die ganze Modulbibliothek zu testen. Außerdem waren zu dem Zeitpunkt des Zielmodultests viele der benutzten Untermodule noch gar nicht vorhanden. Beim Top-Down Ansatz beginnt der Entwickler mit den übergeordneten Modulen und arbeitet sich von oben nach unten durch die Modulhierarchie hindurch. Die untergeordneten Modulen mussten simuliert werden.



Ich habe für jeden Aufruf eines untergeordneten Moduls einen Platzhaltermodul erzeugt – einen sogenannten Stub – aber diese Platzhaltermodul musste sinnvolle Datenergebnisse zurückgeben, d.h. der Entwickler musste für jeden simulierten Untermodul eine Reihe stellvertretender Rückgabewerte bereitstellen. Diese Datenwerte

mussten nicht nur zum vereinbarten Datentyp passen, sie mussten auch zu ihrer Verwendung im Testobjekt passen, d.h. sie mussten alle Bedingungen erfüllen. Es war nicht möglich einfach Zufallswerte zu erzeugen.

In meinem ersten Modultestwerkzeug, dem Prüfstand, erlaubte ich es dem Benutzer die simulierten Daten dynamisch im Dialogbetrieb zu setzen. Er bekam während der Ausführung des Moduls die Namen und Typen der Parameter und konnte geeignete Werte zuweisen. Hier durfte nur im Batch-Modus getestet werden, d.h. die Testdaten mussten im Voraus definiert sein. Auch das hat funktioniert, war aber sehr zeitaufwendig für jede aufgerufene Funktion die möglichen Rückgabewerte zu spezifizieren. das Tool hat zwar korrekt funktioniert, war aber von den Entwicklern nicht angenommen. Als Folge der Objektorientierung war der Code in viele tausend Einzelbausteine aufgeteilt. Manche Module im GEOS System haben bis zu 200 Untermodule aufgerufen. Der Aufwand, alle Untermodule zu simulieren war einfach zu hoch. Man hatte zwar die großen monolithischen Codebausteine vermieden, dafür jedoch viele Abhängigkeiten zwischen den kleinen Bausteinen geschaffen. Diese Abhängigkeiten zwischen den Moduln spiegelten sich in den Abhängigkeiten zwischen den Entwicklern wider. Der Entwickler des Hauptmoduls war von jedem anderen Entwickler abhängig, dessen Untermodul er verwendete. Wollte er etwas an der Schnittstelle ändern – was am Anfang häufig der Fall war – musste er dies mit den anderen betroffenen Entwicklern absprechen. Wie so oft in der IT-Welt, führt die Lösung eines Problems zur Schaffung weiterer Probleme. Hier war dies die Abhängigkeit der Bausteine untereinander. Man konnte zwar das Verhalten jener Module simulieren, von denen das zu testende Modul abhängig war, aber es war nicht gesagt, dass die simulierten Module sich wirklich so verhalten würden, wenn sie implementiert sind. Vielleicht hat der andere Entwickler ganz andere Vorstellungen davon, was er zurückgeben sollte. Der Übergang von der prozeduralen zur objekt-orientierten Programmierung hat die alten Probleme nur gegen neue Probleme ausgetäuscht.



Dies war nicht nur ein Problem für mich mit meinem Modultestwerkzeug. Es war auch ein Problem für das Projekt im Allgemeinen. Es verzettelte sich in gegenseitigen Abhängigkeiten so sehr, dass das Projekt begann von der reinen Objektorientierung Abstand zu nehmen und Codebausteine zu duplizieren. Das führte wiederum dazu, dass es von einem und demselben Modul mehrere Kopien gab, für jedes Teilsystem ein anderes Duplikat. Dadurch hatte der Teilsystemverantwortlicher alles in seinem Bereich unter seiner Kontrolle. Er war auf keine fremde Softwarebausteine angewiesen. In der Software Entwicklung werden die Probleme letztendlich hin und hergeschoben, ohne sie wirklich zu lösen.

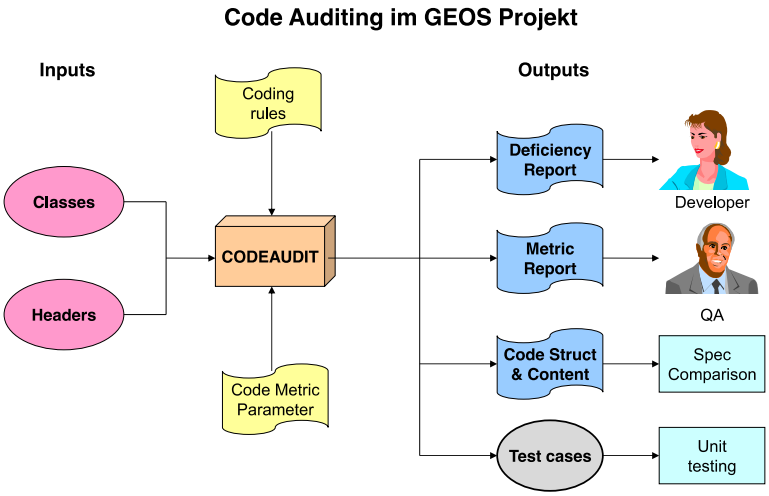
Ich bin jedenfalls mit meinem Modultester an dem Problem der Abhängigkeiten gescheitert. Nach einem halben Jahr war mein Tool zwar lauffähig aber nur wenige Entwickler waren bereit es zu nutzen. Es war zu aufwendig die Testumgebung aufzubauen. Der Versuch, einen Modultest im Projekt einzuführen wurde aufgegeben. Jetzt hieß es, Module zu kompilieren und dann über den Zaun zu der Testabteilung zu werfen, die sowieso alles testen sollte. Man entschied sich also für den Big-Bang Ansatz.

9.2.2 CPPAnal

Meine Arbeit an den C++ Testtreiber war nicht ganz umsonst. Durch die Parsing des Codes um einen Testrahmen zu generieren, musste in der Lage, den Code zu analysieren. Ich konnte einzelne Mängel und Schwachstellen im Code erkennen. Zu der Zeit

gab es bereits eine Reihe Bücher über Programmierregel, wie das von Tim Budd über Java [Budd98] und von Steve Qualline über C++ [Qual99]. Tim Budd hatte damals für mich in dem Budapester Testlabor als Praktikant gearbeitet. Inzwischen war er Vollprofessor an der Universität Arizona State. Außerdem gab es bereits in der SDS eine Programmierkonvention. Einer aus der Sprachentwicklungsgruppe war ein Jahr vor meinem Eintritt in die SDS bei der Microsoft in den USA gewesen. Die Firma hatte ihn zusammen mit zwei anderen dorthin geschickt, um von Microsoft zu lernen, was sie mit ihrer C/C++ Entwicklung vorhatten. Dabei ist er auf die Microsoft Codierrichtlinie für C und C++ gestoßen. Dazu gehörte die sogenannte Hungarian Notation für die Benennung der Daten und Funktionen, genannt nach dem C++ Chefprogrammierer Karoly Simoni, der sie erfunden hat [Simo97]. Er hat sie zurückgebracht und publiziert, aber kaum jemand in der Entwicklung hatte davon Notiz genommen. Er hatte keine Möglichkeit gehabt, die Regeln durchzusetzen. Er konnte sie nur als Empfehlung anbieten.

Ich habe die circa 40 bestehende Regel übernommen und durch andere aus der C/C++ Literatur ergänzt, sodass ich am Ende auf mehr als 60 Regel für die Gestaltung des Codes kam. Bis Ende 1998 hatte ich für die SDS eine mehrseitige Programmierrichtlinie verfasst und den Entwicklern bereitgestellt [Sned99]. Ich habe sie auch gewarnt, ich wurde ein Werkzeug bauen um ihren Code zu kontrollieren. Den Entwurf des Werkzeugs habe ich gleich beigelegt. Somit habe ich ein neues Projekt für die Qualitätssicherung ins Rollen gebracht. Wenn wir die Codemodule nicht testen könnten, sollten wir sie zumindest prüfen. Sowohl der Geschäftsführer als auch der technische Leiter der SDS hat das Projekt gebilligt und haben mir ihre Unterstützung zugesagt. Ich habe vorgeschlagen, den Code nicht nur zu prüfen sondern auch zu messen. Mit dem Tool SoftDoc hatte ich schon zu Beginn der 80er Jahren COBOL und PL/I Code geprüft und gemessen. Im Laufe der Jahre sind andere Sprachen wie Assembler und Natural dazu gekommen. Schon damals hatte ich begonnen die Größe und Qualität des Codes gemessen. Mit dem PC Nachfolgeprodukt von SoftDoc, dem Tool SoftAnal habe ich in den 90er Jahren mehrere größere Source-Analysen durchgeführt u.a. für die deutsche Börse, die Landesbanken und die niederländische Telekom. Die gleiche Funktionalität wollte ich auch der SDS bieten. Ich habe nicht nur das Tool angeboten, sondern auch die Dienstleistung. Ich würde alle drei Monate, bzw. bei jedem neuen Release den gesamten GEOS Code und den neuen Core-banking Code analysieren, prüfen und messen.



CODEAUDIT Program checking & measurement

Der Geschäftsführer der SDS, Herr P. war an der Größenmessung besonders interessiert. Zu jener Zeit war die Function-Point Zählung besonders populär. Alle wollten Function-Points zählen, auch Herr P. Er brauchte die Function-Point Zahl um seinen Geldgeber den Fortschritt des Projektes zu belegen, denn die Projektkosten sind ständig gestiegen und es gab bis dahin kaum Anwender. Er wollte die Kosten mit den Function-Points rechtfertigen. Er meinte, die Function-Points sei das Dow-Jones Index der Software Industrie. Ich war schon immer skeptisch was das Zählen von Function-Points anbetraf, ich kannte mich damit zu gut aus, aber ich habe ihm zugesichert, ich würde auch die Function-Points zählen. Das war mein Zugeständnis, um das Projekt zu bekommen. Da musste ich an die Frage von Dr. Rothhardt denken als er aus der ehemaligen DDR kam – „wie ehrlich kann man in einer kapitalistischen Marktwirtschaft sein?“.

Die Arbeit am CPPAnal – so hieß das neue Analyse Werkzeug – war viel schwieriger als ich mir zunächst vorgestellt hatte. C ist keine einfache Sprache und C++ erst recht nicht. Hinzu kam die C-Precompiler Sprache – die #ifdef Anweisungen, die wahlweise mitcompiliert werden und zu guter Letzt die Makrosprache der SDS, alles in einem einzigen Source-Text. Die erste Herausforderung bestand darin, die verschiedenen Anweisungsarten auseinander zu dividieren. Wurde eine Anweisung identifiziert, musste ich sie in Syntaxelemente zerlegen und die Einzelelemente zählen -

z.B. die Operatoren und die Operanden, die Variabel und die Kostanten. Mit den früheren Sprachen – wie COBOL und PL/I gab es die vielen Schlüsselwörter an denen man die Semantik der Anweisungen erkennt. In C und C++ gibt es nur wenigen Schlüsselwörtern. Die Semantik hängt von der Position der Sprachelemente ab.

Ich musste mit jeder Anweisung zweierlei Aufgaben bewältigen – messen und prüfen. Zum einen musste ich die Syntax Elemente zählen, z.B. für die Halstead Metrik muss man Operanden und Operatoren zählen, für die McCabe Metrik muss man Knoten und Kanten zählen, und für die Henry Metrik muss man Eingänge und Ausgänge zählen. Das Schlimmste war es, die Function-Points zu zählen. Function-Points ergeben sich aus den Eingaben und Ausgaben einer Komponente. In den früheren Sprachen wie COBOL, PL/I und NATURAL gab es Schlüsselwörter für die Anweisungsarten. Für Ein- und Ausgabeoperationen gab es Schlüsselwörter wie READ, WRITE, DELETE und FETCH. Da wusste der Parser, es handelt sich um eine Ein- oder Ausgabe Operation. Auch die SQL Datenbankoperationen sind leicht erkennbar. Nichts dergleichen in C++. Es gibt zwar eingebaute IO-Operationen wie get und put aber sie werden meistens nur zu Testzwecken benutzt. Mit C und C++ sollte die Verquickung mit dem Betriebssystem vermieden werden. Das oberste Ziel ist die Unabhängigkeit von der technischen Umgebung. Deshalb sollten die Interaktionen mit dem Betriebssystem und den Datenbanksystem ausgelagert werden. Dafür sollten die Entwickler für jede Umgebung eigene IO-Funktionen bereitstellen, die sie von dem Mainline-Code ausrufen. In der Regel sind diese Funktionen in einer speziellen Bibliothek, wo sie von der jeweiligen Systemgruppe betreut werden. So was das auch im GEOS Projekt. Ich hatte keinen Zugang zu jener Bibliothek. Die Interaktionen mit der Umgebung, d.h. die Eingabe-und Ausgabe-Operationen konnte ich nur an den Namen der Funktionen erkenne. So kam ich auf die Idee, eine Funktionstabelle mit den Namen aller ausgelagerten Funktionen aufzubauen. In dieser Tabelle musste der Toolbenutzer neben dem Funktionsnamen auch den Funktionszweck eintragen. Dies war die einzige Möglichkeit für mich die Eingaben und Ausgaben zu erkennen und somit auch Function-Points zu zählen. Eine solche Tabelle mit den C++ Funktionsmakros habe ich für die GEOS Systeme aufgebaut. Bis zum heutigen Tag, wenn ich zu einem Kunde gehe um objektorientierte Software zu messen, beginne ich damit, eine Funktionstabelle der speziellen IO und DB Funktionen aufzubauen und zwar nur um die Function-Points zählen zu können.

Kalkulation von Weiterentwicklungskosten nach Function-Points

	KMS			PMS		GEA	KOS		ZAS				FIS	Zentrale Systeme				Total
	KMS	PVBE	MMS	PMS	PMSI	GEA	KOS	ITM	ZAS	ARPL	AUF	TRM	FIS	KOND	ARS	LAW	STS	
Batch Proc.	0	0	0	0	0	0	546	0	117	39	0	117	195	0	0	0	78	1092
Dialoge Proc.	2418	182	0	78	572	1300	1274	78	234	494	104	104	494	1066	286	546	1300	10530
Service Proc.	1354	414	0	1587	161	2553	4554	138	460	1058	184	391	2047	552	1265	92	739	17549
Data Objects	570	150	0	120	0	250	790	10	30	100	50	120	550	110	500	0	50	3400
Reports	0	117	0	0	0	39	0	0	39	0	0	0	195	0	0	0	0	390
Anz. Tabellen	915	120	0	300	0	390	465	30	0	195	225	90	195	630	360	300	1350	5565
Total	5260	983	0	2085	733	4532	7629	256	880	1886	563	822	3676	2358	2411	938	3514	38526
Person Months	263	49,2	0	104,3	36,7	226,6	381,5	12,8	44,0	94,3	28,2	41,1	183,8	117,9	120,6	46,9	175,7	1926,3

Subsystem	FPs	PM`s	PY`s
KMS	6243	312	26
PMS	2818	141	12
GEA	4532	227	19
KOS	7885	394	33
ZAS	4151	208	17
FIS	3676	184	15
Zentrale Systeme	9221	461	38
Gesamt	38526	1926	161

10% Growth = 250 MM
20% Growth = 500 MM
30% Growth = 750 MM

Type	Berechnung	Faktor
Batch Process	(3 Inputs x 6)+(3 Outputs x 7)	39
Dialog Process	(2 Inputs x 6)+(2 Outputs x 7)	26
Service Process	(2 Inputs x 4)+(3 Outputs x 5)	23
Data Objects	(x 10)	10
Tables	(x 15)	15
Reports (Müller)	wie Batch	39

Person month (PM) = FP/20
Person Year (PY) = PMs/12

Zum anderen musste ich die Anweisungen prüfen – ob sie erlaubt sind oder nicht und ob sie richtig formuliert sind oder nicht. So habe ich die verwendeten Datennamen geprüft, ob sie der Namenskonvention entsprechen. Gleichzeitig habe ich die Formatierung der Anweisungen, die Reihenfolge der Anweisungen und die Einrückung der Anweisungen kontrolliert. Wenn irgendeine Regel verletzt wurde, habe ich den betroffenen Anweisungen in einen Mängelbericht zusammen mit der verletzten Regel ausgedruckt. Diese Regelprüfung erwies sich als besonders ärgerlich für manche Entwickler, die mit den Regeln nicht einverstanden waren. Dies sollte später Anlass für meine Entlassungen sein, aber die Regel habe ich nicht erfunden, diese habe ich bei der SDS vorgefunden. Ich habe mir nur die Mühe gemacht, sie zu kontrollieren.

Am Ende kamen zwei Ergebnisse aus der Codeanalyse hervor – ein Messbericht und ein Prüfbericht. In dem Messbericht waren die Quantitäts-, Komplexität- und Qualitätsmetriken zusammengefasst. Dazu gehörten die Größenmasse Lines, Statements, Data-Points, Object-Points und Function-Points. In dem Prüfbericht wurden

pro Codemodul alle Regelverletzungen ausgelistet mit einem Verweis auf die Stelle, wo sie aufgetreten sind. Es wurde dabei zwischen schweren, mittelschweren und leichten Mängeln unterschieden. Leider habe ich mit diesem Bericht keine Freunde gewonnen. Software Qualitätssicherung ist eine undankbare Aufgabe, weil jeder Entwickler etwas Anderes unter Qualität versteht. Viele der sogenannten Mängel wollten sie nicht akzeptieren weil sie mit den Regeln nicht einverstanden waren. Diese wurden mit ihnen nie abgesprochen [Sned99a].

Die Messung der Software über den Source Code diente dem Management. Das Management suchte Zahlen über die Größe und Komplexität der Systeme um ihre Ausgaben entweder zu rechtfertigen oder zu planen. Die Prüfung der Software, bzw. die Aufdeckung der „Smells“ sollte der Qualitätssicherung dienen. Da es aber keine Qualitätssicherung in dem Sinne gab, diente es nur dazu die Entwickler zu verärgern. Es gibt natürlich Prüfungen über die man streiten könnte wie über die maximale Größe einer Methode. Es gäbe aber auch Prüfungen wie die Vermeidung eingebauter Texte – sogenannte „hard-wired data“ oder die Herauslösung zu tief verschachtelten Codeblöcke – die durchaus sinnvoll sind. Es ist jedoch sehr schwer die Entwickler dafür zu gewinnen. Entwickler mögen per se keinerlei Regelung. Meine Beharrung auf diese Codeprüfungen sollte mir am Ende zum Verhängnis werden. Bei den Prüfungen, wie bei manch anderer Sache, bin ich einen Schritt zu weit gegangen.

Die ersten Source-Analysen habe ich bereits Anfang 1999 durchgeführt. GEOS hatte zu diesem Zeitpunkt schon 6279 Source Moduln mit 2,3 Millionen Zeilen, 716.147 Anweisungen und 28.640 Function-Points Es dauerte etwa 2 Tage alle Sourcen zu verarbeiten. Da ich die Messzahlen in Tabellen akkumuliert habe, die in Intervalle gesichert wurden, konnte ich die den Analyselauf jederzeit abbrechen und am nächsten Tag wieder fortsetzen. Auf dieser Weise war ich in der Lage, alles allein in einer vertretbaren Zeit durchzuführen. Hauptsache war, Herr P. war mit der Function-Point Zählung zufrieden. Das Analyseprojekt sollte fortgesetzt werden [Sned00].

9.2.3 JavAnal

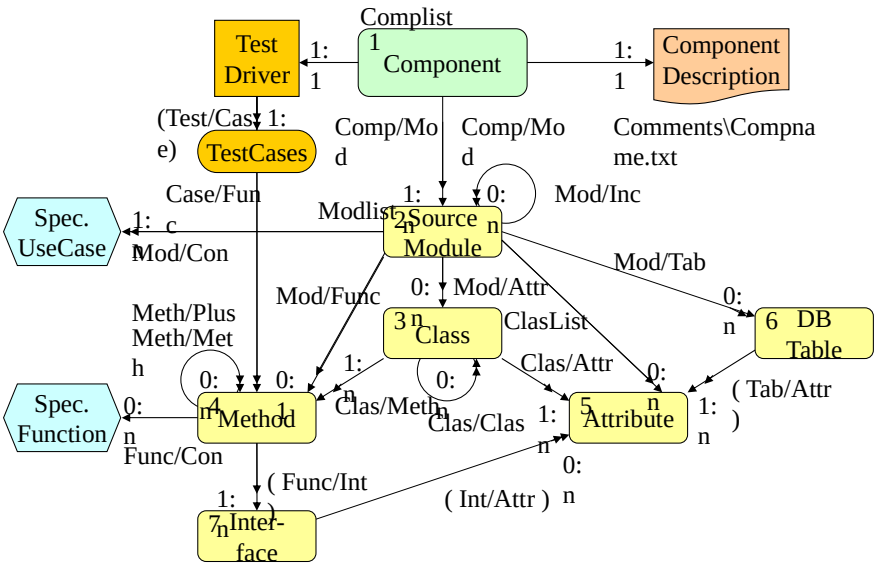
Im Jahr 2002 bekam ich die ersten Java Sourcen zu analysieren. Sie gehörten zu dem neuen GEOS Frontend, das das alte Maskenorientierte Benutzeroberfläche ablösen sollte. Da ich wie sonst immer nur begrenzte Zeit hatte, begann ich mit meinen C++ Parser und habe ihn schrittweise angepasst. Bald entdeckte ich die Unterschiede zwischen C++ als Hybridsprache und Java als eine reine objektorientierte Sprache. Bis ich fertig war musste ich den Parser ganz überarbeiten aber auf dieser Weise lernte ich Java kennen. Dank der Hilfe eines jungen Kollegen war ich bald in der Lage auch die Java Komponente zu messen. Es dauerte noch länger bis ich so weit war die

Codemängel zu erkennen. Erst muss man überhaupt wissen wonach man zu suchen hat. Manche sogenannte „Smells“ wie z.B. eingebaute Texte oder Zahlen sind die Gleichen wie in C++, andere wie z.B. die Nutzung des „final“ Attributs für Klassen ist Java spezifisch. Hier ist mir zum ersten Mal wirklich die Kluft zwischen den Entwicklergenerationen bewusst geworden. Die Java Generation nutzte ganz andere Begriffe. Statt von Mängel sprachen sie von „Smells“ und statt von „Reengineering“ sprachen sie von „Refactoring“. Oft redete man aneinander vorbei. Die neuen Gurus der Objekttechnologie, Leute wie Martin Fowler und Bob Martin, haben die Sprache der jungen Generation geprägt. Sollte ein Entwickler der alten Generation es überhaupt schaffen in die neue Generation aufgenommen zu werden, müsste er es erst schaffen ihre Sprache zu bemächtigen [Fowl99].

9.2.4 SoftRepo

Bei der SDS in Wien wurde ich wie früher bei der UBS in Zürich öfters gefragt nach Zusammenhänge und Abhängigkeiten im System. Ab einer bestimmten Größe ist ein Softwaresystem für die Analytiker und Entwickler selbst nicht mehr überschaubar. Sie schwimmen eben in einem großen Meer und haben keine Ahnung wo sie sich befinden. Typisch Fragen waren die Fragen woher eine bestimmte allgemeingültige Funktion aufgerufen und wo eine globale Variabel verwendet wird. Man sollte die bereits vorhandenen Funktionen wiederverwenden, aber keiner wusste wo sie waren. Es gab mehrere Tausende davon. Vor allem wusste man nicht wie sie sich zueinander verhalten. Alle hatten Angst etwas zu ändern denn keiner konnte übersehen welche Konsequenzen das haben würde.

In Zusammenhang mit einer Verlängerung des Transaktionsschlüssels kam die Frage auf wo überall im Code dieser vierstellige Schlüssel vorkam. Den Transaktionsschlüsselfeldern wurde in etlichen Modulen einen festen Text zugewiesen um die aktuellen Banktransaktion zu identifizieren. In anderen Modulen wurde der aktuelle Schlüssel wieder abgefragt. Um eine neue Bankkunde in Deutschland zu gewinnen müsste dieser Schlüssel von vier auf acht Stellen erweitert werden. Das hatte zu Folge, dass sämtliche Textzuweisungen verändert werden müssten. Ich bekam den Auftrag alle dieser Stellen zu identifizieren. Ich habe es mit einer Textsuche gemacht, aber es hat zwei Wochen gedauert. Daraufhin habe ich mich entschlossen eine Repository mit einer relationalen Datenbank zu bauen. Damit könnten Entwickler gesuchte Daten oder Funktionen innerhalb Minuten lokalisieren. Der Leiter der Entwicklung sah den Nutzen ein und war damit einverstanden. So kam ich dazu ein Repository für objektorientierte Systeme zu bauen sowie SoftOrg früher für prozedurale Systeme war.



Solange die Fragen der Entwickler sich nur auf einzelne Module beschränken genügt eine Textsuche. In dem Moment wo die Frage sich auf Teilsysteme oder das ganze Produkt bezieht ist eine einfache Textsuche nicht mehr ausreichend. Der Source Code ist auf viele verschiedene Bibliotheken verteilt und zum Teil in verschiedenen Sprachen. Eine ordentliche Impactanalyse geht nur über eine Datenbank durch die man navigieren kann von einem Artefakt zum anderen. Wichtig ist, dass man die Artefakte über semantischen Ebenen hinweg verbinden kann, z.B. ein Testfall mit einem Codemodul und der Codemodul mit einer Anforderung. Ich habe vier komplexe logische Schemen entworfen

- Ein Schema für die Anforderungen mit den CMF-Entitäten
- Ein Schema für die Architektur mit allen Entwurfsentitäten
- Ein Schema für den Code mit allen C++ und Java Elementen
- Ein Schema für den Test mit allen Testdokumenten, Testprozeduren und Testfällen. (siehe Repository Abbildung)

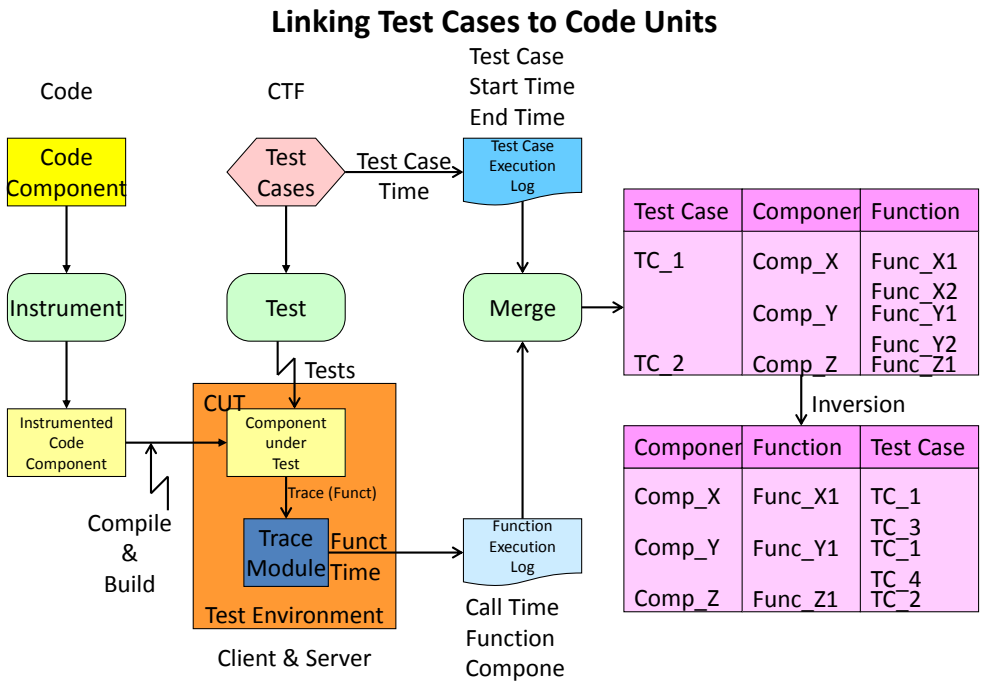
In München hatte ich einen Werkstudenten angestellt der mir geholfen hat die Tooloberflächen zu programmieren. Er war ein hervorragende C++ Entwickler und wir kamen gut miteinander aus. Er lernte Softwareengineering von mir und ich lernte C++ von ihm. Zusammen haben wir das Repository aufgebaut auf Basis einer MS-Access Datenbank. Zu zwei und später mit der Frau Erdös haben wir alles gebaut was zu einem vollen Repository-System gehört – der Loader, die Abfrage Utility, der Berichtsgenerator, der Impactanalysator und der Release-Planer. Später war es sogar möglich den Aufwand für Änderungen aufgrund der Impactanalyse zu schätzen [Sned00].

Es hat bis zu drei Tage gedauert die Repository mit den 5800 Anforderungstexten, den 1600 Entwurfselementen, den 4000 Sourcetexten, den 6500 Datenbankelementen und den 89.000 Testfällen zu bevölkern, aber dies geschah nur einmal vierteljährlich wenn ein neues Release freigegeben wurde. Mit Hilfe dieser Repository konnte ich helfen manche tiefgreifende Änderungen und Ergänzungen zum vorhandenen Produkt zu planen und zu schätzen. Leider ist es mir nie gelungen, die Projektleiter mit dem Tool selbst zu arbeiten. Wir haben alles getan um das Tool bedienungsfreundlich zu gestalten, aber wo das Wissen seitens der Anwender fehlt ist nichts zu machen. Nur wenige Projektleiter waren bereit sich mit der Sache tiefer zu befassen. Das Interesse hat gefehlt.

Nichtdestotrotz war die GEOS-Repository eins meiner größten Software-Engineering Leistungen. Es ist erst richtig fertig geworden kurz vor meiner Entlassung aus der Firma. Ich habe sie mitgenommen und später in zahlreichen Reverse Engineering Projekten eingesetzt. Ich habe sie auch in meinem Unterricht an der Uni benutzt. Die Studenten sollen lernen wozu ein Repository gut ist.

9.3 Impaktanalyse der Wartungsaufträge

Eine häufig gestellte Frage war nach den Kosten einer Änderung. Ich wurde immer wieder aufgefordert zu schätzen was die Durchführung einer Änderung kosten würde. In diesem Zusammenhang wurde ich auch befragt, welche Code-Komponente wieder getestet werden müssen wenn eine bestimmte Komponente geändert wird. Beide Fragen führen auf eine Impact-Analysis zu, ein häufig behandeltes Thema in der Software-Wartungsliteratur.



Zuordnung der Testfälle durch dynamische Analyse

Mein erster Versuch war über die statische Analyse. Angefangen mit der geänderten Klasse bzw. C-Prozedur, suchte das Werkzeug alle Klassen bzw. Prozeduren, die von dieser Klasse bzw. Prozedur abhängig waren, d.h. die übergeordneten, untergeordneten und nebengeordneten. Da dieses System einen hohen Verflechtungsgrad hatte, waren es sehr viele aber sie wurden erkannt. Was noch fehlte war die Verbindung der betroffenen Codebausteine zu den Testfällen. Ich sollte auch herausbekommen, welche Testfälle diese Bausteine testen.

Die Antwort auf diese Frage war keineswegs leicht. Wie ich später bei einem Besuch von Nokia in Helsinki erfuhr, ist dies ein universales Problem. Auch die Tester dort haben mit dieser Frage gerungen. Sie hatten trotz mehrfacher Versuche bis dahin keine Lösung gefunden. Dabei war der Ausgangspunkt im GEOS Projekt gar nicht so ungünstig. Sämtliche Testfälle waren in einer relationalen Testfalldatenbank, wo sie mit den Anforderungen verknüpft waren. Jeder Testfall war einer Anforderung zugeordnet, jede Anforderung konnte mehrere Testfälle haben. Was fehlte, war die Verbindung von den Anforderungen zu dem Code. Ich hatte früher vorgeschlagen, dass

die Entwickler in dem Kommentarkopf zu jeder Methode bzw. bzw. zu jeder C-Funktion die sie schrieben, einen Verweis auf die Anforderung einbauen, die von dem jeweiligen Codebaustein erfüllt wird. Ich hatte sogar einen Template zu diesem Zweck entworfen und habe mit dem Code-Auditor geprüft, ob er ausgefüllt ist. Ich wollte eine Tabelle der Code-zu-Anforderungen Beziehungen aufbauen. Wie gegen die Meisten meiner Vorstoßes haben die Entwickler auch hiergegen protestiert, so habe ich das Vorhaben aufgegeben. Jetzt standen die Tester vor dieser Kluft und wussten nicht, wie sie diese überbrücken sollten. Ich bekam einen Auftrag von der Leiterin der Testabteilung, es zu versuchen.

Zunächst habe ich versucht, über die statische Analyse die Funktions- und Datennamen in dem Code mit den Namen der Anforderungsbeschreibungen abzugleichen. Wenn es in einem Codebaustein viele Treffer gab, habe ich den Codebaustein der Anforderungen zugeordnet. Wo Unklarheit herrschte, habe ich mit den Entwicklern gesprochen. Leider waren die Ergebnisse dieser Untersuchung unzureichend. Bei allzu vielen Code-Moduln war es nicht möglich, sie eindeutig Anforderungen zuzuordnen, auch nicht über die Nachfrage bei den Entwicklern. Viele hatten die Firma schon verlassen und die Anderen wussten gar nicht mehr, welche Anforderungen sie hätten erfüllen müssen. Der Bruch zwischen den Anforderungen und dem Code war auf dieser Weise nicht zu schließen [SnDo99].

Der zweite Versuch war über die dynamische Analyse. Um die Testüberdeckung zu messen, hatte ich einen Instrumentor entwickelt, der in jedem Codezweig einen Durchlaufzähler einbaut. Mit diesen Proben war es auch möglich, die Uhrzeit zu registrieren. Dadurch ließ sich feststellen, genau zu welcher Uhrzeit jede Funktion, z.B. jede Methode, ausgeführt wurde.

Der Zuständige für die Fehlerverfolgung, Dr. H. wusste, dass der verwendete Testmonitor die Anfangs- und Endzeit eines jeden Testfalles registriert. Es war sein Vorschlag, die Codebausteine auf dieser Weise den Testfällen zuzuordnen. Alle Codezweige, die zwischen der Startzeit und Endzeit eines Testfalles ausgeführt werden, gehören eben zu diesem Testfall. Ich griff die Idee auf und verwirklichte sie in einem neuen Werkzeug. Die Ausgabe des Werkzeuges war eine Tabelle der Methoden und Funktionen geordnet nach Klasse und Modul mit den Kennzeichen der Testfälle, die sie durchlaufen. Mit dieser Tabelle war es möglich die Code-Funktionen den Testfällen zu zuordne. Leider bin ich nicht dazu gekommen, dieses Vorhaben durchzusetzen. Ich musste vorher die Firma verlassen [Sned04].

9.4 Aufwandsschätzung der Wartung und Weiterentwicklung

Vielleicht mein größter Beitrag zum GEOS Projekt war die Messung der Systeme zwecks der Release-Aufwandsschätzung. Bis dahin gab es in der einschlägigen Literatur kaum etwas über die Planung und Schätzung der Wartung und Weiterentwicklung vorhandener Großsysteme. Das Thema „Software Produktmanagement“ war stark unterentwickelt im Verhältnis zu „Software Projektmanagement“. Alle waren auf die Entwicklung neuer Systeme fokussiert und niemand hat sich um die Erhaltung der bestehenden Systeme gekümmert. Franz Lehner und ich haben das Thema Wartungsmanagement in unseren beiden Büchern Softwarewartung am Anfang der 90er Jahren angesprochen aber tiefer sind wir nicht reingegangen. Jetzt im Rahmen meiner Tätigkeit im GEOS-Projekt kam ich darauf wieder zurück.

Projektion der Evolutionskosten

Based on this data the next Release of the FIDIS System will require:

Metric	Change Rate	Impact	Effort
Function-Points	current state	22 673	1134 PMs
	with 10% change	24 940	141 PMs
	with 20% change	27 208	283 PMs
	with 30% change	29 475	431 PMs
Object-Points	current state	116 104	1451 PMs
	with 10% change	127 714	181 PMs
	with 20% change	139 325	363 PMs
	with 30% change	150 935	536 PMs
Data-Points	current state	129 904	1624 PMs
	with 10% change	142 894	203 PMs
	with 20% change	155 886	406 PMs
	with 30% change	168 875	601 PMs
Statements	current state	420 000	1050 PMs
	with 10% change	462 000	131 PMs
	with 20% change	504 000	263 PMs
	with 30% change	546 000	388 PMs

Für mich war es offensichtlich, dass die Planung, Schätzung und Steuerung der Wartungsarbeit andere Methoden verlangten als bei einer Neuentwicklung der Fall ist. Bei der Wartung und Weiterentwicklung bestehender Softwaresystemen gibt es nicht ein Projekt sondern mehrere, die neben und nach einander stattfinden. Eigentlich ist jede Fehlerkorrektur zwischen Releases, bzw. jedes Hot-Fix ein eigenes Projekt. Der betroffene Entwickler, hier der Trouble Shooter“ erhält ein Auftrag – die

Fehlermeldung – und führt ihn durch. Zum Schluss muss er seine Korrektur testen und der Anwender muss ihn abnehmen. Mit dem Produkt GEOS gab es hunderte solcher Miniprojekte, die alle Zeit und Geld kosteten. Sie mussten geplant und geschätzt werden, zumindest in ihrer Summe. Die Schätzung der Fehlerkorrekturkosten setzt voraus, dass die Anzahl der Fehler pro Release vorausgesagt wird. Dies geschah aufgrund der bisherigen Fehlerstatistik verbunden mit dem Ausmaß der Testüberdeckung. Dazu kommt der mittlere Aufwand pro Fehlerkorrektur. Auch die Fehlerbeseitigung hat eine Produktivität und diese Produktivität gehört gemessen. Die Frage ist, wie lange braucht ein Mitarbeiter im Schnitt einen Fehler zu korrigieren. Die Antwort ist sehr umgebungsspezifisch und muss aus den Erfahrungswerten früheren Releases entnommen werden. Ich habe begonnen die Aufwandsstatistik auszuwerten und kam auf Werten zwischen 0,5 und 2,5 Personentage, je nach Systemgröße und -komplexität. Das war die Ausgangsbasis für die Schätzung der Fehlerbehebungskosten [Sned04].

Wartung beschränkt sich nicht auf die Fehlerkorrektur. Es werden auch Änderungen und Erweiterungen vorgenommen. Diese werden über „Change Requests“ beantragt. Bei der Erhaltung und Weiterentwicklung vom GEOS gab es jährlich mehrere Hundert Change Requests. Sie mussten gefiltert, genehmigt und in Releases gebündelt werden. Dies war die Aufgabe der Produktmanager. Meine Aufgabe war es die Releases zu schätzen. Die Voraussetzung dafür war die Messung des betroffenen Anteils des Codes. Ich musste erst feststellen welche Codebausteine durch die Änderung betroffen werden und welche hinzu kommen. Die Größe der betroffenen Bausteine wurde durch das Änderungsausmaß justiert und die Summe aller Einzeländerungen genommen. Die neuen Bausteine wurden aufgrund einer Textanalyse der Änderungsanträge geschätzt. Die beiden Schätzungen wurden zur einer gesamten Release-Schätzung zusammengefasst.

Unterstützt wurde ich in meinen Bemühungen die Wartungskosten in Griff zu bekommen durch zwei junge Kollegen, bzw. Kolleginnen, die für die Verwaltung der Fehlermeldungen und Change Requests zuständig waren. An Hand unserer gemeinsamen Erfahrung mit der Messung und der Versionsverwaltung haben wir entschieden ein Buch darüber zu schreiben – Software-Produktmanagement. Das Buch wurde 2004 von dpunkt Verlag in Heidelberg herausgegeben. Bis es so weit war, war ich nicht mehr bei SDS, aber das Buch hat sich gut verkauft [SHT04].

9.5 Produktivitätsmessung und Verletzung des Österreich. Arbeitsschutzes

Wie Herr P. zu sagen pflegte „Function-Points sind das Dow-Jones Index der Software Industrie“. Er brauchte diese Zahl, um die hohen Kosten der Entwicklung gegenüber den Geldgebern zu rechtfertigen, denn diese sind stets gestiegen. Bis Ende 1999 waren mehr als 300 Personen an den beiden Projekten GEOS und DRIVE beteiligt. DRIVE sollte neben GEOS als Code-Banking-System dienen. Es wurde Anfang 1998 begonnen mit einer neuen Mannschaft - größtenteils junge Leite direkt von der Universität und bis Mitte 2001 hatte es eine Größe von 5556 C-Source Modulen mit 1.680 Kilo Codezeilen, 623 Kilo Anweisungen und 50.000 Function-Points erreicht. Diese Codemenge kam zu der GEOS Codemenge hinzu, so dass ich bei der Source Analyse gleich zwei große Systeme zu verarbeiten hatte [SnBr03].

Ein Hauptanliegen des SDS Managements war es, die Entwicklung neuer Komponente und die Enthaltung alter Komponente abzuschätzen. Ich wusste, dass ich dazu eine Produktivitätsdatenbank brauchte, also habe ich schon 1999 begonnen, nicht nur die Quantität und Qualität der einzelnen Systeme zu messen, sondern auch die Produktivität der zuständigen Teams zu messen. Es war unumgänglich die Menge der produzierten Function-Points, Object-Points und Anweisungen, die pro Personenmonat produziert wurden, zu zählen um den Aufwand künftiger Projekte hochrechnen zu können. Hierbei stellten sich jedoch die ersten Probleme mit den Kollegen heraus. Sie hatten sich irgendwie damit abgefunden, dass man die Anzahl ihrer Fehler und Mängel zählten, jetzt aber sollte auch noch gezählt werden, wie viel Code sie pro Monat ausliefern. Das kam nicht gut an. Solange, als die Messung auf einer relativen abstrakten Ebene stattfand – auf System-, Teilsystem- oder Komponentenebene – fühlte sich keiner betroffen, aber als ich begann auf der Modulebene zu messen, ging das zu weit, denn die Module konnten einzelnen Mitarbeitern zugeordnet werden und kam heraus, dass die Qualität mancher Module weit unter dem Durchschnitt lag. Es ging aber noch ein Schritt weiter. Ich wollte nachweisen, dass die Qualität des Codes mit der Fehlerrate korreliert. Bei Moduln mit der niedrigsten Codequalität ist dies tatsächlich gelungen. Das hat einige der betroffenen Entwickler mir übel genommen. Sie meinten, ich hatte sie ungerecht bewertet. Dies sollte für mich später Folgen haben.

9.6 Verbahnung in die Testabteilung

Ende 2002 musste mein Chef und Gönner, der Herr P. gehen und wurde durch einen Herrn Dr. B. aus Deutschland ersetzt. Das österreichische Bankenkonsortium war mit dem Fortschritt der GEOS Entwicklung unzufrieden und war nicht mehr bereit die hohen Kosten weiter zu tragen. Ein neuer Manager ist aus Deutschland gekommen um deutsche Ordnung in den Wiener Chaos zu bringen. Das erste, was er machte, war einen Betriebsrat zu installieren. Der Betriebsrat wurde von den Mitarbeitern gewählt

und als Vorsitzenden des Rates wurde gerade jener Entwickler gewählt, der sich am meisten über meine Qualitätsmessung geärgert hat. Die erste Handlung des Betriebsrates war die Softwaremessung einzustellen. Es stellte sich heraus, dass eine Leistungsmessung ohne das Einverständnis der Betroffenen gegen die Österreichischer Arbeitsverfassung stößt. Ich bekam einen gesalzenen Brief von der Personalabteilung, der mir verbat weitere Messungen dieser Art ohne Absprache mit dem Betriebsrat auszuführen. Ich bin nämlich damit gegen die Österreichische Arbeitsverfassung gestoßen.

Der neue Geschäftsführer hätte mich eigentlich entlassen sollen, aber noch wollte er mich behalten. Er versetzte mich in die Testabteilung, die ultimative Bestrafung für IT-Mitarbeiter, die sich etwas zu Schulden kommen lassen. Ich wollte bei der SDS bleiben und habe die Versetzung angenommen. Die Entlassung sollte ein Jahr später folgen.

In der Testabteilung kam ich zurück zu jener Tätigkeit, wofür ich ursprünglich bestellt wurde. Allerdings sollte ich keine neue Testwerkzeuge entwickeln, sondern den Testprozess untersuchen und verbessern. Die Zahl der Kundenfehlermeldungen war immer noch zu hoch. Von 100 gemeldeten Fehlern im System kamen 72 von den eigenen Testern und 28 von den Kunden. Laut der neuen Geschäftsführung mussten mindestens 85% von dem Testteam aufgedeckt werden. Außerdem dauerte der Systemtest immer noch zu lange und verzögerte damit die Auslieferung neuer Releases. Kurzum, es sollte schneller und effektiver getestet werden [SnJu06].

Ein Problem war organisatorischer Art, nämlich die Trennung der Tester von den Entwicklern. Damals herrschte die Meinung, der Test musste unabhängig von der Entwicklung erfolgen, die Tester durften von den Entwicklern nicht beeinflusst werden. Ich hatte in meinen Seminaren und Büchern über Testen dazu beigetragen. Alles hat Vor- und Nachteile. Der Nachteil der Trennung lag in der mangelnden Kommunikation. Der Grundsatz eines unabhängigen Tests basiert auf der Annahme, dass die Tester wissen, wie das System sich verhalten sollte. Entweder sind sie Sachgebiet Experten, wie die Endanwender selbst, oder sie haben eine ausführliche Beschreibung des Sollverhaltens, bzw. eine detaillierte Spezifikation. In diesem Fall fehlten beide. Die Tester waren größtenteils Studenten oder Hilfskräfte mit wenig Wissen über den Handel mit Aktien. Die Systemspezifikation – obwohl mit einem Werkzeug erzeugt – war trotzdem sehr oberflächlich und auch nicht in Einklang mit dem Code. Die Konzeptler – so hießen sie – waren auch von den Entwicklern getrennt. Wie die Tester, saßen sie in einem anderen Stockwerk und lebten in einer eigenen Welt. Ihre Vorstellungen, wie die einzelnen Anwendungsfälle abzuarbeiten sind, waren oft nicht ohne

erhebliche Anpassungen nicht programmierbar, oder sie wurden von den Entwicklern falsch verstanden. Um adäquat spezifizieren zu können, muss ein Analytiker mit den Möglichkeiten der Programmierung gut vertraut sein, sonst gibt er etwas vor, was so nicht realisierbar ist und der Entwickler macht was Anderes daraus. Durch die räumliche Trennung fehlte die rechtzeitige Rückkopplung von den Entwicklern zu den Konzeptlern. Sie erfolgte erst durch die Fehlermeldungen der Tester, die merkten, dass irgendetwas nicht stimmt.

System	Testfälle	Fehler entdeckt	Vom Tester entdeckte Fehler	Vom Kunden entdeckte Fehler	%Tester entdeckt
FIVS	36735	5269	4372	897	0.830
GEOS	37521	828	632	196	0.764
KMS	793	114	78	36	0.685
NOST	1343	1342	1086	256	0.810
STS	1411	840	706	134	0.841
ZSYS	2145	206	190	16	0.923
GESAMT	79155	8599	7064	1535	0.822

Ein anderes Problem war psychologischer Art. Der Systemtest war zu diesem Zeitpunkt nur semi-automatisch. Es gab zwar eine Werkzeugunterstützung für die Eingabe der Testdaten, aber die Tester, bzw. die Testerinnen – ein Großteil der Tester war weiblich - , mussten die einzelnen Datenwerte eintippen und die einzelnen Mausklicks betätigen. Die eingegebenen Oberflächen wurden von einem Capture/Replay Tool aufgezeichnet und später zurückgespielt, aber die Tester mussten oft daran etwas ändern oder korrigieren. Außerdem mussten sie bei allen neuen und geänderten Transaktionen die Ausgaben visuell prüfen und bestätigen. Dazu mussten sie wissen welche Ausgabewerte es geben sollte. Entweder rechneten sie nach oder entnahmen

sie aus der Spezifikation Das Testen erforderte jedenfalls äußerste Konzentration und viel Disziplin.

Dabei ist mir aufgefallen, dass die Mädels dieser Aufgabe besser gewachsen sind. Sie haben die größere Aufmerksamkeit aufgebracht und waren disziplinierter, und haben ihre Arbeit auch besser organisiert. Die Jungs neigten dazu zu spielen. Sie ließen sich leicht ablenken und waren insgesamt weniger produktiv. Die Testarbeit hat sie eher gelangweilt. Wenn Fehler auftraten, haben sie sie oft gar nicht bemerkt. Die Mädels haben dagegen alles genau registriert, Sie waren unter diesen fabrikähnlichen Umständen zweifelsohne die besseren Tester. Vielleicht wäre es anders ausgefallen, wenn die Buben sich mehr mit ihrer Arbeit identifiziert hätten [Sned03].

Ein drittes Problem war technischer Art. Zu dem Zeitpunkt, als ich zu der Testabteilung gekommen bin, gab es über 80.000 Testfälle für den Systemtest. Bei jedem neuen Release hat man alle wiederholt, die zu dem geänderten Teilsystem gehörten, auch wenn nur einige Codezeilen in wenigen Moduln sich geändert hatten. Das nahm viel Testaufwand und große Rechenkapazität in Anspruch. Die Leiterin der Testabteilung hatte das Problem erkannt und suchte nach einer Möglichkeit nur jene Testfälle zu wiederholen, die die geänderten Module betrafen. Deshalb der Auftrag den sie mir erteilte, herauszubekommen welche Codebausteine von welchen Testfällen getestet werden. Ich habe versucht, wie schon oben geschildert, die Frage zu beantworten. Leider hat sich herausgestellt, dass dieses Problem, wie damals das mit dem Modultest, gar nicht so einfach zu lösen war. Ich brauchte Zeit und vor allem brauchte ich die Mitwirkung der Anderen [Sned05].

9.7 Abschied von der SDS

Meine Vorschläge, die Tester mit den Entwicklern enger zu integrieren und den Test statt nur halb, wirklich voll zu automatisieren, wurden von der Leiterin der Testabteilung angenommen. Aber mein Vorhaben, die Testfälle mit Codebausteinen zusammenzuführen blieb auf der Strecke. Wir hätten den gesamt Code instrumentieren und nochmals durchtesten müssen. Folglich wurde es nie zu Ende geführt.

Anfang 2003 hatte das Bankenkonsortium entschieden, das GEOS Projekt loszuwerden und die Entwicklungsfirma SDS zu verkaufen. Voraussetzung für den Verkauf war eine drastische Verkleinerung der Entwicklungsmannschaft. Von 350 Mitarbeitern mussten 140 gehen. Die Geschäftsführung und der Betriebsrat sollten entscheiden, wer geht. Ich stand natürlich ganz oben auf der Liste. Im Mai 2003 bekam ich per Email mitgeteilt, dass ich bis Ende Juli meine Arbeiten abschließen und die Firma verlassen soll. Es gab ein kleines Fest in der Testabteilung bei der ich von der

Testabteilungsleiterin feierlich verabschiedet wurde. Meine Zeit als Test- und Qualitätsberater in der SDS war zu Ende. Übrig blieb ein Buch – Testen objektorientierter Software, das ich mit Prof. Mario Winter von der Fachhochschule Köln verfasste [SnWi02]. Darin wurden meine schwer erworbenen Kenntnisse zu diesem Thema zusammengefasst.

9.8 Beginn meiner Lehrtätigkeit an der Hochschule

Schon 1998 wurde ich von den italienischen Kollegen in der internationalen Software-Maintenance Community eingeladen in ihrem Masters Programm für Software-Engineers mitzumachen. Das Programm der Universität Benevento um Berufstätigen ein Zusatzstudium als Software-Engineer an zu bieten wurde von der EU gefördert. Mit den Geldern aus Brüssel hat die Universität namhaften Persönlichkeiten aus der Software-Engineering Welt nach Benevento geholt um einwöchige Kurse in englischer Sprache zu halten. Ich habe einen Kurs über Reengineering gehalten. Später habe ich auch noch Aufwandsschätzung und XML gelehrt. So konnte ich meine ersten Erfahrungen als Hochschullehrer sammeln. Zu diesem Zeitpunkt hatte ich schon über 25 Jahre Erfahrung als Ausbilder in der deutschen Industrie gehabt. Jetzt sollte ich lernen wie man mit Studenten umgeht. Es war für mich eine neue Erfahrung. Einmal im Jahr für fünf Jahre bin ich mit der Bahn nach Benevento gefahren um dort eine Woche zu lehren. Das sollte der Anfang einer neuen Nebentätigkeit sein.

Mein langjähriger Freund und Wegbegleiter auf dem Gebiet der Softwarewartung, Professor Franz Lehner von der Universität Regensburg, hatte mit viel Mühe durchgesetzt, dass ich auch in Bayern an der Hochschule ohne Promotion lehren dürfte. Im Sommer 2000 begann ich mit einem Seminar „Software Engineering für Wirtschaftsinformatiker“ an der Universität Regensburg. Wir haben zusammen ein Curriculum für Software Engineering in der Wirtschaftsinformatik ausgearbeitet und in der Zeitschrift für Wirtschaftsinformatik veröffentlicht [LeSn04]

. Ein Semester lehrte ich Softwareentwicklung, das nächste Software-Projektmanagement. Das hat sich jedes Jahr wiederholt für 15 Jahre. Kurz danach wurde ich von meinem ungarischen Kollegen in dem IEEE Maintenance Programm Committee – Tibor Gymothy – gebeten ein Wochenkurs für seine Doktoranden an der Universität Szeged zu halten. Das hat sich jedes Jahr im Frühjahr wiederholt. Ich habe in Szeged abwechselnd Software-Testtechnologie und Software Reverse- und Reenginnering gelehrt. Inzwischen hat Franz Lehner einen Lehrauftrag an der Uni Passau erhalten. Für drei Jahre habe ich ihn auch dort unterstützt. Über die Uni Passau kam ich zur Uni Budapest. Dort gab es ein deutschsprachiges Lehrprogramm für Wirt-

schaftsingenieure in Kooperation mit der Uni Passau. Der Lehrgang „Software Engineering für Betriebswirte“ wurde auch angeboten. Da ich schon in Budapest zuhause war kam mir das sehr gelegen. Ich habe dort Projektmanagement und Qualitätssicherung gelehrt. Schließlich kam Professor Jürgen Ebert von der Universität Koblenz auf mich zu und bat mir die Möglichkeit auch dort im Wintersemester zu lehren. Das war immer in der Karnevalzeit. Softwaretest und Softwarewartung waren dort meine Themen. So kam es dass ich in meiner Zeit bei der SDS in Wien gleichzeitig an sechs Universitäten in drei Länder unterrichtet habe.

Zusätzlich zu der Projektarbeit, der Beratungstätigkeit und der Lehre kam noch die wissenschaftliche Arbeit. Bereits im Jahre 1998 wurde ich in den Steuerungsausschuss der europäischen Maintenance und Reengineering Konferenz berufen. Im Jahre 2001 kam ich noch in den Steuerungsausschuss für die internationale Software Maintenance Konferenz. In dieser Kapazität habe ich 2002 die europäische CSMR und 2005 die internationale ICSM in Budapest als General Chair organisiert und veranstaltet. Im Rückblick kann ich feststellen, dass die Jahre von 2000 bis 2006 der Höhepunkt in meiner akademischen Laufbahn war. Ich habe Software Engineering, Software-Reengineering, Softwaretest und Softwarewartung an sechs verschiedenen Universitäten in drei verschiedenen Ländern – Italien, Ungarn und Deutschland – gelehrt. Außerdem habe ich dem Steuerungsausschuss von zwei internationalen Konferenzen gedient – ICSM und CSMR. Und das alles neben meiner Tätigkeit als Tester bei der SDS und später bei der ANECON. Der geschäftliche Höhepunkt meines Lebens war schon längst überschritten. Diesen hatte ich 20 Jahre früher erreicht. Ich musste aber dort weitermachen um meinen Lebensunterhalt zu verdienen. Als Lehrbeauftragte kann einer kaum über die Runden kommen. Dies zeigt, dass geschäftlicher und akademischer Erfolg zwei paar Stiefel sind. Sie haben miteinander nichts zu tun. Im Gegenteil, je tiefer man in der akademischen Welt steckt, desto weiter entfernt ist man von der beruflichen Praxis. Die Ziele dieser beiden Welten stehen konträr zu einander.

10 Neuanfang als Tester in Wien

Nach meiner Entlassung aus der SDS blieb ich vier Monate lang ohne Arbeit. Ich wollte nach Deutschland zurück und wandte mich an alten Bekannten in München und anderswo die eine Firma hatten, aber keiner konnte mich gebrauchen. Ich war ihnen zu alt und passte nicht mehr in ihre Unternehmenskultur hinein. Ich musste erkennen, dass mit 63 Jahren man aus dem Rennen ist. Das ist die Zeit der Frühpensionierung, ich war jedoch nicht bereit die Flinte ins Korn zu werfen. Gesundheitlich war ich noch in einem guten Zustand und der Wille war dort weiterzumachen. Außerdem wäre meine Rente zu wenig gewesen. Ich hatte das Haus in Bayern noch abzuzahlen. Ich musste nur jemanden finden der bereit war mir Projekte zu geben.

Deshalb habe ich die Zeit benutzt meine berufliche Qualifikation noch im fortgeschrittenen Alter nachzubessern. Ich habe in dieser Zwischenzeit die Prüfung zum „Certified Tester“ gemacht und auch die Prüfung zum „Certfied Function-Point Schätzer“. Ich habe das alte Buch über Software-Produktmanagement mit meinen ehemaligen SDS Kollegen fertiggestellt und ein neues Buch – Software-Projektkalkulation begonnen. Es erschien erst Anfang 2005 [Sned05]. Zum Glück habe ich an den GI-Fachkonferenzen weiter teilgenommen. Ich war weiterhin in vier verschiedene Fachgruppen – Test und Analyse, Software-Metrik, Reengineering und Software Management. Auf einer solchen Konferenz in Magdeburg saß ich einer jungen Frau aus Wien gegenüber. Ich erzählte ihr von meinem Schicksal bei der SDS und sie schlug mir vor mich bei ihrer Firma in Wien vorstellig zu werden. Ihr Chef – der Manfred Baumgartner – war dabei eine Testabteilung aufzubauen und er würde Tester suchen. Insofern passte es ganz gut, dass meine letzte Stelle bei der SDS in der Testabteilung war und dass ich gerade die „Certified Testerprüfung“ in Erlangen abgelegt hatte. Ich war unsicher ob ich mich als Tester mit 63 Jahren in der Praxis noch behaupten konnte, aber ich war bereit, es darauf ankommen zu lassen. Wenn ich weiterarbeiten wollte, blieb mir nichts Anderes übrig.

Also wiedermal auf nach Wien und mich vorstellen, diesmal in der alten AKH im 9. Bezirk. Es war zu meinem Vorteil dass ich zwei der ANECON Geschäftsführer aus früheren Zeiten noch kannte, der Einer von der UBS in Zürich und der Andere von meinen Vorträgen für die GI-Fachgruppe Test. Zum Glück gab es noch nicht so viele Tester auf dem Wiener Markt. Testen war als eigenständige Tätigkeit noch nicht anerkannt. Somit hatte der Chef der neugegründeter Testabteilung – Manfred Baumgartner - kein Problem damit mich trotz fortgeschritten Alters als freie Mitarbeiter in

seiner Testgruppe aufzunehmen. Also konnte ich meine alte Lebensweise wieder fortsetzen. Ich kehrte zu meiner früheren Pension in Oberlaa zurück und fuhr mit dem Rad jeden Tag in die Stadt hinein.

10.1 Fehlersuche in einem Telekommunikationsbetrieb

Meine erste Aufgabe bei der ANECON passte nicht gerade zu meinem bisherigen Erfahrungsprofil aber das ist nicht selten der Fall. In der IT-Welt wird einer immer vor Aufgaben gestellt, auf die man nicht vorbereitet ist und muss vor dem Kunden so tun als ob man in diesem Bereich schon immer zuhause war. Der Kunde hier war ein schwerfälliges, halbstaatliches Unternehmen im Bereich der Telekommunikation, bei dem der Begriff „Pragmatisierung" noch seine Gültigkeit hatte. Diesen Begriff habe ich in meiner Wanderungen durch die Wiener Betrieben immer öfter zu hören bekommen. Er verkörperte die Sehnsucht der Wiener nach Sicherheit und Stabilität. Wer pragmatisiert ist, ist unkündbar. Immer wenn es kriselte und nichts mehr ging, wurde ich von dem Kundenmitarbeitern daran erinnert, dass egal was geschieht, sie bleiben in ihrer Position unantastbar. Damit wurde die Diskussion um unangenehme Organisationsfragen gleich abgewürgt. Als Opfer der Unwägbarkeiten der kapitalistischen Marktwirtschaft kann ich diesen Wünsch abgesichert zu sein durchaus verstehen. Anderseits hat es fatale Konsequenzen für die Wettbewerbsfähigkeit der Unternehmen wie das Schicksal der sozialistischer Länder und der österreichischen Staatsbetriebe nur so gut belegt.

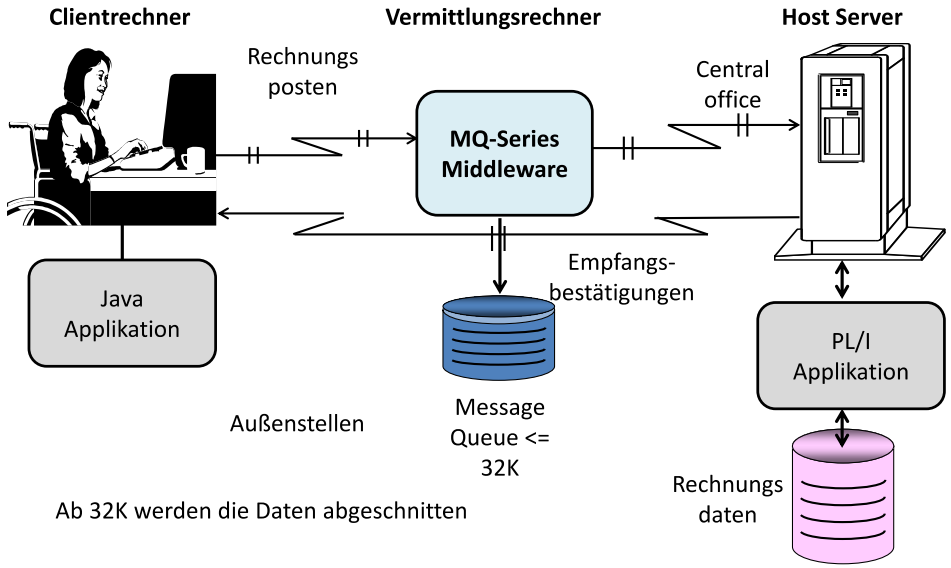
So war es auch bei diesem Projekt für die ANECON. Seit Monaten schlug sich der Kunde mit Software Fehlern herum, die den IT-Betrieb erheblich störten. Keiner dort war besonders motiviert sie herauszufinden. Meine Aufgabe bestand darin, den Kundenmitarbeitern beizubringen, wie sie Fehler finden sollten. Dies ist eine der häufigsten Arbeiten in der IT-Praxis, „Trouble Shooting" nennen es die Amerikaner, aber eine Arbeit, die keiner machen will. Auch wenn sie wollten, wissen die meisten nicht, wie sie ansetzen sollten. Es gibt kaum Kurse oder Literatur zu diesem ungeliebten Thema. Kein echter Fachmann will sich mit sowas banalen beschäftigen, erst recht nicht wenn die Fehler nicht von ihm stammen. Ergo hatte ich wenige Anhaltspunkte.

Mein Ansatz war sehr theoretisch und deshalb hier in dieser Situation wenig hilfreich. Ich hielt ein Workshop zum Thema „Software Trouble Shooting" – Fehlerdiagnose und Debugging". Der zweitägige Workshop umfasste sieben Lehreinheiten:

- Fehlermanagement: wie man mit Fehlermeldungen umgeht
- Fehlerursachen: wie und wo Software Fehler entstehen
- Fehlerdiagnose: wie Fehler einzukreisen und festzusetzen sind

- Fehlertypen: die häufigsten Fehlertypen in den Sprachen des Kunden (PL/I und Java)
- Debuggingtechniken: Techniken und Tools für die Fehlerlokalisierung
- Fehlerkorrekturen: wie man vorgeht, die gefundenen Fehler zu isolieren und zu korrigieren, ohne unerwünschte Nebenwirkungen
- Fallstudien: Beispiele von Softwarefehlern und wie sie werden

Datenübertragung bei einer Telekommunikationsfirma



Um dieses Workshop aufzubauen grab ich die wenige Literatur die ich zum Thema „Software Debugging“ hatte wieder heraus [Brow73] und versuchte ich sie zu aktualisieren und ergänzen [Dunn84]. Das Workshop war vielleicht hilfreich, die Kenntnisse der Entwickler bezüglich der Fehleranalyse zu vertiefen, aber an die wahren Fehlerursachen in diesen Betrieb kam es nicht an. Die wahren Probleme waren eher organisatorischer Art. Die IT Abteilung war aufgeteilt in zwei Gruppen - eine für die Server-Software auf dem IBM-Mainframe und eine für die Clientsysteme auf verteilten Unix Rechnern. Jede Gruppe kümmerte sich um sein eigenes Gebiet. Es gab weder ein übergeordneter Produktmanager noch ein allgemeiner Systemarchitekt. Die Client-Gruppe, die mehrheitlich aus ukrainischen Leiharbeitern bestand, entwickelte

die neuen Java Frontend-Applikationen, die Server-Gruppe, bestehend aus altgedienten, pragmatisierten Angestellten, pflegte ihre alte PL/I Applikationen auf dem Host. Die Client-Applikationen forderten Daten von dem Serversystem. Diese holte die Daten aus der zentralen Datenbank und übergab sie an das Client. Für die Übertragung der Daten wurde das Mittelware-Produkt „MQ-Series“ von der IBM benutzt. Keiner fühlte sich zuständig für das Niemandsland zwischen Frontend und Backend.

Irgendwo in der Dokumentation zu MQ-Series stand, dass die maximale Länge einer Nachricht 32 K Bytes beträgt. Diese Aussage hat keiner gelesen. Wenn die Nachricht länger war, wurde der Rest einfach abgeschnitten – ohne Meldung. Man könnte behaupten, dies wäre ein Mangel des Produktes gewesen, aber der Produktlieferant hätte entgegnen können, die Anwender sollten seine Dokumentation lesen. Normalerweise war die 32K für diese Anwendung genügend, aber es gab immer wieder Fälle, wo sie überschritten wurde. Dann meinte jede Gruppe, die andere Gruppe hätte die Daten verloren. Es ging Monate lang hin und her, bis ich endlich auf die Idee kam, durch die Produktdokumentation zu blättern und prompt fiel mir die Aussage auf. Wäre ich mit dem MQ-Series besser vertraut gewesen, hätte ich gleich Verdacht geschöpft. So ist wertvolle Zeit und viel Aufwand darauf gegangen, diesen Fehler in der Anwendungssoftware zu suchen. Um die Einschränkungen zu umgehen, mussten die Host-Programmierer ihre Rückgabenachrichten eben auf splitten und mehrere zurücksenden. Dieses Verfahren verursachte mehr Arbeit seitens der Hostentwickler aber damit war das Problem gelöst. Ich hatte keine große Figur gemacht und berechtigterweise war der Kunde nicht gerade mit meiner Dienstleistung zufrieden. Ich hätte die technische Umgebung – hier die IBM Systems Applikation Architecture – SAA – besser kennen sollen. Das MQ-Series war ein alter Hut [Groc91]. Ich hätte mich besser damit auskennen sollen.

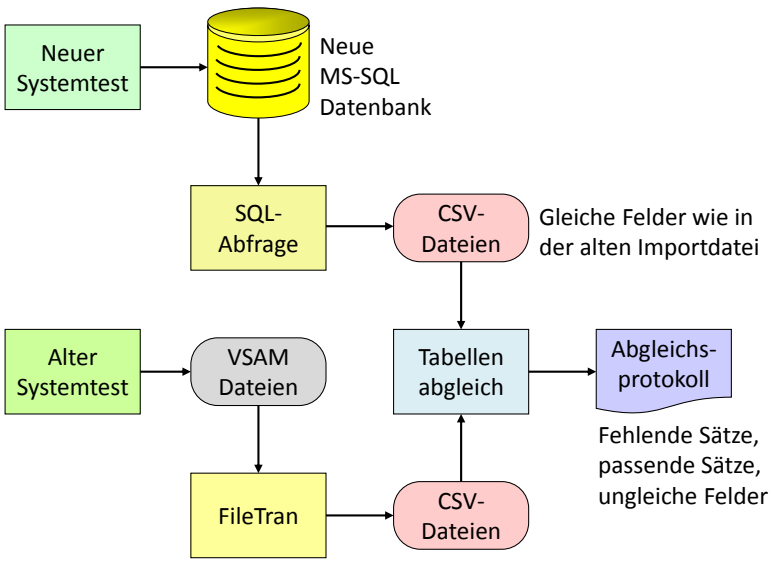
Später würde dieser Kunde sich an mir rächen, denn dieser – mein erster Kunde bei der ANECON – sollte auch der Letzte sein. Zehn Jahre später veranlasste dieser Kunde dass ich von der ANECON entlassen wurde. Für mich war diese Entlassung in Wirklichkeit die verspätete Rechnung für mein Versagen im ersten Projekt. Das Leben sorgt für Gerechtigkeit.

10.2 Regressionstest eines migrierten DotNet Systems

Schon während dieses ersten Beratungsprojektes für die Telekommunikationsfirma wurde ich zu einer anderen, halb-staatlichen Institution in Wien geschickt. Diese Institution war gerade dabei, ihre gesamte Software von dem IBM Mainframe auf einen PC-Netz zu migrieren. Die Zielumgebung war die DotNet Architektur von Microsoft.

Die alten Programme – in PL/I geschrieben – sollten in C# neu entwickelt werden. Die Stammdaten wurden in VSAM Dateien auf dem Mainframe gehalten. Sie sollten in Microsoft-SQL Datenbanken auf dem PC-Server migriert werden. Die Prozesse in denen die Rechnungen, Mahnungen und Gerichtsmitteilungen an die österreichischen Unternehmen hervorgingen liefen in regelmäßigen Abständen auf dem Mainframe. Sie wurden in einem Massendruckverfahren ausgedruckt und von Wien aus per Post an die gewerbetreibenden in den verschiedenen Bundesländern versendet. Dieses Verfahren wurde Anfang der 80er Jahren eingeführt und lief schon seit über 25 Jahren. Es war im wahrsten Sinne des Wortes ausgereift aber leider nicht mehr zeitgerecht. In Zukunft sollten die Aussendungen als XML Dateien ausgegeben und per elektronische Datenträger an die Landesämter übermittelt werden. Dort sollten sie lokal ausgedruckt und verteilt werden. Natürlich hatte jedes Bundesland ihre eigenen Spezifika, die auf dieser Weise besser bedient werden konnte.

Regressionstest durch den Abgleich der alten und neuen GUN-Daten



Als ich Spätherbst 2003 zu dem Projekt gekommen bin, waren sie schon fleißig dabei, die neuen Programme zu implementieren und die Daten zu migrieren. Auch hier gab es den einen älteren Mitarbeiter, der vom Anfang an dabei war und der alle

fachlichen Kenntnisse in seinem Kopf herumtrug – der Herr S. Jetzt stand er kurz vor dem Pensionsalter und niemand konnte ihn ersetzen. Das wusste er auch und dementsprechend war sein Verhältnis zum Projektleiter. Obwohl bereits in Pension hatte er eingewilligt, als Berater im neuen Ablöseprojekt mit zu arbeiten. Er verfasste die Fachkonzepte und stand den neuen Projektmitarbeitern für Fragen zur Verfügung. Auch ich musste mich öfters an ihn wenden, um als Tester die Zusammenhänge im System zu erfahren. Es erinnerte mich an das Krupp Projekt damals für die Bundesbahn als die Entwickler alle Schlange standen von der Tür des Allwissenden. Die Konzentration des ganzen Wissens über den fachlichen Inhalt der Applikation in einem Kopf war der Hauptengpass im Projekt. Es ist immer wieder das Gleiche. Ein junger, akademisch gebildeter Neuling wird von der Firmenleitung zum Projektleiter ernannt. Er soll den Aufbruch in die neue IT-Welt leiten, ist jedoch dabei auf die Unterstützung durch die Träger der alten Welt angewiesen. Diese sind nicht gerade begeistert vom bevorstehenden Systemwechsel, es sei denn sie stehen unmittelbar vor der Pensionierung. In diesem Fall war Herr S. schon pensioniert und hatte keinen Anlass den Systemwechsel zu behindern. Dennoch gab es immer wieder Spannungen zwischen ihm und der neuen Projektleitung. Dies scheint typisch für jeden Generationswechsel in der IT zu sein [BrSt95].

Validation der Rechnungsdaten im WKO Projekt

New:MitgliedsNummer Old:MitgliedsNummer	
Non-Matching Fields	**Non-Matching Values**
RecKey:3665	
New: Ocr_Zeile	84+_123000003665<_00500240168+_100204
Old: Ocr_Zeile	84+_123000003665<_00400240168+_100204
RecKey:3665	
New: RechnungsBetrag	264.10
Old: RechnungsBetrag	264.00
RecKey:6888	
New: RechnungsBetrag	167.00
Old: RechnungsBetrag	166.00
RecKey:8752	
New: RechnungsBetrag	199.00
Old: RechnungsBetrag	100.00
Total Number of old Records checked:	09
Number of old Records found in new File:	09
Total Number of new Records checked:	09
Number of new Records found in old File:	09
Total Number of Fields checked:	126
Total Number of non-Matching Fields:	04
Percentage of matching Fields:	97 %
Percentage of matching Records:	100 %

Das neue Gebührenabrechungssystem sollte in einer MS-DotNet Umgebung laufen und in C# und MS-SQL implementiert werden. Das alte System war mit CICS-PL/I auf dem Mainframe implementiert. Wie die meisten Systeme dieser Art bestand es aus einem Online-Frontend und einem Batch-backend. Verbunden waren die beiden Systemteile über eine gemeinsame Datenbasis, bestehend aus 8 große VSAM Dateien. Die Benutzeroberfläche bestand aus 24 CICS-BMS Masken für die Datenerfassung. Die Wesentlichen an dem System waren die Aussendungen, die für jedes Bundesland periodisch versendet wurden. Sie wurden in Wien am Hostrechner ausgedruckt, kuvertiert, etikettiert und an die über eine Million Empfänger per Post versendet. Mit dem neuen System sollte das anders werden. Die Aussendungen sollten als XML-Dateien an die Länderkammer elektronisch geleitet und dort nach einer länderspezifische Anpassung ausgeschickt werden. Die Stammdaten sollten weiterhin in der Wiener Zentrale geführt werden, aber als circa 120 einzelnen MS-SQL Tabellen auf PC-Rechnern. Die CICS-PL/I und PLI Batch Programme wurden von den eigenen Entwicklern mit Hilfe eines DotNet Spezialisten neu geschrieben. Der DotNet Spezialist konstruierte den Framework und lieferte das technische Knowhow, Herr S. lieferte das nötige Fachwissen. Er verfasste auch das Fachkonzept, bzw. das Lastenheft. Die Entwickler waren größtenteils ehemalige PL/I Programmierer die auf C# umgeschult wurden. Dies war ihr erstes DotNet Projekt. Ein Kollege von der ANECON hat sie unterstützt.

Die Aufgabe des Test-Teams war es, die Korrektheit der neuen Lösung nachzuweisen und dabei die Fehler an die Entwickler zu melden. Dies war mein erster echter Test eines Internetsystems obwohl ich früher mit einem Kollegen bei der SDS einen Artikel zu dem Thema geschrieben habe [SnGö00]. Mir als Teamleiter wurden zwei Kollegen von der ANECON bereitgestellt –ein freiberuflicher und ein Student von der TU Wien. Mein Plan war, dass diese beiden sich mit dem Test der Frontend-Transaktion befassen, während ich mich um die Backendprozesse kümmere. Ich habe vorgeschlagen, ein Capture-Replay Tool für den Dialogtest zu beschaffen aber die Projektleitung hat dies aus Kostengründen abgelehnt. Ich habe geahnt, dass wir den Test mehrfach wiederholen würden, die Projektleitung war aber am Anfang sehr optimistisch und meinte, ein manueller Test würde genügen. Wozu waren die zwei Dialogtester da, wenn nicht um die Dialogprozesse zu testen? Für den Test der Batchprozesse war ein Tool unentbehrlich. Dafür war aber auch kein Geld vorhanden, also entwickelte ich selbst einen Tool für den Datenabgleich. Normalerweise dürfte man ein Tool während eines Projektes nicht entwickeln. Entweder baut man es vorher oder nachher, aber hier blieb mir keine andere Wahl. Ich musste mehrere Millionen Datensätze und Druckzeilen abgleichen und jede geringste Abweichung erkennen. Dies

geht nur automatisch. Daher baute ich den ersten Prototyp zum DataTest, meistens abends in der Pension oder im Zug auf dem Weg von und nach Wien.

Während ich an dem Testwerkzeug herumgewerkelte habe, hielt ich die Projektleitung mit jeder Menge Testdokumente hin. Ich versuchte ihm zu erklären, dass das alles zur Testplanung gehört. Ich wusste von früher dass es nie gut ist Projektstunden gegen Werkzeugentwicklung zu buchen. Dafür hat der Geldgeber wenig Verständnis. Er möchte sehen, dass das Projekt vorankommt. Das Projekt kommt voran, wenn Dokumente und Berichte geliefert werden. Ich habe einen ausführlichen Testplan nach ANSI-IEEE-829 verfasst. Dazu habe ich etliche Begleitdokumente verfasst, um den Testplan zu vertiefen und zu ergänzen:

- ein Dokument über den Testprozess,
- ein Dokument über die Testorganisation,
- ein Dokument über die Testdatenverwaltung
- ein Dokument über die Testausführung und
- ein Dokument über die Testauswertung.

Während ich vordergründig das Projektmanagement mit Dokumenten geblendet habe, habe ich im Hintergrund das Abgleichs-Werkzeug fertiggestellt. In der Zwischenzeit haben meine beiden ANECON Kollegen die Oberflächentestfälle in Excel Tabellen spezifiziert. Die Testfälle ergaben sich aus einer Analyse der neuen DotNet Masken im Zusammenhang mit der Beschreibung der Vorgänge die uns Herr S. bereitstellte. Am Ende waren mehr als 2000 Testfälle für die 24 Dialog-Anwendungsfälle spezifiziert. Nach einem Monat war es so weit und sie konnten beginnen zu testen. Inzwischen war es allen klar, dass der Termin vom 1. Januar 2003 nicht zu halten war. Die neuen C# Programme waren völlig unstabil und kaum lauffähig. Außerdem hat die Datenkonversion nur Schrott produziert. Der dafür zuständige junge Entwickler – ein Externer von einer anderen Firma - war mit dieser Aufgabe total überfordert. Sowas hatte er noch nie gemacht. Die anderen Teammitglieder haben ihn deshalb gemobbt. Ich versuchte ihn zu helfen aber es war zu spät. Die Konversionsprogramme mussten neu konzipiert werden. Ergo wurde der Termin auf den 30. April verschoben.

Diese Verschiebung war für mich von Vorteil. Jetzt konnte ich die Zeit nutzen, mein DataTest Werkzeug fertigzustellen und auszutesten. Im neuen Jahr kam ein älterer, erfahrener Entwickler von der ANECON, der die Datenkonversionsprogramme neu schrieb. Nach nur sechs Wochen konnte ich beginnen, die migrierten Daten zu testen. Um dies zu bewerkstelligen musste ich die alten VSAM-Dateien von dem Mainframe herunterladen und in CSV – Comma Separated Value – Dateien auf dem PC-Arbeitsplatz umsetzen. Gleichzeitig habe ich die neuen SQL-Datenbanken mit

den dafür vorgesehenen Dienstprogrammen ebenfalls in CSV-Dateien umgewandelt. Nachher war es möglich, die CSV-Dateien die vom Mainframe stammten mit den konvertierten CSV-Dateien aus der MS-SQL Datenbank mit DataTest abzugleichen. Das Tool musste wegen der großen Datenmengen ein paarmal erweitert werden aber nach kurzer Zeit war das Ergebnis zufriedenstellend. Die verbleibenden Datenfehler aus der Datenmigration wurden mit dem Tool schnell erkannt. Nach einigen Wiederholungen lief die Datenkonversion absolut korrekt. Jetzt konnte ich anfangen die Aussendungen zu testen.

Der Test der Aussendungen war um einiges schwieriger als der Test der Datenmigration. Die alten Aussendungen waren Listen auf Endlospapier. Ich musste ein weiteres Tool entwickeln, um die Datenwerte anhand der Titel und Feldpositionen in der Liste zu erkennen. Die Feldwerte wurden aus den Listen in ein XML-Dokument mit den Titeln als Tags hineinkopiert. Die Lösung dieser Aufgabe erforderte einen völlig neuen Ansatz zur Textverarbeitung. Im Anschluss an dem Projekt habe ich ein wissenschaftliches Paper über dieses Verfahren mit dem Titel „Reengineering Reports“ für die IEEE Reverse Engineering Konferenz verfasst [Sned04]. Es war in der Tat eine echte Pionierarbeit.

Nachdem es gelungen ist die Listenfelder zu identifizieren, konnten die aus den alten Aussendungslisten abgeleiteten XML-Dateien mit den in XML-formatierten Aussendungsdateien aus dem neuen DotNet System verglichen werden. Hierbei wurden zunächst mengenweise Fehler aufgedeckt, die an die Entwickler zur Korrektur berichtet wurden. es hat einige Wiederholungen gegeben, bis die Aussendungsdaten einigermaßen richtig waren.

In der Zwischenzeit haben die anderen ANECON Tester die neuen Dialog-Anwendungsfälle getestet. Jetzt gab es über 3000 verschiedene Testfälle zu testen, wenn man alle Kombinationen von Eingabedaten betrachtet. Meine beiden Kollegen waren am Anfang sehr fleißig und testeten bis zu 32 Testfälle pro Tag. Dabei haben sie bei der Hälfte der 32 Testfälle mindestens einen Fehler gefunden. Sie saßen am Bildschirm mit drei offenen Fenstern, eins für die Testfalltabelle, eins für die Zieloberfläche und eins für die Fehlermeldungen. Aus der Testfalltabelle haben sie den nächsten Testfall genommen, in der Zieloberfläche habe sie die Testwerte zugewiesen und in der Fehlermeldungsmaske haben sie den Fehler berichtet [Sned05]. Das ging relative flott, bis bald Ermüdungserscheinungen auftraten.

Natürlich wurde der Termin 30. April auch nicht eingehalten und musste nochmals auf den Oktober verschoben werden, bis alle Fehler behoben sein sollten. Das Werk-

zeug DataTest lief mittlerweile einwandfrei. Ich konnte ab März die aktualisierte DotNet Datenbanken mit den aktualisierten VSAM Dateien vergleichen und ab April sogar die Auswertungslisten automatisch kontrollieren. Das größte Problem war für mich die Verwaltung der vielen Testdateien, mindestens vier pro Aussendungslauf. Der Test der Backendprozesse war jedoch endlich erfolgreich automatisiert. Wir konnten die Tests sowohl der Aussendung als auch der Berichtsgenerierung jederzeit wiederholen. Dies war ein großer Schritt nach vorn in der Testautomatisierung. [Sned05a]. Der Test der Frontend-Transaktionen war hingegen weiterhin manuell und deshalb noch viel zu aufwendig. Als bekannt wurde, dass wir den gesamten Test mit einer neuen Version wiederholen sollten, haben sich meine beiden Kollegen geweigert, die Arbeit fortzusetzen. Ihnen war es zu öde geworden. Der Testvertrag mit uns wurde deshalb gekündigt und ein Testteam aus Rumänien geholt. Ich habe meine Testwerkzeuge schweren Herzens an sie übergegeben und den rumänischen Testmanager, ein Doktor der Informatik aus Temesvar, in unseren Testprozess eingewiesen. Ich habe mich von dem Projekt verabschiedet. In der Schweiz waren es die Inder die mich ablösten, hier waren es die Rumänen. Ich hätte wissen müssen, dass die Wiener für so eine wiederholende, langwierige Arbeit nicht geeignet sind. Nachher hat dieser Kunde sich doch noch für ein automatisiertes Dialogtestwerkzeug entschieden.

10.3 Der Test eines Landes-Webportals in Dresden

Im Jänner 2005 wurde ich von der ANECON zu einen neuen Projekt im fernen Dresden abkommandiert. Sie hatten dort ein Testprojekt für das Land Sachsen gewonnen und ich sollte den Test planen und die Testfälle spezifizieren. Als Ausgangspunkt erhielt ich ein 110-Seitigs deutschsprachiges Word Dokument in dem 71 Anweisungsfälle spezifiziert wurden. Später kamen drei weitere Dokumente dazu. Den Testplan mit 16 Kapiteln nach der ANSI-Norm-829 habe ich innerhalb zwei Wochen fertiggestellt. Die anderen ANECON Mitarbeiter kamen erst Anfang Februar dazu. Neben der Erstellung des Testplans habe ich auch die Qualität des Lastenheftes kontrolliert. Dabei ist mir aufgefallen, dass die nicht-funktionalen Anforderungen wie Performanz, Zuverlässigkeit, Sicherheit und Wartbarkeit nicht quantifiziert und daher nicht messbar waren. Ich habe das Lastenheft an die Staatskanzlei zurückgeschickt und gebeten, sie mögen in jeder nicht-funktionalen Anforderung eine vergleichbare Zahl haben. Dies haben sie auch gemacht und ich konnte beginnen, die Testfälle aus dem Dokument abzuleiten.

Das Lastenheft wurde zwischen mir und zwei andern ANECON Mitarbeiter aufgeteilt. Jeder hat aus dem Text Testfälle entnommen. Es hat nicht lange gedauert bis ich zum Schluss kam, auch diese öde, repetitive Aufgabe ließe sich automatisieren. Wir

haben den Text nach Aktionen, die auszuführen, Zustände, die zu prüfen und Bedingungen, die zu erfüllen bzw. nicht zu erfüllen sind durchsucht. Ich war der Meinung, dies könnte ebenso gut von einem Textverarbeitungswerkzeug geleistet werden. Ich hatte schon im GEOS Projekt begonnen, die Spezifikationstexte zu analysieren um Testfälle zu erkennen. Dort war jede Anforderung eine getrennte kleine Textdatei im ASCII Textformat. Hier aber hätte ich mit einem einzigen großen Word-Dokument zu tun, das viele verschiedene Entitätentypen beinhaltete – Ziele, Anforderungen, Geschäftsregel, Geschäftsobjekte und Anwendungsfälle mit festdefinierten Attributen. Zum Glück wohnte ich einem abgelegenen Ort außerhalb Dresden – Cossebaude - und hatte abends viel Zeit daran zu arbeiten. So entstand das Tool „TestSpec“, um aus den deutschsprachigen Texten logische Testfälle zu gewinnen [Sned06]. Während ich tagsüber daran arbeitete, die Testfälle manuell zu spezifizieren, arbeitete ich nachts dran, dieselben Testfälle automatisch zu ermitteln. Das hatte den Vorteil, dass ich die automatisch erstellten Testfälle mit den manuellen vergleichen konnte, um die Funktionsfähigkeit des Tools zu bestätigen.

Testkennzahlen aus dem Web Portal Test

- Egovernment Web Site mit drei Applikationen
- 1844 Testfälle wurden spezifiziert
- 1495 Testfälle wurden ausgeführt = 81% Deckungsrate
- 176 Testfälle deckten Fehler auf = 12% Trefferrate
- 450 Fehler wurden durch den Test entdeckt = 18 per KDSI
- 56 Fehler wurden nach der Freigabe entdeckt = 11% von allen
- Systemtest entdeckte 89% aller bisher berichteten Fehler
- Systemtest dauerte 4 Monate von Jänner bis Mai 2005
- 392 Testertage wurden gebraucht
  - = 3.8 Testfälle pro Testertag
  - = 1.1 Fehler gefunden pro Testertag

Zum Schluss habe ich 946 Testfälle automatisch erzeugt, im Vergleich zu den 743, die ich zusammen mit meinen ANECON Kollegen spezifiziert habe. Da war der Stand Anfang März 2005. Es sind dann zwei weitere Tester aus der sächsischen Landesverwaltung dazu gekommen, so dass wir im März allein für den funktionalen Test zu fünft waren. Für den nicht funktionalen Test gab es ein separates Team mit wechselnden Spezialisten, je nachdem, was gerade getestet wurde. Es gaben neben dem üblichen Performance-Test noch ein Sicherheitstest, ein Usability Test und ein Integrationstest. Für den Test der Web-Services musste zusätzlich ein Testwerkzeug besorgt werden, das aber leider nicht hielt, was es versprach. Wir müssten selber einen Testrahmen schaffen, was gerade ausreichte, um die Aufgabe zu erfüllen. Dies war die Geburtsstunde des Webtest Tools, das zwar in diesem Projekt nie richtig zu Einsatz kam, aber später im Mittelpunkt meiner Forschung über Testautomation stand [Sned07].

Eigentlich war das Web-Portal Testprojekt ein seltener Erfolg wenn man beim Testprojekt überhaupt vom Erfolg reden kann. Was ist denn der Erfolg von einem Test? Da im Verlauf des Tests immer mehr Testfälle dazu kamen, war der Endstand 1844 Testfälle, wobei von denen aus Zeitgründen nur 1495 bzw. 81% ausgeführt wurden. Genau 450 Fehler wurden von den Testern gemeldet, das waren 18 per 1000 Java Anweisungen. Nach der Freigabe des Portals wurden bis zum Jahresende nochmals 56 Fehler von den Portalbenutzern, vor allem in der Landesverwaltung gemeldet. D.h. der Systemtest hatte 89% aller gemeldeten Fehler erkannt. Der Systemtest dauerte knapp über 4 Monate. Von Januar bis Mai 2005 wurden 392 Tester Tage gegen das Projekt gebucht [Sned05]. Die Landesregierung war mit den Ergebnissen zufrieden, weshalb die ANECON noch lange Jahre als Testservice Lieferant für das Land Sachsen blieb. Dies war in erster Linie dem Organisationstalent unseres Testmanagers Manfred Baumgartner zu verdanken. Ich wäre gerne in Dresden geblieben aber ein anderes Projekt hat auf mich gewartet. Kurz nach dem Abschluss des Projektes wurde ich ein wenig weiter westlich nach Ilmenau in Thüringen versetzt.

10.4 Testmessung für eine Versicherung an der Mosel

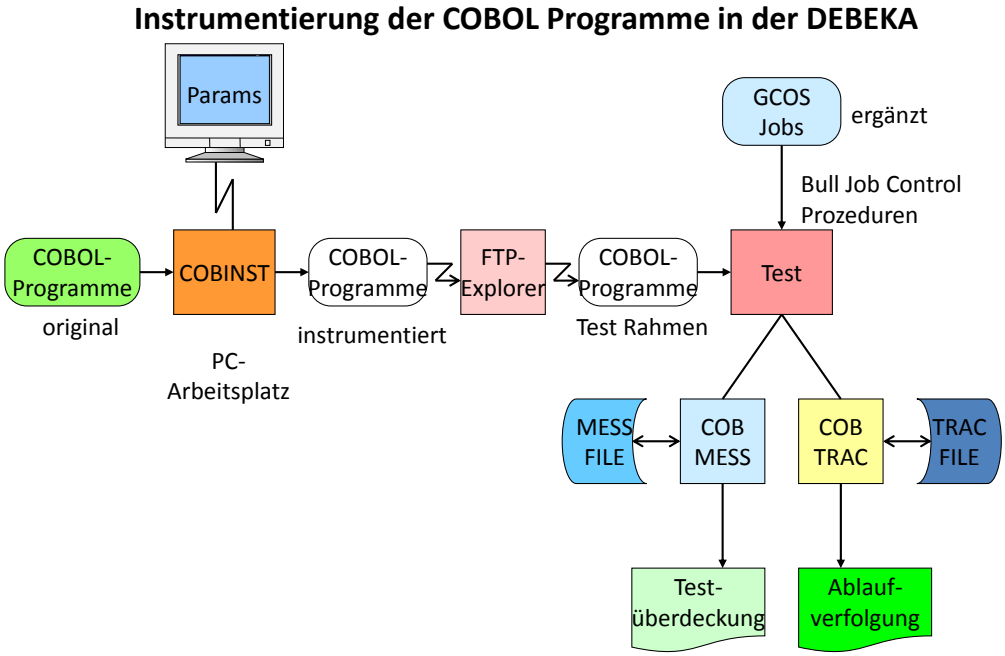
Zur Zeit des WKO-Testprojektes habe ich parallel dazu einen Kurs über Testen an der Uni Koblenz gehalten. Unter den Studenten war eine von der Versicherungsfirma auf der anderen Seite der Mosel. Diese Firma war dabei, ein größeres System zu testen und hatte dafür eine externe Beratungsfirma ins Haus geholt. Die Studentin meinte, ich soll mit dem dortigen IT-Leiter sprechen, was ich auch tat. Es stellte sich heraus, dass die Firma schon bei der dritten Versuch war, das komplexe System zu testen. Zwei andere Beratungshäuser hatten schon Testansätze probiert und sind damit ge-

scheitert. Mit dem erneuerten Versuch war der IT-Leiter etwas skeptisch und hat mich gebeten dort hineinzuschauen. Was die IT-Leitung wollte, war also ein Testprojekt-Audit.

Wie nicht anders zu erwarten war, war die zuständige Beratungsfirma von meinem Auftrag nicht gerade begeistert. Derr Projektleiter – ein junger, energischer und selbstbewusster Betriebswirt – versuchte mich gleich in eine andere Richtung zu lenken. Er wusste von meinen Toolentwicklungstätigkeit und hat mich gebeten, ein Werkzeug für die Testüberdeckungsmessung des Programmtests bereitzustellen. Fortan hatte ich also zwei Projekte,

- eins für die Bewertung des Testansatzes und
- eins für die Messung der Testüberdeckung.

Beide Projekte begannen nach dem Semesterabschluss im Juli 2005. Mit dem zweiten Projekt kam ich gut voran. Ich holte Erdös, Kati aus Budapest um mir zu helfen. Wir bekamen einen großen Büroraum und konnten ungestört arbeiten. Schon bis Anfang Oktober hatten wir ein funktionierendes Werkzeug auf die Beine gestellt, erstens weil wir so etwas schon mehrfach gebaut hatten und zweitens weil wir viele fertige Komponente aus früheren Projekten hatten.



Das Werkzeug COBINST bestand aus einem COBOL Präprozessor - der Instrumentor und einem COBOL Postprozessor – der Berichterstatter. Der Instrumentor baute Durchlaufzähler in jeden Zweig mit mehr als 800 COBOL Programme ein und fügte eine Routine hinzu, um die Zweigtabelle vor dem Test zu initialisieren und nach dem Test abzuspeichern. Der Postprozessor wertete die Zweigtabelle aus um einen Trace-Bericht der durchlaufenen Programmpfade, einen Testüberdeckungsbericht über die ausgeführten Zweige und einen Metrik Bericht über die Qualität des Tests auszugeben. Der Präprozessor und Postprozessor liefen auf dem PC, der Testprozessor lief auf dem Hostrechner. Die Entwickler waren mit dem Ergebnis von diesem Projekt sehr zufrieden. sie konnten sofort erkennen, in wie weit ihre Programme von den Testern getestet wurden [Sned04b].

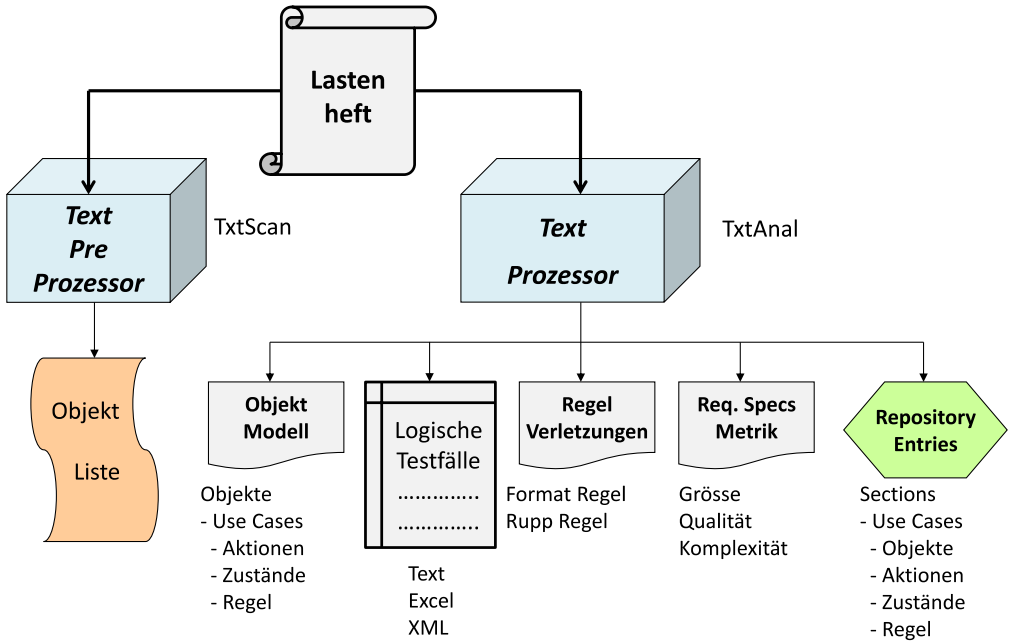
Mit den ersten Projekt – die Audit des Tests – war keiner so recht zufrieden, am wenigsten die Testberatungsfirma. Ich stellte fest, dass ihr Test absolut unzulänglich war. Sie hatten die Zeit und den Aufwand für den Test unterschätzt und hatten viel zu wenig Testfalle spezifiziert. Demzufolge war die Testüberdeckung viel zu niedrig. Sie

haben nur knapp über die Hälfte der Funktionalität getestet. Die Konsequenz daraus wäre gewesen, das Projekt zu verlängern und den Abnahmetermin abermals zu verschieben. Dieses Urteil dürfte nicht an die Geschäftsleitung geraten. Meine Berichte mussten erst an die Testprojektleitung gehen, die sie entsprechend umformulierte, ehe sie an die IT-Leitung weiterging. Ich wurde dem Testprojekt unterstellt und musste mich den Anordnungen des Projektleiters unterwerfen. Der ganze Zweck eines externen Testaudits war damit ad absurdum geführt. Ich war heil froh, das Projekt mit dem Beginn des nächsten Semesters Ende Oktober verlassen zu können.

10.5 Qualitätssicherung in einer mitteldeutschen Bundesbehörde

Über die Beziehung, die sie durch das Testprojekt in Dresden zu den Beratungsfirmen in Mitteldeutschland gewonnen hat, hatte ANECON ein weiteres QS-Projekt bei dem Bundesamt für Wasserbau und Binnenschifffahrt in Thüringen gewonnen. Dieses Projekt sollte vom Herbst 2005 bis Ende 2007 dauern. Die Aufgabe bestand darin, die Lastenhefte, die an potentielle Auftragnehmer ausgingen, zu prüfen und die Softwareprodukte die von dem Auftragnehmer zurückkamen, zu testen. Die Leitlinie hierfür war das V-Modell-XT, das bei den Bundesbehörden gerade eingeführt wurde. Diese Behörde war für 39 Softwareprojekte verantwortlich, verteilt über die Bundesrepublik, überall wo es Gewässer gab. Die Projekte reichten von der Wassertiefenvermessung bis zur Prüfung der Schifffahrtskapitäne.

Dieses sollte eins der angenehmsten Projekte werden dass ich je hatte, sowohl was die Arbeit als auch was die Umstände anbetrifft. Ich wohnte in einem alten Waldhotel, oben auf dem Berg oberhalb Ilmenaus, das früher in der DDR für das Parteikader reservier war. Nach dem Mauerfall hatte ein westdeutscher Unternehmer es aufgekauft und renoviert. Es gab zu diesem Zeitpunkt nur wenigen Gästen aber für mich war es genau das Richtige. Morgens bin ich mit dem Fahrrad in die Stadt runtergefahren – ein Höhenunterschied von circa 250 Meter und abends nach der Arbeit wieder hinauf. Bis zum Hotel habe ich es nie ganz geschafft, die letzten 50 Höhenmeter musste ich zu Fuß gehen – aber ich habe nie aufgehört es zu versuchen. Ich war inzwischen über 65 und musste alles tun um mein Gesundheit und mein Gewicht zu halten. Die Radfahrten durch den Thüringer Wald und die Abende in dem alten Waldhotel blieben mir in Erinnerung. Es gab kein Fernseher auf den Zimmern und ich konnte frühmorgens und abends ungestört an dem Buch „Der Systemtest“ arbeiten, die ich zusammen mit zwei ANECON Kollegen geschrieben habe. Es sollte unser meist verkauftes Buch werden [SnSB06].



Ergebnisse der Anforderungsanalyse bei der BAW

Auch die Arbeit in der Bundesanstalt passte sehr gut zu mir. Mit dem Qualitätsmanager – ein promovierter Betriebswirt aus Sachsen - habe ich mich gut verstanden und mit den Projektleitern bin ich auch gut ausgekommen. Es waren fast alle studierte Menschen mit hohem Bildungsniveau, die man oft auf dieser Ebene in dem mitteldeutschen öffentlichen Dienst vorfindet. Sie haben verstanden, was ich für sie leisten wollten und haben es geschätzt. In den zwei Jahren, als ich dort tätig war, habe ich mehrere Seminare über alle Aspekte der Software Qualitätssicherung gehalten, mehrere Richtlinien für die Anforderungsdokumentation und den Test verfasst und die Qualität von mehreren Lastenheften gesichert. Es wurde immer streng nach V-Modell-XT gearbeitet [BrRa06].

Die zuständigen Projektleiter sind mit ihrem Lastenheft zu mir gekommen und nachdem ich sie strukturiert und markiert habe, lies ich das Dokument von dem Textverarbeitungswerkzeug analysieren. Damit wurden die logische Testfälle extrahiert, das Dokument in Bezug auf die Regel geprüft, die Anforderungen gemessen und den Aufwand für den Test geschätzt. Zum Schluss hat das Tool auch noch eine Prüfspezi-

fikation nach dem ANSI-829 Standard und der V-Modell-XT Norm erstellt [Sned14]. Natürlich müssten die Projektleiter die Prüfspezifikation nachbearbeiten, aber sie hatten ein Dokument mit dem sie anfangen konnten. Wie mit Programmen, ist es leichter ein bestehendes Dokument zu ergänzen und zu ändern als ein neues aus dem Nichts zu schaffen.

Das Lastenheft selbst habe ich kommentiert und mit den Projektleitern besprochen, die es dann nachgebessert haben. Erst als das Dokument einen Minimum Qualitätsstand erreicht hat, wurde es zur Ausschreibung freigegeben. Neben der Liste der gekennzeichneten funktionalen und nicht-funktionalen Anforderungen waren in dem Lastenheft auch die Anwendungsfälle spezifiziert. Hierfür wurde eine Schablone mit allen wesentlichen Attributen eines Anwendungsfalles – Auslöser, Vorbedingungen, Nachbedingungen, Ablaufpfade, Eingaben, Ausgaben und Beziehungen zu andere Anwendungsfällen – aus der Literatur von Cockburn übernommen und erweitert [Cock02]. Dabei hat mir eine Studentin der Verfahrenstechnik von der angrenzenden Universität geholfen. Zum Schluss des Lastenheftes folgten die Abnahmekriterien für das fertige System. Sie wurden als Anhang zum Lastenheft angeheftet.

Nachdem das Projekt vergeben wurde, sollte von dem Auftragnehmer ein Pflichtenheft zurückkommen. Ich wollte diese ebenfalls kontrollieren, hatte aber keinen Einfluss auf deren Gestaltung. Es gab keine verbindliche Norm für die Gestaltung der Pflichtenhefte, das V-Modell war in diesem Punkt recht ungenau, so dass die Auftragnehmer einen großen Spielraum hatten. Manche haben recht detaillierte Entwürfe abgegeben, andere nur grobe Absichtserklärungen. Im Gegensatz zu den Mitarbeitern der Behörde konnte ich ihnen nichts vorschreiben. Hier kommen wir zur Schwachstelle des gesamten Verfahrens. Nachdem die Behörde das Projekt vergeben hat, hatte sie keinen Einfluss mehr auf den weiteren Verlauf. Erst am Ende, als das fertige Produkt abgegeben wird, kommt der Auftraggeber wieder ins Spiel. Er kann nur noch das fertige System abnehmen, oder auch ablehnen. Zu diesem Zweck wurde ein aufwendiger Abnahmetest vorbereitet. Geprüft wurde auch die abgelieferte Dokumentation. Nicht selten haben die gelieferten Produkte sich als unzulänglich erwiesen. Schon in den ersten Tests hat sich gezeigt, dass das Produkt nicht den Erwartungen entspricht. Die Behörde konnte die Abnahme verweigern, was sie auch tat, aber dann hatte sie keine Lösung zu ihrem Problem. Sie musste warten auf die Nachbesserung und den ganzen Abnahmetest wiederholen. In einigen Fällen hat das gelieferte Produkt den Abnahmetest nie bestanden und das Projekt wurde mit Verlust abgebrochen. In den meisten Fällen wurden jedoch die Abnahmekriterien soweit gelockert, dass das System endlich abgenommen wurde. Als Ergebnis hatte die Behörde ein Produkt, das ihre Anforderungen nur teilweise erfüllte. Es folgen eine Reihe „Änderungsanträge“,

die sich auf die offenen Anforderungen bezogen und die mussten alle zusätzlich bezahlt werden [WiSn12].

Das Ganze lief auf ein Spiel hinaus, wobei der Auftraggeber ein unfertiges, unzulängliches Produkt zum vereinbarten Termin zum vereinbarten Preis ablieferte, um dann weitere Zahlungen zu erhalten, um das Produkt irgendwann fertigzustellen. Der Auftraggeber ist in einer Falle. Wenn er das Produkt schlichtweg ablehnt, muss er wieder von vorne anfangen und eine neue Ausschreibung machen. Um dies zu vermeiden, bleibt ihm nichts Anderes übrig als weiter zu zahlen und weiter zu warten, bis er das Produkt hat, was er haben wollte. Die Ursache des Problems liegt in dem Verfahren. Um den Auftrag zu erhalten muss der Auftragnehmer möglichst preiswert anbieten und auch den Wunschtermin des Auftraggebers akzeptieren, wohl wissend, dass er am Ende doch noch mehr Geld und Zeit bekommt. Als Folge entpuppt sich das V-Modell als Farce. Es tauscht nur eine heile. geordnete Welt vor, die es in der Wirklichkeit nicht geben kann. In meiner Zeit dort sind die Anforderungsdokumente zwar qualitativ besser geworden, aber sie waren immer noch nicht fachlich vollständig. Bis auf einige glückliche Ausnahmen haben die Verfasser immer etwas ausgelassen. Folge dessen war die abgelieferte Software unvollständig. Hinzu kam, dass die Auftraggeber die Projekte fast immer unterschätzt haben. Sie dauerten länger und kosteten mehr, als sie dafür angeboten hatten. Ihre Bezahlung hing jedoch von der Abgabe und Abnahme des Produktes ab. Um ihr Geld zu bekommen, müssten sie etwas abliefern in der Hoffnung, die Behörde wurde es doch noch abnehmen. Diese Hoffnung ging nicht immer auf. Die Abnahme wurde zunächst verweigert, aber da die Behörde zu dieser späten Stunde keine Alternative hatte, wurde es doch noch beim zweiten oder dritten Versuch abgenommen und dass Spiel mit den Änderungsanträgen begann.

Dies war sicherlich nicht im Sinne der Väter des V-Modells. Sie wollten eine heile, geordnete Entwicklungswelt schaffen, in der die Projekte der öffentlichen Hand „on Time“ und „in Budget“ abgewickelt werden. Der ausgeklügelte Entwicklungsprozess mit den vielen genormten Dokumenten als Zwischenergebnis sollte einen geordneten Projektablauf sichern, aber Softwareentwicklung lässt sich leider nicht ohne weiteres begradigen. Sie ist eher wie ein wilder Fluss, der seinen Weg zum Meer sucht. die Menschen in den Softwareprojekten sind keine perfekten Wesen und imperfekte Wesen können keine perfekten Produkte herstellen – trotz aller wohlgemeinten Vorschläge zur Prozessoptimierung. Für meine Begriffe wird der Prozess überwertet. Was zählt sind die Menschen und ihre Werkzeuge.

Trotz der Unzulänglichkeiten des V-Modell-XT wäre ich gerne in Ilmenau geblieben. Ich hatte sogar Pläne, dort eine lokale Testmannschaft aufzubauen, um die Abnahmetests durchzuführen und zu wiederholen. Ich wollte den Abnahmetest automatisieren, damit er kostengünstig wiederholt werden könnte. Aber leider wie so oft in meinem Leben kam es anders. Den Fortsetzungsvertrag hat meine Firma ANECON vermasselt. Das Folgeprojekt hatte einen wesentlich höheren Umfang an Personal. Statt um eine ging es um mindestens drei Fachkräfte. Der zuständige Account-Manager hatte versäumt, den Versicherungsbetrag zu erhöhen und so sind wir aus formalen Gründen aus dem Wettbewerb für das Folgeprojekt ausgeschieden. Die deutsche TÜV hat den Ausschlag erhalten. Ich müsste mich schweren Herzens von den Mitarbeitern der Behörde verabschieden und unverrichteter Dinge nach Wien zurückkehren.

10.6 Der Test eines Data Warehouse Systems

In Wien warteten weitere, weniger erfreuliche Testprojekte auf mich. Ein Projekt hatte ich schon begonnen, als ich noch in Ilmenau war. Es war der Test eines Data Warehouse Systems für eine österreichische Großbank. Die anderen Tester von der ANECON waren schon seit einigen Monaten im Einsatz dort, als ich dazu kam. Aus Sicherheitsgründen müssten wir in dem Gebäude der Bank arbeiten. Der einzige Raum war ein kleines Dachzimmer mit winzigen Fenstern oben, unter dem Dach eines alten Gebäudes im 2. Bezirk. Wir saßen dort zu fünft ohne Ventilation. Die Arbeitsbedingungen waren alles andere als angenehm, aber wir waren ja nur Tester, und Tester haben sich mit solchen Räumlichkeiten abfinden. Tester sind eben Menschen dritter Klasse in der IT-Hierarchie. Es war ein riesen Kontrast zu dem schönen Glasgebäude und den großen klimatisierten Raumen in Ilmenau mit Ausblick auf den Thüringer Wald. Ich saß in jenem Dachzimmer an einen Katzentisch in der Ecke eingequetscht zwischen den jungen Kollegen.

Generiertes Testskript

```
file: ACCOUNT;
// This comparison procedure assumes that the old file contains the following attributes:
//     from TAB_X Table: A_ID, A_TYPE, ATTR_S, ATTR_R, ATTR_C, ATTR_S, ATTR_D, ATTR_F,
//     from TAB_Y Table: ATTR_C, ATTR_D, ATTR_E, ATTR_F, ATTR_P
//     from TAB_Z Table: ATTR_R
 if ( new.A_ID = old.ATTR_ID );
      assert new.A_ID = old.A_TYPE if (old.A_TYPE = "G");
      assert new.A_ID = "S" if (old.ATTR_A = "R" & old.ATTR_S = "S");
      assert new.A_ID = "C" if (old.ATTR_A = "R" & old.ATTR_S = "C");
      assert new.A_TYPE = "L" if (old.ATTR_A = "R" & old.ATTR_S = "L");
      assert new.A_RATE = old.ATTR_C if (old.ATTR_B = "0");
      assert new.A_INTEREST = old.ATTR_D;
      assert new.A_ACCT = old.ATTR_P;
      assert new.START_DATE = "2005.05.15" if (old.ATTR_H <> "0");
      assert new.REVIEW_DATE = old.ATTR_V if (old.ATTR_F <> "0" & old.ATTR_N <> "0") ;
      assert new.REVIEW_DATE = "NULL" if (old.ATTR_F = "0") ;
      assert new.PRIMARY_INDICATOR = "P" ! "S" ! "*n.a.*";
      assert new.INDICATOR = "inactiv" if (old.ATTR_R = X"3") ;
      assert new.INDICATOR = "inactiv" if (old.ATTR_R = X"1") ;
      assert new.INDICATOR = "closed" if (old.ATTR_R = "C") ;
      assert new.INDICATOR = "active" if (other);
end;
```

Meine Aufgabe in diesem Projekt war es, mit meinem Datenvalidationswerkzeug „DataTest“ die Eingangsdaten aus den osteuropäischen Tochterbanken zu verifizieren und mit dem Testspezifikationswerkzeug „TestSpec“ die Testfälle zu generieren. Das Datenwarenhaus war eine Sammlung von 265 SQL Tabellen mit über 11000 einzelnen Datenelementen. Eine Mitarbeiterin der Bank aus Czechen hatte schon eine riesend Regeltabelle in Excel mit einer Regel für fast alle belegten Datenelemente erstellt. Leider ist es ihr nie gelungen, Regel für alle Daten zu schreiben aber immerhin für die meisten Ergebnisattribute. Von den 5300 Ergebnisdaten hatten 4700 eine Regel. Es gab 7 verschiedene Regeltypen:

- eine eins zu eins Zuweisung einer variabel bis einer anderen Tabelle
- eine eins zu eins Zuweisung einer Konstantenwert
- eine Auswahl aus einer Liste alternativer Konstantenwerte bzw. eine Enumeration
- eine Vereinigung mehrerer Quellattribute aus einer einzigen anderen Tabelle bzw. eine Concatenation

- eine Vereinigung mehrere Quellattribute aus mehreren andere Tabellen bzw. eine Join
- eine Zuweisung des Zielattributs aufgrund einer Rechenoperation mit konstanten Werten und variablen Attributen aus der gleichen Tabelle
- eine Zuweisung des Zielattributes als Ergebnis einer Standardfunktion.

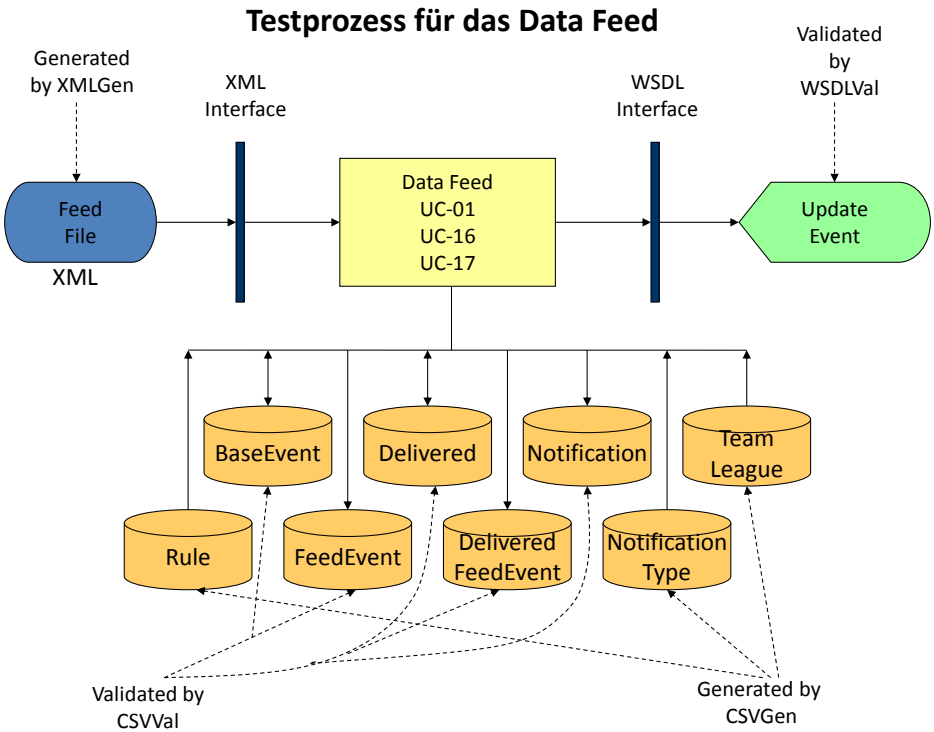
Diese Regel waren in einer semiformalen englischer Sprache mit booleschen Logik formuliert. Mein Testvalidationswerkzeug verarbeitet Assertionen in einem strengen formalen Format. Um die 11000 Daten automatisch zu prüfen musste ich also 4700 solche informale Regelbeschreibungen in formale Assertionen umsetzen. das gehörte zu den schwierigsten Aufgaben in meiner Karriere als Software Tester. Ich habe mein Tool TestSpec erweitert um die informale englische Regel in formale Assertion-Regel zu versetzen. Es ist gelungen, 4100 Regeln voll zu übersetzen. Die verbleibenden 600 Regeln mussten per Hand umgeschrieben werden, was circa 20 Minuten pro Regel gekostet hat. Ich habe also 266 Arbeitsstunden mit der Übersetzung der Regel verbracht. Ohne das Tool hätte es den vierfachen Aufwand gekostet und das wäre nicht zu rechtfertigen gewesen. Für den Test gab es nur ein geringes Budget [Sned06].

Auf dieser Weise konnte ich 266 Assertionsskripte mit 4897 einzelnen Assertionen produzieren. Die Assertionsskripte wurden von dem Werkzeug DataTest gelesen und gepaart mit den Eingabedateien aus den Osteuropäischen Ländern. Leider stellte es sich durch den Abgleich der Eingabedateien mit den Datenregelt heraus, dass die Eingabedaten weitgehend ungültig waren. Mehr als 20% der Datenwerte entsprachen nicht der spezifizierten Regel. Die viele Abweichungen wurden von dem DataTest Tool protokolliert und das Protokoll ging an das Projektmanagement. Dieses musste zur Kenntnis nehmen, dass die Qualität der gelieferten Daten so niedrig war, dass es sich nicht lohnte, sie dort in Wien zu bereinigen. Man musste sie schon an der Quelle prüfen und korrigieren. Demzufolge wurde das Projekt abgebrochen. Leute wurden gesucht, um zu den osteuropäischen Zweigstellen zu gehen und dort vor Ort die Daten zu bereinigen ehe sie nach Wien an die Zentrale weitergeleitet werden. Ich hatte noch das Projekt in Ilmenau und wollte nicht nach Moskau gehen. Vielleicht habe ich eine einmalige Gelegenheit verpasst.

10.7 Der Test eines Internet-Wettsystems

Als ich im Sommer 2007 von Ilmenau nach Wien zurückkehren musste wurde ich von meinem Abteilungsleiter an ein neues Testprojekt in Wien gleich weitervermittelt. Ich sollte ein Testkonzept mit Testplan, Testentwurf und Testspezifikation für ein agiles Entwicklungsprojekt ausarbeiten. Das war ein abrupter Übergang von der star-

ren geordneten Welt des V-Modells zur chaotischen, flexiblen Welt der agilen Entwicklung. Dort wurde ich in den Ritualen der agilen Entwicklung eingeweiht. Ich war mittendrin unter jungen Menschen mit anderer Kleidung, anderer Gepflogenheiten und vor allem mit einer anderen Sprache. Es war weit entfernt von dem strengen Wasserfallmodell der deutschen Behörden. Jeden Tag wurden die Projektaufgaben neu definiert. Ich musste jeden Morgen um 10 Uhr in einem Kreis stehen, für das tägliche Standup-Meeting. Mir wurde zum ersten Mal bewusst wie alt ich bin. Ich war der einzige im Kreis ohne Turnschuhe. Ich ging gleich hinaus und besorgte mir ein Paar Turnschuhe. Das wichtigste in einem agilen Projekt ist sich den anderen anzupassen.



Der erste Schritt war für mich einen Testplan zu erstellen. In der agilen Literatur zum Thema Test spricht man von dem 24-Stundigen Testplan [CrGr09]. Zum Glück gab es schon ein ausführliches Anforderungsdokument mit 95 funktionale, 5 nichtfunktionale und 12 Datenanforderungen erstellt. Die einzelnen Anforderungen waren auch schon gekennzeichnet. Ich musste an dem englischsprachigen Dokument kaum was ändern. Inzwischen hatte ich die englischsprachige Version meines Testanalyse

Tools schon verwendet, um große englischsprachige Anforderungsdokumente zu analysieren – u.a. für das T-Mobile Handy Software und für das Londoner Metro. Auf dieses Word-Dokument lief alles auf Anhieb und produzierte alles was ich brauchte – einen Testplan, eine Testaufwandsspezifikation und 667 logische Testfälle, bzw. Testbedingungen. Dieser erste Schritt dauerte nicht mehr als eine Woche. Ich war damit voll im Trend, agiler als agil dank der Testplanungsautomation.

Der nächste Schritt dauerte etwas länger. Die Erstellung des Testentwurfs war noch nicht automatisiert. Ich musste ihn manuell als XMI Dokument erstellen. Es dauerte eine Weile, bis ich die Gliederung des Entwurfs konzipiert habe. Ich musste für jeden Testlauf ein Szenario mit mehreren Testschritten – jeden mit einer Untermenge der logischen Testfällen – definieren. Da ich XML als Dokumentationssprache gewählt hatte, entpuppte sich dieser Schritt als Übung in der Erstellung eines XMI Schemas. Ich musste so lange daran basteln, bis der XML Syntax-Checker von Microsoft es akzeptierte und dies war nur die Syntax. Die Semantik des Inhaltes musste auch stimmen im Vergleich zu dem Testplan und im Bezug zu den bereits erstellten logischen Testfällen. Später habe ich auch diesen Schritt mit TestSpec automatisiert, aber wie allzu oft war ich für dieses Projekt zu spät daran [Sned02].

Der nächste Schritt, die Umsetzung der logischen Testfälle in physikalische Testfälle mit stellvertretendem Datenwerte sollte noch länger dauern. Da dieses System ein bestehendes Datenaufnahmeprojekt ablösen sollte, konnte ich die alten Eingangsnachrichten als Ausgangsbasis wiederverwenden. Die Eingaben waren Fußballspielergebnisse – die übliche Spielstatistik – die Anzahl Tore, die Anzahl Schüsse aufs Tor, die Anzahl Eckbälle, die Dauer der Ballkontrolle, die Anzahl roter und gelber Karten, usw. Ich habe meinen älteren Sohn, Stephan, der zu dieser Zeit studiert hat, damit beauftragt aus den alten Nachrichten Testdaten für das neue System zu erstellen. Neben den alten Nachrichten gab ich ihn die Excel-Tabellen mit den logischen Testfällen und er gab sie mir zurück mit repräsentativen Testwerten. Nach einem Monat hatten wir die Testfälle für 12 Anwendungsfälle.

Der letzte Schritt in meiner Arbeit in diesem agilen Projekt war die Generierung der Testdaten, obwohl das eigentlich über meinen eigentlichen Auftrag hinausging. Die Daten waren Email-Nachrichten mit XML Dokumente als Anhänge. In den XML Dokumenten war die Spielstatistik enthalten. Es ging also darum, aus den XML Nachrichten die SQL Datenbanken zu generieren. Als Testskriptsprache benutzte ich meine bisherigen Assertionen. Mit der Generierung der Datenbanken hatte ich keine Probleme, die Datenbankstrukturen waren flach, aber mit den verschachtelten Strukturen der XML Dateien hatte ich Probleme. Hinzu kam, dass die Namen der Daten

nicht die Tags sondern die Attribute waren. Ich musste daher sehr viel Zeit damit verbringen, den XML Generator zu tunen. Dann war meine Zeit zu Ende und ich musste abziehen, gerade als ich kurz vor dem Durchbruch stand. Laut dem Vertrag mit dem Anwender sollte als Testanalytiker den Test nur planen und entwerfen. Ich war scheinbar zu teuer um den Test selbst durchzuführen. Also wurde ich wurde von anderen ANECON Tester abgelöst, die mit Testautomation nicht viel im Sinne hatten. Sie versuchten den Test weiter manuell durchzuführen und brauchten dafür mehr Zeit als vorgesehen war. Am Ende wurden sie durch eine slowakische Testfirma abgelöst, die wesentlich billiger war. Für den manuellen Test sind Tester aus den Hochlohnländern eben zu teuer. Das Preis/Leistungsverhältnis stimmt nicht. Für Hochlohn-Testfirmen gibt es demzufolge keine Alternative zur Testautomation [Sned08].

10.8 Testplanung für die Wiener Krankenhäuser

Meine letzten Testprojekte für die ANECON waren eher nur Testplanungsprojekte. Als solche waren sie nur von kurzer Dauer, höchstens zwei Monate. Es ging hauptsächlich darum, einen Testplan zu verfassen, den Testaufwand zu schätzen und das Testprojekt zu planen. Das erste Projekt dieser Art war für die Wiener Krankenhäuser. Sie waren dabei, ein neues Krankenhaus-Informationssystem auszuschreiben und wollten dazu ein separates Testprojekt anlegen. Ich sollte für sie einen Testprozess vorschlagen und ein Testkonzept ausarbeiten. Das ist in einer solchen Umwelt keine leichte Aufgabe, denn es ging nicht um ein Projekt, sondern um mehrere. Die Stadt Wien hat abgesehen von dem Allgemeinen Krankenhaus AKH. 13 verschiedene Institutionen – Krankenanstalten, sozialmedizinische Zentren und Pflegeheime. Sie sollten alle von einem einzigen Informationssystem bedient werden. Das Informationssystem – KIS – sollte Bestandteil eines allumfassenden Krankenhaus-Verwaltungssystem sein. Wie immer bei derartigen großen staatlichen Verwaltungsprojekten war alles höchst politisch. Es sollten einheitliche Prozesse mit einer einzigen gemeinsamen Datenbasis geben, dennoch sollten die beteiligten Institutionen einen möglichst großen Gestaltungsraum haben.

Um das Ausschreibungsdokument zusammenzustellen wurde jede beteiligte Instanz aufgefordert, ihre Anforderungen zu verfassen und an die zentrale Projektleitung einzureichen. Auf dieser Weise kam eine Reihe sehr unterschiedlicher Dokumente zusammen, manche recht gut strukturiert mit sogar gekennzeichneten Anforderungen, andere mit wirren Prosatext scheinbar vom Oberarzt geschrieben., Am Ende sollte aus dieser bunten Dokumentensammlung eine konsistente Ausschreibung bzw. „Call for Tender“ herauskommen. Wie bei den deutschen Behörden, ist es mir hier bei diesem Projekt eingeleuchtet, wie schwer es ist eine brauchbare Beschreibung

eines komplexen IT-Systems auszuarbeiten, mit genügend Inhalt, dass ein Fremder in der Lage sein sollte daraus ein Angebot abzuleiten geschweige denn ein funktionierendes IT-System. In dieser Hinsicht gebe ich den Gründern der agilen Entwicklung Recht. Man kann sich die ganze Vorausspezifikation derart umfangreicher Softwaresysteme ersparen. Man soll lieber mit einem Teilsystem anfangen und schauen wie es sich entwickelt.

```
<Requirement id = "FR-KIS-13">
  <RequirementHeader>
   <RequType>Functional</RequType>
   <RequPriority>hoch</RequPriority>
   <RequOwner>Ott</RequOwner>
   <RequStatus>defined</RequStatus>
   <RequLabel>Patientendatenverfolgung</RequLabel>
  </RequirementHeader>
  <RequirementBody>
   <RequTextLine>Jeder Patient muss im ADM-System administriert werden. Dabei wird eine
    Aufenthaltszahl</RequTextLine>
   <RequTextLine>vergeben. Die Administration kann fuer stationaere Patienten zentral oder dezentral
    erfolgen.</RequTextLine>
   <RequTextLine>ambulante Patienten erfolgt sie in der jeweiligen Ambulanz.</RequTextLine>
  </RequirementBody>
 </Requirement>
 <Requirement id = "FR-KIS-14">
  <RequirementHeader>
   <RequType>Functional</RequType>
   <RequPriority>mitt</RequPriority>
   <RequOwner>Ott</RequOwner>
   <RequStatus>defined</RequStatus>
   <RequLabel>Patientenidentifikation</RequLabel>
  </RequirementHeader>
  <RequirementBody>
   <RequTextLine>Bei stationaer aufgenommenen Patienten ist die Aufnahmezahl gueltig bis zum Ende
    des stationaeren Aufenthalts</RequTextLine>
```

Ableitung der Anforderungen aus dem prosa Anforderungstext

Anderseits wollen die Geldgeber, in diesem Fall die Stadt Wien, wissen was das Ganze kosten soll, ehe sie sich auf ein solch teures Unterfangen einlassen. Außerdem sind sie ihren Geldgebern verpflichtet, ein Budget aufzustellen und sich daran zu halten. Sie sind auch verpflichtet zu voraussagen, wann das Ganze fertig wird. Sie sind also in einer Zwangslage. Sie müssen Aussagen treffen aufgrund von Information, die sie gar nicht gewinnen können. Der schwarze Peter wird an die potentiellen Anbieter weitergegeben. Sie sollten aus dieser Wust an Texten, Tabellen und Diagrammen einen festen Preis und einen endgültigen Termin abgeben – ein Ding der Unmöglichkeit. Es hätte nur einen Sinn, wenn in den Dokumenten wirklich alle Systementitäten identifiziert sind- die Geschäftsprozesse, Geschäftsobjekte, Geschäftsregel, Akteure,

Anwendungsfälle usw. Danach hätte der Anbieter vielleicht eine Chance ein begründetes Angebot abzugeben. Aber aus den grobstrukturierten, inkonsistenten und unvollständigen Fließtexten hätte er keine Chance [Broy07].

Ich habe mein Bestes getan, den chaotischen Dokumenten eine Struktur zu verleihen und die Systementitäten zu identifizieren. Bei dem Einen oder Anderen Dokument ist es mir auch gelingen. Ich habe die Anforderungen markiert und klassifiziert, die Systemschnittstellen eingezeichnet und einzelne Objekte und Regel identifiziert und als solche zu erkennen gegeben. Bis zu den Anwendungsfällen bin ich nie gekommen. Die Dokumente blieben auf einem groben, allgemeingültigen Niveau. Dennoch konnte ich sie mit dem Textanalysewerkzeug bearbeiten und sogar Testfälle ableiten. Ich wusste jedoch sehr wohl, dass dies nicht alles war. Die Dokumente stellten nur die Spitze eines Eisberges dar. Unsichtbar blieb, was darunter lag. Hier stellte de sich die Frage, ob eine unzulängliche Beschreibung besser ist als gar keine. Das Tool berichtete, dass die Dokumente unvollständig und weit von dem Sollqualität entfernt war, aber was bring das anzumahnen? Ich glaube kaum, dass die Analytiker in den Krankenhäuser und Kliniken sie noch überarbeiten würden. Der Termin für die Ausschreibung stand unmittelbar davor. Als Projektplaner muss einer mit der Information zurechtkommen, was man hat. Immerhin konnte ich eine Anforderungsdatenbank aufbauen mit mehr als 2000 Anforderungen und 5000 Testfällen aus den einzelnen Dokumenten [Sned09].

Das Ziel des Analyseprojektes war es, neben einen Ausschreibungsdokument eine Aufwandsschätzung und ein Service Level Agreement zu liefern. Mit der Aufwandsschätzung habe ich mich schwer getan. Ich musste die Anzahl Function-Points und Data-Points aus der Dokumentenanalyse nach oben korrigieren, wusste aber nicht um wieviel - um das Zweifache, dreifache oder gar das Vierfache? Der Auftraggeber wollte meine Schätzung mit dem Angebote der Anbieter vergleichen. Ich hatte den Nachteil, dass ich nicht wusste, welche Produktivität die Anbieter als Basis ihrer Schätzungen heranziehen. Ich konnte nur irgendeinen Industriedurchschnitt verwenden. Es dürfe also nicht wundern, dass meine Schätzung um einiges unter den Schätzungen der Anbieter lag.

Es war nicht das erste Mal, dass ich eine Risikoanalyse für ein Großprojekt durchführte. Ich hatte schon die Risikoanalyse für das FISCUS Projekt in Deutschland gemacht. Aber mit einem Eskalationsplan für den Fall, dass die Risiken eintreten, hatte ich noch nie zu tun gehabt – dies war für mich neu. Ich habe mich in die einschlägige Literatur eingearbeitet und vier Eskalationsstufen definiert:

- auf der Stufe der Projektleiter,

- auf der Stufe der Bereichsleiter,
- auf der Stufe der Produktmanager und
- auf der Stufe der Unternehmensleitung.

Außerdem definierte ich mehrere Eskalationsszenarien, darunter

- fehlende Funktionalitäten,
- unzulängliche Qualität,
- Terminüberschreitung und
- Kostenüberschreitung.

Für jeden dieser Fälle schrieb ich einen Vermittlungsprozess vor. Meine Erfahrung als Gerichtsgutachter für T-Systems kann mir hier zu gute. Ich kannte sehr wohl die Argumente beider Seiten – Auftraggeber und Auftragnehmer – und konnte die Schritte zur Schlichtung des Konfliktes aufzeichnen. Es ist immer wieder das Gleiche. Am Ende kommt es zur Abfindung mit der Ist-Situation, die Funktionalität wird gestrichen, die Qualitätsansprüche werden heruntergeschraubt, es wird mehr Geld nachgeschoben und der Termin wird verschoben. Wichtig ist, dass das Projekt weitergeht.

Der Eskalationsplan floss in die Service-Level Vereinbarung ein. Diese regelte den Betrieb des Informationssystems. Das 42-Seitige Dokument schrieb die zuständigen und Rollen vor, wie z.B. Help Desk, Konfiguration Kontrolle, Service Kontroller und Liaisonsbeauftragter – legte Grundsätze für die Zusammenarbeit fest, regelte die Verantwortlichkeiten der beteiligten Parteien und spezifizierte die anfallenden Berichte. Der Eskalationsprozess wurde eingebaut und sämtliche nicht-funktionale Anforderungen angehängt. Als Ergänzung dazu definierte ich 24 Operationsmaße, 20 Servicemaße, 12 Kostenmaße und 17 Zufriedenheitsmaße. Es sollte möglich sein den Grad der Kundenzufriedenheit möglichst genau zu messen. Eine weiterführende GAP Analyse war auch vorgesehen [Sned15].

Obwohl ich nicht genau schätzen konnte, was das ganze Verbundprojekt kosten würde, hatte die Projektleitung des Krankenhausbundes keinen Anlass mit meinem Beitrag unzufrieden zu sein. Ich hatte getan, was ich tun konnte unter den Umständen. Das Frustrierende in der Testberatung ist das meistens die Voraussetzungen fehlen um eine fundierte Arbeit zu leisten, gerade in großen, politischen Projekten im öffentlichen Dienst. Das hatte ich schon in dem FISCUS Projekt in Deutschland erfahren. Man kann nur versuchen das Beste daraus zu machen.

10.9 Meine letzten Testplanungsprojekte

Dass mein Bestes nicht immer genug ist, zeigten die letzten Testprojekte für die ANECON. Das Eine war für eine weltberühmte Getränkefirma. Nach dem Planungsprojekt für den Krankenhausverbund kam ich dorthin, um für sie ein Testkonzept auszuarbeiten. Sie war ein IBM AS-400 Kunde und hatte die Organisation und die Mentalität eines mittelständischen Betriebes, obwohl sie Teil eines weltweiten Konzerns war. Als ich vorgeschlagen habe, ihre Testfälle zumindest semiformal zu spezifizieren und eine Testfall-Datenbank aufzubauen, haben sie mich als weltfremder Spinner abgezeichnet. Sie konnten sich mit meiner „akademischen Vorgehensweise“ überhaupt nicht anfreunden. Ich musste an meinem letzten AS400 Kunde in Wienerneudorf denken – eine Firma in der Damenbekleidungsindustrie. Ich wollte dort ein zweitägiges Seminar über Integrationstest halten. Schon am zweiten Tag haben sie mich herausbegleitet. Der Kurs war für sie viel zu praxisfern, zumindest was sie unter Praxis verstanden haben. Die Lehre war auf einer allgemeinen methodischen Ebene und ging nicht ausreichend auf die Eigenarten der AS400 ein. Jeder Betrieb hat eine eigene selbst erfundene Praxis, die in der Regel von dem Chefentwickler dort diktiert wird. Jeder, der diese Praxis in Frage stellt, ist ein unerwünschter Gast. Es wäre nicht so schlimm gewesen, nur als sie mich verabschiedeten, kam ein Wolkenbruch auf und ich musste in strömenden Regen an der Landstraße nach Wien zurückradeln. Ich fühlte mich wie ein verdrängter Hund.

Diese Erfahrungen mit mittelständischen Betrieben gab mir zu denken. Das Wissen was ich hatte konnte ich nicht überall einbringen. Es kann sein, das es überhaupt nicht zu dem Kunden passt. Dann kann es eher schädlich als nützlich sein. Wissen muss man dosieren, je nachdem wer es empfangen soll.

Kurz danach schickte mich ANECON nach Stuttgart zu einer Zahlkartenfirma, auch ein mittelständisches Unternehmen. Dieses hatte eine Ausschreibung gewonnen und war an dem deutschen Gesundheitskarteprojekt beteiligt. Ich sollte für sie die Testkosten schätzen. Ich hatte mittlerweile einige Erfahrung damit [SnJu10]. Der zuständige Manager war ein junger Betriebswirt. Ich sollte für ihn ausrechnen, was der Test der Apothekenanateil der Gesundheitskarte kosten würde. Ich machte mich daran, die Anforderungsdokumente mit meinen Werkzeugen zu analysieren und binnen weniger Tage kam ich an und lieferte einen Bericht, wonach er circa 12000 Testfälle brauchen würde und das bedeutet mindestens 1500 Testertage. Sein Budget war allerdings viel kleiner und so passten ihm diese Zahlen überhaupt nicht. Meine Schätzung hatte für Unruhe im Projekt gesorgt. Er bat mich den Betrieb sofort zu verlassen und mich dort nie wieder blicken lassen.

Aus der Zeitung habe ich später erfahren, dass die erste Einführung der Gesundheitskarte wegen Qualitätsprobleme abgebrochen werden musste. Es hieß, sie brauchten viel mehr Zeit zu testen. Dies war für mich ein gewisser Trost, aber meine Zeit als Tester für die ANECON war vorbei. Mein Chef traute sich nicht mehr mich zu einer Kunde zu schicken. Ich war zu alt und zu akademisch. Meine Karriere als Tester neigte sich dem Ende zu. Wir mussten ein anderes Tätigkeitsfeld für mich finden. Schon damals wäre es ihm bestimmt recht gewesen, hätte ich mit meinen 69 Jahren gekündigt. Ich rechne es ihm hoch an, dass er mich nicht gleich weggeschickt hat. Das sollte erst fünf Jahre später erfolgen.

11 Vom Tester zum Vermesser

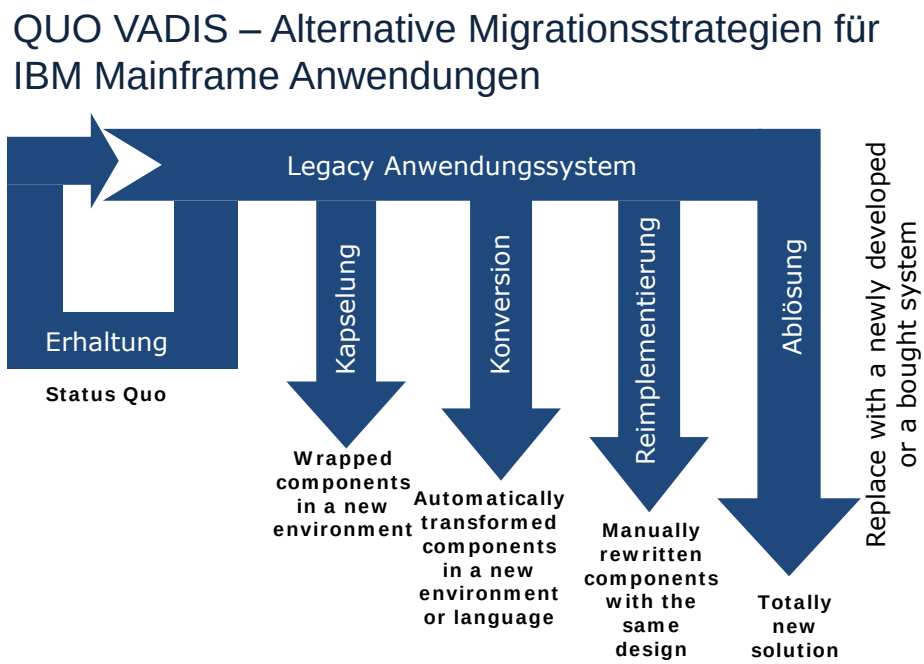
Bei der ANECON habe ich mich nach 2007 vom Testen zum Vermesser verwandelt. Ilmenau war mein letztes Testprojekt. Als aus meiner Idee dort eine Testkompetenzstelle mit jungen Testern aus der Umgebung aufzubauen nichts geworden ist, war meine Karriere als Software-Tester beendet. In den anderen Testprojekten habe ich nicht mehr hineingepasst, zumindest nicht in die der ANECON. Einerseits war ich schon zu alt, anderseits waren meine Vorstellungen zum Test inzwischen zu weit weg von den Vorstellungen der Durchschnittsanwender. Ich war darauf fixiert jede Testaktivität von der Planung bis zu Bewertung zu automatisieren. Diese Absicht war nicht verkehrt, sie war eben zu früh. Die Stunde der Testautomation hatte noch nicht geschlagen und für das herkömmliche Testen war ich schon zu alt und zu teuer. Ich musste ein neues Umfeld suchen. Das sollte die Softwarevermessung sein.

11.1 Das größte Messprojekt aller Zeit

Es hatte sich aufgrund meiner GI-Vorträge und Veröffentlichungen in Deutschland herumgesprochen, dass ich über Werkzeuge verfüge, um die Größe und Komplexität von Software Systemen anhand des Source-Codes zu vermessen. Mit diesen Messwerkzeugen konnte ich nicht nur Zeilen und Anweisungen, sondern auch Function-Points und Object-Points zählen. Diese Werkzeuge hatte ich schon lange gehabt, die ersten für Assembler, COBOL, PL/I und Natural Code hatte ich bereits in der 90er Jahren entwickelt, um den Aufwand für meine damaligen Migrationsprojekte zu schätzen. Außerdem hatte ich schon bei der SES ausgesprochene Messprojekte gehabt, z.B. für die niederländische Post, die Schweizer Telekom und die Berliner Bank. Bei der niederländischen Post ging es darum alternative Standardanwendungssysteme mit einander zu vergleichen. Ich müsste 1995 nach Gröhningen reisen um dort drei große Billing-Systeme zu messen und zu bewerten. Bei der Schweizer Telekom ging es um die Qualität des bestehenden Abrechnungssystems, ob es sich lohnen würde es abzulösen. In der Schweiz habe ich die Softwareproduktivität untersucht und verglichen zwischen einer Bank, einer Versicherung und der Bundesbahn [Sned97]. Bei der Berliner Bank ging es um die Kosten einer Migration. Ein solches Messprojekt dauerte zwischen zwei und vier Wochen je nachdem wie umfangreich die Codemasse war und endete mit einem ausführlichen Bericht über die Größe, Komplexität und Qualität des betreffenden Systems. Später im GEOS Projekt begann ich mich mit den Sprachen C, C++ und Java zu befassen. Also hatte ich die Messwerkzeuge schon lange ehe ich zu der ANECON gekommen bin, aber niemand dort hatte danach gefragt.

Lediglich in einem Migrationsprojekt für die Wiener Stadtwerke habe ich sie eingesetzt um den Aufwand für die Konversion von PL/I und VisualAge Programme zu schätzen [Sned05]. Erst als ein großes Versicherungskonzern aus Deutschland im Jahre 2008 auf mich zukam um ihre Softwaresysteme zu messen, wurde ANECON darauf aufmerksam. Das besagte Konzern war ein Konglomerat aus verschiedenen Versicherungsfirmen die gerade gemergt wurden. Jede dieser Firmen brachte ihre Altsoftware mit und es war Aufgabe des neuen Konzerns die Software zu warten und weiterzuentwickeln. Das Angebot für das erste ANECON Messprojekt lautete: „Angebot für Softwaremessung zur Ermittlung der Komplexität und Qualität der bestehenden Software-Systeme und Abschätzung des erforderlichen Aufwands für ihre Evolution und Wiederverwendung."

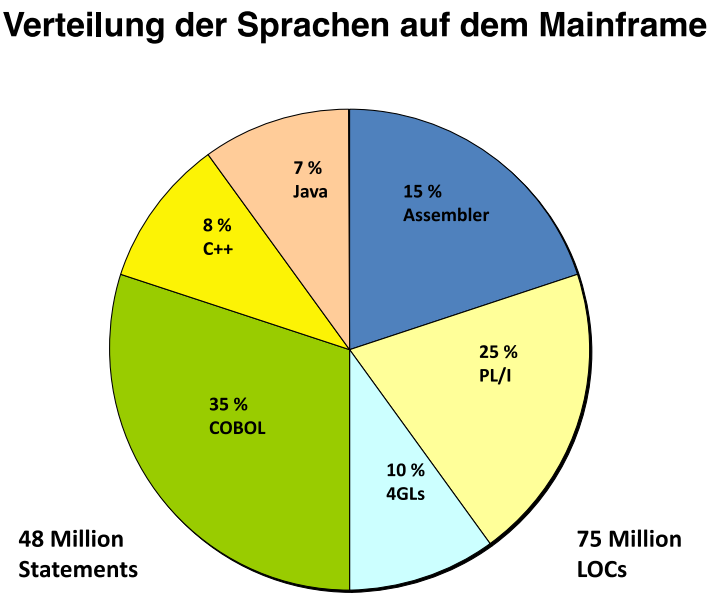
Das Ganze soll innerhalb drei Monate bemessen und ausgewertet sein.



11.1.1 Die Vielfalt der Programmiersprachen

Was ich aber zu Beginn nicht wusste war, dass es sich um mehr als 70 Millionen Codezeilen in sieben verschiedenen Sprachen handelte. Hätte ich das gewusst wäre

ich mit dem Angebot vorsichtiger gewesen. Neben den klassischen IBM Sprachen Assembler, PL/I, COBOL und CSP, gab es auch C++, Java und Easytrieve. Um das Projekt durchführen zu können, musste ich zunächst die alten Werkzeuge auf Vordermann bringen und durch neue Tools für CSP, bzw. VisualAge und Easytrieve ergänzen. Das hieß, ich musste zunächst das Messwerkzeug vorbereiten. Vor allem musste ich es um die zusätzlichen Sprachen erweitern. Da ich dies auf Kosten des Projektes nicht machen durfte, habe ich es auf eigener Rechnung neben der Voruntersuchung erledigt. Dafür hat mein Kollege – Richard Seidl – mich bei der Voruntersuchung tatkräftig unterstützt. Er half bei der Kalibrierung der Codier-Richtlinien, der Gewichtung der Metrik und der Identifizierung der I-O Funktionen für die Function-Point Zählung.



11.1.2 Fixierung auf Function-Points

Dieser Kunde war besonders daran interessiert, die Function-Points nach der konventionellen IFPUG Methode zu zählen [Albr83]. Der zuständige Mitarbeiter seitens des Kunden glaubte sehr an dieser Methode und war in der GI-Fachgruppe für Function-

Point Zählung aktiv. Es ist wahrlich eine Glaubenssache, denn Function-Points sind schwer zu definieren und zu erkennen. Fast jeder, der von dieser Methode überzeugt ist, weiß selber nicht genau, wie die Function-Points gezählt und gewichtet werden, und ein jeder, der sie tatsächlich zählt, tut dies auf einer anderen Art und Weise. Ich hatte dies schon Mitte der 80er Jahren erkannt als der Autor des ersten deutschsprachigen Buches zu diesem Thema – Thomas Noth - für mich als Werkstudent gearbeitet hat. Er hat über Function-Points promoviert und sammelte Daten bei verschiedenen Anwenderbetrieben [Noth84]. Je mehr Einblick ich in die Sache bekam, desto mehr habe ich daran gezweifelt. Ich kam mir vor wie der Junge in dem Märchen von dem Kaiser und sein neues Kleid. Schon Ende der 80er Jahren hatte ich mich von der Methode distanziert und nach einer anderen, solideren Methode gesucht. Es folgte bei mir die Data-Point Methode [Sned90] und später die Object-Point Methode [Sned96]. Bei dieser beiden Methoden konnte ich die Zählung wiederholen und zum gleichen Ergebnis kommen – bei der Function-Point Methode war dies der Fall. Function-Points musste man an Hand der Anforderungsspezifikation zählen und dies ging nur manuell. Es gab dabei einen großen Interpretationsbedarf. Erst im Jahre 2014 ist eine OMG Norm erschienen für die einheitliche Zählung von Function-Points anhand des Source Codes [OMG14]. Auch diese Norm ist interpretationsdürftig.

In der internationalen Metrik-Gemeinde, zu der ich auch gehörte, bin ich früher als entschieden Gegner der Function-Point Methode aufgetreten. Dennoch war ich gezwungen, die Methode anzuwenden, um überhaupt an Messprojekte heranzukommen, so wie in diesem Fall hier. Also beugte ich mich dem Druck des Marktes und versuchte das Beste daraus zu machen. Gerade in der IT-Welt, wo alles so komplex und unübersichtlich ist, neigen sich die Menschen wie primitive Urwaldvölker an irgendetwas zu glauben, das keine rationale Begründung hat. Jeder sucht nach einem Licht in der Dunkelheit. So stark ist der Wunsch der IT-Verantwortlichen ihre Projektkosten im Voraus zu erkennen, dass sie bereit sind an allem zu glauben. Der Wunsch ist der Vater der Gedanken.

Die System-ein und –ausgaben, um die es bei der Function-Point Zählung geht, sind in den alten prozeduralen Sprachen wie COBOL und PL/I sowie in den 4GL Sprachen wie Natural und ABAP anhand der IO, DB und DC-Anweisungen halbwegs erkennbar, nicht jedoch in den neueren objektorientierten Sprachen wie C++, C# und Java. Dort gibt es nur wenig Standard-IO Anweisungen, dafür aber viele Ein- und Ausgabefunktionen die vom Anwender selbst definiert werden. Im ersten Fall sind sie am Namen zu erkennen, im zweiten Fall weiß nur der Benutzer, was sie bedeuten. Demzufolge ist die Voraussetzung für eine Messung vom objekt-orientierten Code die Erfassung sämtlicher IO-Methoden und deren Unterscheidung nach Eingaben und

Ausgaben. In diesem Projekt hat dies fast ein Monat gedauert. Es war notwendig den gesamten Quellcode zu scannen und alle Methodennamen herauszufischen, um anschließend von den zuständigen Entwicklern herauszubekommen, was sie eigentlich bedeuten. Das gleiche Problem hatte ich mit den Makros in der Assemblersprache. Sie waren halbwegs dokumentiert, sodass mein Koautor in einem Monat eine Makrotabelle für die Assembler Programme aufstellen konnte, in der die Ein- und Ausgaben sowie die Datenbankzugriffsfunktionen gekennzeichnet waren. Für C++ und Java hat es länger gedauert weil man den Code der Funktionen selbst anschauen musste.

11.1.3 Umstrittene Programmierregel

Gleichzeitig habe ich mich mit den Gruppenleitern in der Softwareentwicklung über Programmierregel auseinandergesetzt. Mein Messwerkzeug zählt alle Programmierregel-verletzungen und gewichtet sie nach Schwere. Das geht zurück auf das ursprüngliche Analysewerkzeuge - SoftDoc, das ich in den 80er Jahren von dem SZAMOK Institut habe entwickeln lassen. Damals wollte ich die Qualität des Codes messen und dazu braucht man Regeln. Für jede Programmiersprache, die ich geprüft habe, begann ich mit einer Reihe vordefinierter Regel. Dese Regel kommen zum Teil aus der Fachliteratur, zum Teil aus meiner eigenen Erfahrung und zum Teil von den Anwendern, mit denen ich zu tun hatte. Vor der Messung hat der Software-Vermesser die Möglichkeit diese Regel einzustellen. Unliebsame Regel kann er ausschalten. So kommt es mit den prozeduralen Sprachen vor, dass viele alte Programmierer das Verbot der GOTO Anweisungen aufheben oder dass junge Java Programmierer das Casting Verbot streichen wollen. Sogar der Sinn und Zweck der Kommentierung wird in Frage gestellt. An diesem Punkt darf ich erwähnen, dass ich wegen der fehlenden Abstimmung der Programmierregel mit dem Betriebsrat in dem GEOS Projekt in Schwierigkeiten geraten bin. Seitdem nahm ich mir immer vor, die Regeln mit den betroffenen Entwicklern abzusprechen. Da es in diesem Projekt so viele verschiedene Sprachen mit so vielen unterschiedlichen Zuständigkeiten gab, dauerte die Abstimmung ungewöhnlich lang. Es führte zu zusätzlichen Aufwänden, mit denen ich nicht gerechnet habe. Der Anwender hat sich wenig darum gekümmert. Die meisten IT-Manager wissen gar nicht, ob sie Programmierregel haben oder nicht, geschweige denn, welche sie haben. Ergo wissen sie nicht, welcher Aufwand damit verbunden ist, sie zu prüfen.

Schließlich kam die Einstellung der Qualitäts- und Komplexitätsmetrik hinzu. Der Vermesser hat die Möglichkeit, die einzelnen Metriken zu gewichten, z.B. kann Wiederverwendbarkeit stärker gewichtet als Testbarkeit sein. Hierzu muss der Vermesser wissen, was der Anwender mit der Software vorhat. Will er sie sanieren, will er sie

konvertieren oder will er sie nur erhalten, wie sie ist? Die Ziele des Anwenders bestimmen die Gewichtung der Messwerte. Die Messung sollte Zielgerichtet sein.

Deshalb hat die Vorbereitung der Messung so lange gedauert. Ich musste mit dem Anwender darüber diskutieren, wie gemessen wird, mit welchen Programmierregeln, wie die Metrik gewichtet wird und wie die Function-Points zu zählen sind. In diesem bestimmten Fall hatte ich mit vielen Anwendern zu verhandeln, denn jede Entwicklungsabteilung hatte eine eigene Umgebung. Die eine hatte noch Assembler und VaG, die andere PL/I und die dritte COBOL. Dann gab es die neuen Projekten mit C++ und Java. Es gab praktisch nichts in der IBM Welt, das hier nicht vorhanden war. Hinzu kam, dass die Schwaben besonders pedantisch sind. Sie wollten über den Messvorgang alles wissen. Ein Projektleiter hat behauptet, die GOTO Anweisung in PL/I sei eine notwendige und nützliche Steuerungsmechanismus. Ich wurde gezwungen diese Regelprüfung zu entfernen und so erging es vielen anderen Regeln. Je älter die Programmiersprache war, umso heftiger verteidigen die Entwickler ihre schlechten Eigenschaften, die sie so lange verwendet haben. Das Problem der Altsoftwarequalität ist also weniger die alten Sprachen als die alten Entwickler. Sie wollen nicht wahrhaben, dass sie ihre Programmiersprache, wie immer sie heißen soll, die ganzen Jahre falsch angewandt haben. Dann kommt ein Fremde daher und will sie dafür zur Rechenschaft ziehen.

11.1.4 Die Messung wird durchgeführt

Endlich im April 2008 war es so weit. Wir konnten mit der Messung beginnen. Wir haben dazu drei verschiedene PC's gebraucht. Ich habe mit meinem Laptop den neuen Code in C++ und Java analysiert. Rudi Majnar – mein ungarischer Mitarbeiter hat den alten Code in Assembler, Vag, Easytrieve, PL/I und COBOL analysiert. Außerdem hat er die Datenbankschemen für die VSAM, IMS und DB-2 Datenbanken analysiert, sowie auch die Bildschirmmasken. An meinem Laptop habe ich die OS-Job Control Prozeduren gemessen.

Nach nur einer Woche war die Vermessung als solche beendet. Wir hatten:

```
  7.854 PL/I Programme      mit 13.530 Include Members
 17.151 COBOL Programme     mit 24.712 Copy Members
  8.975 Assembler Programme mit  9.597 Makros
  3.353 Easytrieve Programme
  3.077 Pseudo Code Module
  3.270 IMS-Datenbankschemen
 31.476 IMS-DC-Masken
 22.090 OS-JCL Prozeduren
```

```
2.977 C++ Klassen und
1.977 Java Klassen
```

geprüft und gemessen. Dies waren 70.752.999 Codezeilen, davon 52.048.897 echte Codezeilen. Die Tools zählten über 42 Million einzelne Anweisungen und 704.918 Function-Points. Dies entspricht einem Function-Point per 60 Anweisungen, was durchaus im Einklang mit der Backfiring Methode von Caspers Jones im Einklang ist, auch wenn wir die Function-Points anhand der Ein- und Ausgaben sowie der Datenbankzugriffe wie die IFPUG vorschreibt, gezählt haben.

Justierung der Function-Points durch Einflussfaktoren

- Data Communication Facilities (Web Service = 5)
- System Type (Internet = 5)
- Response Time (< 3 Secs = 3)
- Client Customization (MultiClient/MultiLingual = 4)
- Transaction Rate (100,000: 1 Million per day = 3)
- Reliability (0.01 Errors per Function-Point = 3)
- Usability (Different user types in casual Mode = 4)
- Business Criticality (Business relevant = 3)
- Processing Complexity (0.5:0.6 = 3)
- Degree of Reusability (0.5:0.6 = 3)
- Migration Requirements (Databases = 2)
- Security Requirements (Internet Level 2 = 4)
- Multi Installation (n Versions in n Countries = 5)
- Changeability (0.5:0.6 = 3)
- Total of Influence Factors = 50 of 70
- 50 / 100 = 0.5 + 0.65 = 1.15
- Unadjusted Size = 2726 Function-Points
- Adjusted Size = 2726 x 1.15 = **3135**

11.1.5 Bewertung der Alternative

Wir brauchten noch einen Monat, um das Zahlenmaterial zusammenzutragen und in eine präsentierbare Form zu bringen. Hiermit hat mein Kollege, Richard Seidl, wertvolle Dienste geleistet. Er hat die Zahlen nochmals überprüft und in Tabellen und Graphiken dargestellt. Den Schlussbericht haben wir Ende Mai abgeliefert und im

Juni erfolgte die Präsentation mit den Konsequenzen aus der Messung. Der Anwender hatte fünf Alternativen:

1) er könnte die Systeme weiter warten wie bisher, also nicht tun,
2) er könnte die Systeme mit einer modernen Programmiersprache neu entwickeln mit einem horrenden Aufwand,
3) er könnte die Systeme sanieren, also sie in den alten Sprachen lassen, aber qualitativ nachbessern,
4) er könnte die Systeme soweit möglich konvertieren, d.h. mit eine moderne Sprache automatisch konvertieren, wobei dies setzt voraus, dass sie erst saniert werden,
5) er könnte einzelne, wirtschaftlich wertvolle Systeme Kapseln und in eine moderne Umgebung einbinden.

Kostenschätzungen für einen Geschäftsbereich

- Geschäftsbereich XYZ
  - 29,3 Mio. Lines of Code
  - 22,2 Mio. Source Statements
  - Languages: COBOL, ASM, PLI, CSP, IMS-DB, DB2, IMS-MFS, OS-JCL

Alternative	COCOMO-I [PM]	COCOMO-II [PM]	Analogie [PM]	Function-Point[PM]	Data-Point [PM]
Jährliche Wartung	294	250	280	239	298
Neuentwicklung	4.113	3.017		2.955	3.656
Sanierung	88	92	89		
Migration	412	424	329	311	372
Kapselung		82	58	71	89

Zu jedem dieser Alternative gab es eine Kostenschätzung basierend auf mindestens drei Schätzmethoden – COCOMO. Function-Point und Data- bzw. Object-Point. Die Kosten für eine Neuentwicklung waren natürlich bei diesem Systemumfang viel zu

hoch. Es blieben also nur vier echte Alternative übrig. Am Ende wurde entschieden zunächst die wichtigsten Systeme zu kapseln und den Rest so zu lassen wie sie sind. Wir hatten unseren Auftrag erfüllt, zu einem pro forma Aufwand von 60 Personentagen. In Wirklichkeit war es viel mehr. Später habe ich auf der internationalen Metrik-Workshop – IWSM2008 – darüber berichtet [Sned08].

11.2 Produktivitätsmessung in einer Java Umgebung

Auf das große Messprojekt für Mainframe Software in Stuttgart folgten mehrere kleinere Messprojekte in Wien. Zwei fanden in einer Java Umgebung statt, die Andere in einer DotNet Umgebung. Die zwei in einer Java Umgebung waren für Banken. Beide wollten die Produktivität ihrer Entwickler ermitteln um sie mit der Produktivität von Entwicklern in Osteuropa zu vergleichen. Die Banken standen unter Kostendruck und mussten sparen. Ein Weg dazu war die Auslagerung der Entwicklung und Wartung in den benachbarten osteuropäischen Ländern. Dort kosten die Entwickler nur die Hälfte. Die Frage ist, ob sie auch nur die Hälfte produktiv sind. Um diese Frage zu beantworten, soll die Produktivität der Entwickler im Ost und West verglichen werden.

Beide Anwender hatten Altsoftware im Einsatz. Der erste hatte PL/I und Java, der zweite COBOL, C++ und Java. Java und Java Script waren die Sprache des Frontendes. PL/I und COBOL waren die Sprachen der Hintergrundsysteme, die meistens schon älter waren. ANECON wollte mir ursprünglich einen jungen Kollegen zur Seite stellen um von mir zu lernen, sowie bereits in Stuttgart, dies ist aber leider nie zustande gekommen. Also blieb ich alleine und musste alles selber verrichten, von der Vorbereitung der Messung bis zur Auswertung der Ergebnisse.

Beim ersten Kunden ging die Messung der PL/I Programme glatt. Mit Java hatte ich ein Problem, weil unter den Java Sourcen viele Java Skript Sourcen waren. Ich bin hier zum ersten Mal mit Java Script konfrontiert und musste dafür einen abgewandelten Parser aus dem Boden stampfen. Es dauerte seine Zeit, dann konnte ich alle Java Source Bibliotheken analysieren. Die Vielfalt der Programmiersprachen hat sich wieder als Kostenfaktor bei der Analyse der Software erwiesen. Mit der Messung der PL/I Programme hatte ich keine Probleme auch wenn sie sowohl mit IMS als auch mit DB2 und CICS Makros durchsetzt waren. Ich musste um die Function-Points zu zählen die Masken, die JCL Prozeduren und die Datenbankschemen in die Messung einbeziehen. Am Ende kamen circa 15000 Function-Points für das Zielsystem mit allen Sprachen heraus. Das entsprach einen Migrationsaufwand von circa 300 Personenmonaten. In diesem Projekt habe ich zum ersten Mal mein neues Tool SoftCalc eingesetzt. Damit konnte ich nicht nur Entwicklungsprojekte sondern auch War-

tungs- und Migrationsprojekte automatisiert abgeschätzt. Die Produktivitätsdaten haben ich aus meinen früheren Migrationsprojekten entnommen [Sned07c]. Wie viele Personenmonate sie wirklich gebraucht haben konnte ich nie erfahren. Der Kunde hat sich aufgrund der hohen Kosten entschieden das alte System zu behalten.

Wenn man die extra Arbeit für den Java Skript Analysator abzieht, hat diese Messung nicht einmal 15 Tage gedauert. Mehr wollte der Kunde auch nicht bezahlen. Im Allgemeinen sind Anwender nicht bereit für Messung zu zahlen, für Test und Prüfungen ja, aber nicht für Messung. Da ich dies sehr wohl wusste, habe ich die Werkzeuganpassungen immer auf eigenen Kosten gemacht.

Der zweite Java Kunde war etwas großzügiger. Er hatte 20 Personentage bezahlt. Seine Absichten waren dieselben, wie die der ersten Kunde. Er wollte die Produktivität zwischen Entwicklungsstätten verglichen, denn auch diese Bank hatte Filialen in Osteuropa, wo sie Programmierkapazität hielt, nicht nur in Ungarn sondern auch in Polen und in der Slovakei. Es ging wieder um die Frage, wo ist es preiswerter zu entwickeln. Die Kosten für Entwickler variieren zwar vom Land zu Land, aber zu dieser Zeit waren sie in der Regel nur einen Drittel der Personalkosten in Wien. Die IT-Zentrale in Wien musste daher ihrem Management einen Beweis erbringen, dass sie ihr Geld wert war, für meine Begriffe – so wie ich die Wiener kenne – ein hoffnungsloses Unterfangen. Es konnte nicht sein, dass die Wiener dreimal produktiver als die Ungarn sind. Nach meiner Erfahrung ist es eher umgekehrt. Wenn man Wien als Standort der Softwareentwicklung wählt dann nicht wegen der Kosten sondern wegen der Nähe zu den Endanwendern. Dennoch habe ich das Projekt angenommen. Ich war gespannt zu sehen was dabei herauskommen würde [Sned07].

Wie bei den früheren Messprojekten, begann auch dieses mit einer Analyse der Messungsanforderungen. In diesem Messprojekt habe ich mich zum ersten Mal mit der Messung von Web Services befasst. Der Kunde hatte gerade angefangen solche Services im großen Stil einzuführen. Neben den herkömmlichen GUI Definitionen in Form von XSL Stylesheets gab es auch WSDL Schnittstellendefinitionen. Die GUI's habe ich als Eingabe und Ausgabe behandelt. In einer WSDL Schnittstelle habe ich jede Operation als Ein- und Ausgabe gezählt. Durch die Verwendung von Services steigt die Anzahl der Function-Points weil sie aus der Sicht der Klienten weitere Datenquellen sind. Jede einzelne Operation hat eine Eingabe und eine Ausgabe. Ob dies im Sinne der Function-Point Methode ist, ist eine Frage der Auslegung. Die Function-Point als Schätzgroße wurde Ende der 70er Jahren eingeführt als die Systeme viel einfacher waren. Es gab festformatierte Bildschirmmasken, Listen und Bänder. Da war es relative leicht Eingaben und Ausgaben anhand der Datenflussdiagramme zu

zählen. Heutzutage ist dies viel komplizierter. Es gibt Fenster in Fenster, Mausklicks, API's, Nachrichten, Mails und URLs. Hinzu kommen die vielen Nachrichten die zwischen den Systemen ausgetauscht werden, siehe das Internet of Things. Nicht die Menschen sondern die Maschinen kommunizieren miteinander. Was Eingabe und was Ausgabe ist, ist keineswegs eindeutig. Für mich hat die ursprüngliche Function-Point Methode schon längst ausgedient, sie ist nicht mehr Zeitgerecht. Dennoch solange als die deutschen und österreichischen IT-Manager immer noch fest daran glauben, muss ich mein Bestes versuchen sie so genau wie möglich zu zählen. Wer bin ich, um einer so weit verbreiteten Lehre zu widersprechen?

Dies bringt eine andere Frage auf, und zwar die der Berufsethik. Ich wurde oft beauftragt etwas zu tun, wovon ich selbst nicht überzeugt war. dazu gehört die Function-Point Zählung. Von mir aus hätte ich eine andere Methode vorgezogen um die Produktivität zu messen, aber da die Anwender gerade an dieser Methode glauben, sehe ich mich gezwungen ihre Wünsche zu erfüllen. Würde ich nur das tun was ich für richtig halte, wäre ich schon längst auf der Strecke geblieben. Unser Wirtschaftssystem zwingt einen sich untreu zu werden. Ich habe mich dabei immer unwohl gefühlt, auch hier, als der Kunde mir dauernd über die Schulter schaute. Der zuständige Mitarbeiter dort – ein junger Angestellte frisch von der Universität – wollte genau wissen, wie ich die Function-Points zähle und ob ich mich genau an den IFPUG Vorschriften halte. Dabei hat dieser Kunde selbst die Einflussfaktoren der IFPUG durch eigene ersetzt. Mit Recht. Die IFPUG Einflussfaktoren sind schon längst überholt. Solche Faktoren, wie ob die Stammdaten im Onlinebetrieb aktualisiert werden, waren vielleicht im Jahre 1979 relevant, heute nicht mehr [Albr79]. Heute sind andere Einflussfaktoren ausschlaggebend, Faktoren wie Sicherheit und Interoperabilität. Als die Function-Point Methode entstanden ist, hat man über Sicherheit nicht allzu viel nachgedacht.

Was die meisten IT-Anwender nicht wissen ist, dass zumindest in Europa die IFPUG Function-Points inzwischen durch COSMIC Function-Points abgelöst werden. In den wissenschaftlichen Gremien der IEEE und der GI wird nur noch nach COSMIC Function-Points gemessen. Das hat sich aber bei den meisten IT-Anwendern noch nicht herum gesprochen. Also denken sie weiter im Bezug zu IFPUG Function-Points. Hätte ich verlangt, dass sie auf COSMIC Function-Points umsteigen, hätte ich sie nur verunsichert. Wenn etwas eine Glaubenssache ist soll man nicht daran rütteln [Abra04]. In den USA ist die IFPUG Zählmethode immer noch vorherrschend. Darum baut die neue OMG Norm für die Zählung von Function-Points an Hand des Source Codes weiterhin auf der IFPUG Methode. Das Ganze ist letzten Endes sehr fragwürdig. Wenn die Function-Point Zählung wirklich ernst zu nehmen ist, muss es

möglich sein sie vorher aus der Anforderungsdokumentation zu entnehmen und nachher aus dem Code und beide Zählungen müssen weitestgehend gleich sein.

Auch in diesem Projekt musste ich verschiedene Sprachen bearbeiten. Für den Frontend gab es Java, Java Script, HTML, XML, XSL, XSD und WSDL. Für den Backend gab es Java, COBOL und C++. Hinzu kamen die SQL Datenbankschemen mit Stored Procedures. Ein Problem gab es mit den HTML/XSL GUI Definitionen. Es war nämlich nicht möglich in dem HTML-Code zu erkennen, welche Seiten Eingaben und welche Ausgaben sind. Bei der Function-Point Zählung wiegen die Eingaben 3 bis 6 und die Ausgaben 4 bis 7 Punkte. Keine der HTML-Experten konnten mir weiterhelfen. Zum Schluss habe ich entschieden, dass Masken mit mindestens einem variablen Feld als Eingabe zu zählen sind. Nur Masken mit ausschließlich festen Konstanten-Feldern zählten als reine Ausgaben. Dies hat in sich die Zählung schon verzerrt.

Die Function-Point Methode unterscheidet zwischen drei Komplexitätsstufen – hoch, niedrig und mittel. Je nach dem werden mehr oder weniger Function-Points erteilt. Um die Komplexität der einzelnen Entitäten hat mein Tool gemäß den IFPUG Richtlinie die Anzahl der Felder bzw. Attribute in den Datenbanktabellen, in den Bildschirmbeschreibungen und in den Schnittstellendefinitionen gezählt. Ich musste den Algorithmus des Tools dem Bankmitarbeitern ausführlich erklären, damit sie den Zählvorgang genau nachvollziehen konnten. Das Tool habe ich sogar erweitert damit die Anwender die Grenzwerte zwischen Komplexitätsgrade von außen steuern können. Jedenfalls habe ich einen riesend Aufwand betrieben, um Punkte zu zählen, deren Aussagewert ohnehin fragwürdig ist. Es ist gut, dass die Manager, die so viel über Function-Points reden und aus ihnen Schlüsse ziehen, keine Ahnung haben, wie sie zustande kommen. Aber dies trifft nicht nur für die Function-Point Messung zu. Die meisten IT-Manager haben wenig Ahnung über Software und wie sie in Detail funktioniert. Es ist auch vielleicht besser so. Sie sollten es gar nicht so genau wissen. Somit sind sie total abhängig von denjenigen, die es verstehen und jene sitzen am unteren Ende der Firmenhierarchie. Wer es weist hat keine Entscheidungskompetenz und wer es entscheiden muss hat kein Wissen. Diese Diskrepanz zwischen Managementbefugnis und Sachkompetenz hat seit der Jahrhundertwende erheblich zugenommen.

Ich habe in diesem Projekt unterschieden zwischen

- GUI-Points die aus den Benutzeroberflächen,
- Service-Points die aus den Service Schnittstellen,
- Data-Points die aus den Datenbankschemen und
- Programm-Points die aus den Moduln entnommen wurden.

Die Programm-Points bezogen sich hauptsächlich auf die Berichte und die Mitteilungen die von den Moduln herausgegeben werden. Ich habe mehrere Durchläufe gebraucht bis ich alle Servicearten verarbeitet habe. Die meiste Zeit ging darauf, die Analyseläufe vorzubereiten. Die Analyse selbst dauerte allenfalls ein paar Stunden, auch wenn mehrere Millionen Codezeilen betroffen waren. Am Ende des Tages hatte ich den rohen Function-Point Count.

Die rohe Function-Point Zählung soll laut der IFPUG Methode durch einen Einflussfaktor justiert werden. Dieser Faktor ist die gewichtete Summe 14 einzelner Faktoren. Diese Einflussfaktoren kommen aus der Uhrzeit der Datenverarbeitung, als die meisten Systeme im Batchbetrieb auf den liefen. Damals war es etwas Besonderes, die Datenbanken direkt im Online-Betrieb zu aktualisieren. Man hatte die Online-Datenbewegungen während des Tages aufbewahrt und erst nachts im Batchbetrieb in die Datenbank eingespielt. Die direkte Online-verarbeitung der Datenbanken galt als eine besondere Anforderung. Auch eine verteilte Datenverarbeitung galt als etwas Außerordentliches. Im Grunde genommen sind die Function-Point Einflussfaktoren keine echte Einflüsse, sondern Produkteigenschaften, welche den Entwicklungsaufwand beeinflussen können. Es hat mich nicht überrascht, dass dieser Kunde die Einflussfaktoren verändern wollte. Er schlug einige neue Faktoren vor, u.a. die geschäftliche Bedeutung der Anwendung, die Sicherheitsanforderungen und die Parametrisierbarkeit [SnHa09].

Ich habe diese neuen Einflussfaktoren in das Werkzeug eingebaut um die gezählten IFPUG Function-Points damit zu justieren. Die Zahl wurde dadurch um rund 25% erhöht. Die Ergebnisse der Zählung habe ich dem Bankmanagement präsentiert und der Auftrag war erledigt. Insgesamt hatte ich nicht mehr als 20 Tage gebraucht, um mehr als 20.000 Function-Points zu zählen. Ob der Kunde damit zufrieden war konnte ich nicht wissen. Beim Messprojekt in Stuttgart habe ich ein Feedback bekommen, hier nicht. Zum Schluss habe ich meinen Zweifel an der Vergleichbarkeit solcher Messungen gehabt. Man könne nur vergleichen, wenn alle Messungen auf der gleichen Art und Weise durchgeführt werden, die Messung der Ungarn wie die Messung der Wiener. Auch dann bleibt es fragwürdig, wie man die Systemgröße in Aufwand umsetzt. Es gibt keine allgemeingültige Produktivitätswerte, nur einzelne Erfahrungen die Ort- und Zeitgebunden sind. Letztendlich hat die Function-Point Zählung nur einen Sinn, wenn der Anwender weiß, was eine Function-Point bei ihm kostet. Dies trifft übrigens auch für die anderen Größenmaße zu. Das Ziel ist, Aufwand zu messen. Function-Points sind in dieser Hinsicht nur ein Mittel zum Zweck [Sned10]. Sie müssen in Aufwand umgesetzt werden. Das kann jedoch nur gelingen, wenn die Pro-

duktivität auch gemessen wird. Die Produktivitätsmessung ist eine unabdingbare Voraussetzung für die Projektaufwandsschätzung aber sie wird aus politischen Gründen peinlichst vermieden. Wenigstens in diesem war der Anwender bemüht, die Produktivität an den verschiedenen Standorten nachträglich zu messen. Wie es ausgefallen ist, habe ich nie erfahren.

11.3 Migrationsaufwandsschätzung eines 4GL Systems

Typisch für die Messprojekte in dieser Zeit war die Messung eines ADABAS-Natural Systems für die Planung, Produktion und Steuerung von holzverarbeitenden Maschinen in der Nähe von Graz. Die Firma hatte einen neuen IT-Leiter von der Uni angestellt und er wollte den Betrieb auf eine moderne Java-basierte Umgebung umstellen. Er hatte eine Schweizer Beratungsfirma engagiert um die Migration zu planen. Die Beratungsfirma gab mir einen Auftrag die alte 4GL Software zu messen und zu schätzen was es kosten würde sie in Java umzusetzen. Wie üblich gab sie mir nur eine Woche Zeit. Es soll ja nichts kosten.

Ich bin guten Mutes nach Graz gereist und habe gleich begonnen die Sourcen zusammenzustellen. Es gab über 8.000 Natural-2 Programmen mit mehr als 1,5 Millionen Codezeilen. Dazu kamen circa 1200 Natural Masken und 360 ADABAS Datenbanken. Ich konnte keine Zeit verlieren. Ich musste diese Codemasse auf meinen Laptop übertragen und gleich mit der Messung beginnen. Als erstes musste ich erleben, dass die für das Natural System zuständige Mitarbeiter keineswegs glücklich über dieses Vorhaben waren. Sie haben mir ihre Meinung von dem frischgebackenen IT-Leiter unverhohlen gleich mitgeteilt und dies ließ nichts Gutes ahnen. Der nächste Rückschlag kam als ich beim ersten Versuch die Masken zu analysieren feststellen musste dass die Maskenbeschreibungssprache sich verändert hat. Die Software AG hatte viele neue Features eingebracht und auch das Format geändert. Meine Natural Werkzeuge waren nicht auf dem neuesten Stand. Mein letztes Natural Projekt war bei dem Bremer Lagerhaus gewesen. Mein Maskenparser produzierte nur Hieroglyphen. Was tun? Unter den wachsamen Augen der Natural Entwickler musste ich zunächst Haltung bewahren. Ja nichts merken lassen. Ich habe die Analyse der ADABAS Datenbankschemen vorgezogen. Aber damit habe ich ebenfalls eine böse Überraschung erlebt. Die Struktur der Datenbeschreibung hat sich auch verändert. Also nur weiter machen als ob alles in bester Ordnung wäre. Wenigstens der Natural Parser hat funktioniert. Ich konnte damit beginnen die Programme zu prüfen und zu messen.

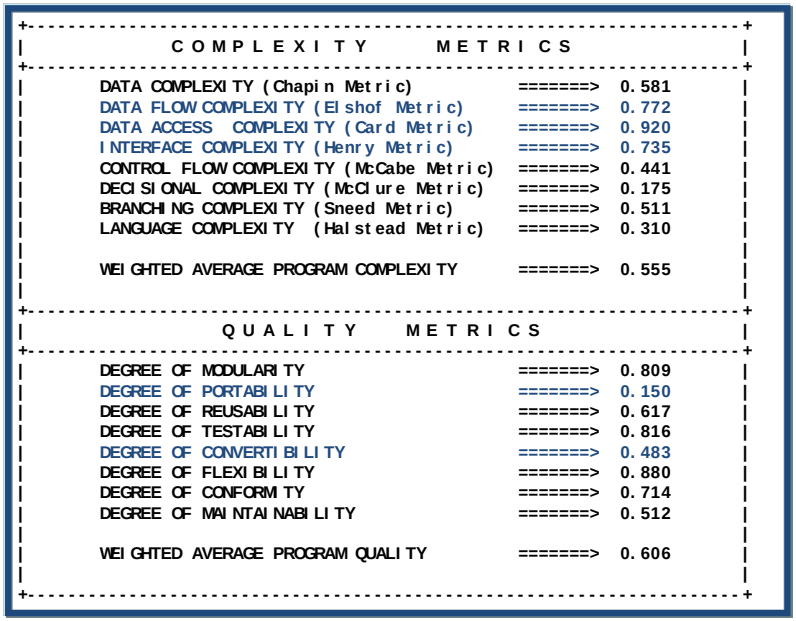
Abends bin zurück zu meiner Pension in einem Dorf in der Nähe des Betriebes außerhalb von Graz gegangen. Es war möglich dort durch den Bergwald zu Fuß zu

gehen. Ich habe mich hingesetzt und den Maskenparser Schritt für Schritt umgeschrieben. Bis 4 Uhr in der Früh lief er durch mit den Beispielmasken die ich mitgenommen hatte. Am nächsten Tag habe ich alle Masken analysiert und sowohl die Function-Points als auch die Data-Points gezählt. Am nächsten Abend hat sich der Prozess wiederholt. Wieder habe ich die Nacht durchgearbeitet und den ADAWAN Parser nachgebessert. Am nächsten Tag konnte ich alle Datenbankbeschreibungen analysieren. Inzwischen hatte ich alle Natural Programme gemessen. Bis zum Abend des dritten Tages war die Systemmessung abgeschlossen. Am vierten Tag konnte ich voll übernächtigt die Daten aggregieren und die Schätzung einer Neuentwicklung durchführen. Am fünften und letzten Tag konnte ich plangemäß meine Ergebnisse vor der IT-Leitung und den Teamleitern präsentieren [Sned13]. Der neue IT-Leiter war schockiert. Er hatte mit so einem hohen Aufwand nicht gerechnet. Die alten Natural Entwickler konnte ihre Schadenfreude kaum unterdrucken. Ich hatte bestätigt, dass sie mit Natural/Adabas sehr produktiv gewesen waren. Mit Java konnten sie diese hohe Produktivität kaum erreichen. Bei der Beratungsfirma die mich bestellt hat war ich untendurch. Ich habe von ihr nie wieder gehört.

11.4 Produktivitätsmessung in einer DotNet Umgebung

Meine nächsten Messprojekte fanden in einem Wiener Betrieb der öffentlichen Hand statt. Es war der gleiche Betrieb, bei dem ich mein erstes Testprojekt für die AN-ECON gemacht hatte. Das erste System, dass ich vermessen habe, war sogar das Gleiche, das wir früher getestet hatten. Ich hatte schon damals beim Testen erkannt, dass die Erhaltung dieses Systems viel kosten würde. Es ging alles zu schnell und die Entwickler waren unzulänglich vorbereitet. Außerdem wurde damals entschieden ein zentrales System für alle Bundesländer zu bauen. Das hat zu einer sehr hohen Komplexität der Software geführt. Wie dem auch sei, das System war jetzt im Betrieb und erforderte einen hohen Wartungsaufwand. Der zuständige Entwicklungsleiter, mit dem ich mich übrigens gut verstanden habe, stand unter gewaltigen Druck von seinem Top-Management und musste seine Kosten rechtfertigen. Er wollte wissen, ob dieser hohe Wartungsaufwand wirklich gerechtfertigt war. Dazu bräuchte er einen Vergleich mit anderen Systemen. Er suchte einen Nachweis, dass seine Entwicklungskosten sich in einem internationalen Durchschnitt bewegen.

Source Code Metrik einer DotNet Anwendung



```
+-------------------------------------------------------------------+
|              C O M P L E X I T Y      M E T R I C S               |
+-------------------------------------------------------------------+
|       DATA COMPLEXITY (Chapin Metric)          =======>  0.581    |
|       DATA FLOW COMPLEXITY (Elshof Metric)     =======>  0.772    |
|       DATA ACCESS  COMPLEXITY (Card Metric)    =======>  0.920    |
|       INTERFACE COMPLEXITY (Henry Metric)      =======>  0.735    |
|       CONTROL FLOW COMPLEXITY (McCabe Metric)  =======>  0.441    |
|       DECISIONAL COMPLEXITY (McClure Metric)   =======>  0.175    |
|       BRANCHING COMPLEXITY (Sneed Metric)      =======>  0.511    |
|       LANGUAGE COMPLEXITY  (Halstead Metric)   =======>  0.310    |
|                                                                   |
|       WEIGHTED AVERAGE PROGRAM COMPLEXITY      =======>  0.555    |
|                                                                   |
+-------------------------------------------------------------------+
|                  Q U A L I T Y    M E T R I C S                   |
+-------------------------------------------------------------------+
|       DEGREE OF MODULARITY                     =======>  0.809    |
|       DEGREE OF PORTABILITY                    =======>  0.150    |
|       DEGREE OF REUSABILITY                    =======>  0.617    |
|       DEGREE OF TESTABILITY                    =======>  0.816    |
|       DEGREE OF CONVERTIBILITY                 =======>  0.483    |
|       DEGREE OF FLEXIBILITY                    =======>  0.880    |
|       DEGREE OF CONFORMITY                     =======>  0.714    |
|       DEGREE OF MAINTAINABILITY                =======>  0.512    |
|                                                                   |
|       WEIGHTED AVERAGE PROGRAM QUALITY         =======>  0.606    |
|                                                                   |
+-------------------------------------------------------------------+
```

So habe ich begonnen meine C# Messinstrumente zu verfeinern und zu verbessern. Im Gegensatz zu Java bei der der Entwickler seine Dienstfunktionen selber definieren muss, werden in C# die meisten Dienstfunktionen von dem DotNet System bereitgestellt. Somit sind sie standardisiert und leichter erkennbar. Der C# Code ist zwar Dotnet spezifisch, lässt sich aber leichter verarbeiten. Ich konnte in der vereinbarten Zeit von 15 Tagen den Code prüfen und vermessen. Auch hier habe ich die Toolentwicklung außerhalb der abgerechneten Zeit betrieben. Die Vermessung selbst dauerte nur wenige Tage. Die meiste Zeit ging drauf die Messung vorzubereiten und am Ende zu erklären. Ich habe die Produktivitätsdaten aus der europäischen Studie von Kathrin Maxwell genommen und auf die gezählten Function-Points übertragen [Maxw02]. Es ergaben sich 22 Function-Points pro gebuchten Personentag. Also konnte der IT-Leiter beruhigt sein. Er lag im europäischen Durchschnitt. Was nicht in dieser Rechnung drin war, waren die Kosten, die noch auf ihn zukommen würden um das System zu erhalten.

11.5 Schätzung der Wartungskosten für ein PPS System

Die Schätzung von Wartungsaufwänden stellt die Schätztechnik vor einer besonderen Herausforderung. Es werden zwar die gleichen Maßzahlen verwendet aber auf einer anderen Art und Weise. Im Mittelpunkt steht die Anzahl Anweisungen, denn nicht die Funktionalität sondern der Code wird gewartet. Der Aufwand bezieht sich auf die Codegröße, die nicht unbedingt im Verhältnis zur Funktionalität. Wenn der Code unnötig aufgebläht ist wird es mehr Anweisungen für weniger Function-Points geben. Das Verhältnis zwischen Codezeilen und Function-Points ist keineswegs fix. Es kann sehr stark variieren je nachdem wie gut der Code implementiert ist.

Das DotNet System, um welches es hier geht, dient zur Steuerung der Produktion von Automobilteile in diversen Werken verteilt durch Europa. Das System hat eine Master-Version, die an einer zentralen Stelle gewartet wird. Diese Version ist aber nirgendwo im Einsatz, sie dient lediglich als Muster für die lokalen Versionen. Jedes Werk hat eine eigene lokale Version der Software, die mehr oder weniger von der Grundversion abweicht. Bei manchen Werken ist sie fast identisch zur Grundversion, in anderen hat sie fast das Zweifache an Größe und Funktionalität. Trotz der Unterschiede in die Funktionalität wurde angestrebt, die Software aus Kostengründen an einer zentralen Stelle für alle Werke zu pflegen. Das Ziel des Schätzprojektes war es, diese Entscheidung zu rechtfertigen [SnEr12].

Wie bei allen Schätzprojekten dieser Art war auch dieses zeitlich und aufwandsmäßig begrenzt. Die Schätzung durfte nicht länger als 20 Tage dauern und nicht mehr als 15 Personentage kosten. Sie begann mit der Sammlung der Source-Code Komponente aus vier stellvertretenden Werken plus die Master-Version. Die Source-Bibliotheken bestanden aus C# Klassen, XSAML Oberflächendefinitionen, XRES resource files und XSD bzw. WSDL Schnittstellendefinitionen. Es gab insgesamt

```
   20.388 Source-Dateien mit
2.800.000 Codezeilen.
```

Allerdings gab es die Mehrzahl der Source-Dateien mehrfach, da sie bei jedem Werk die gleichen waren. Es gab aber auch unterschiedliche Varianten der Sourcen für jedes Werk und auch ganz andere Sourcen. Das größte Werk hatte 5418 Sourcen, das kleinste nur 3405.

Es war deshalb erforderlich, die Software Bibliotheken getrennt zu halten und jedes Werk für sich zu messen. Das Ziel war die Größe, Komplexität und Qualität der Software für jedes Werk zu messen und daraus den jährlichen Erhaltungsaufwand abzuleiten. Die Summe aller Werksaufwände ergibt dann der Gesamtaufwand für die

dezentrale Lösung. Der Gesamtaufwand für die zentrale Lösung wurde errechnet als die Summe der Aufwände für jedes Werk minus der Aufwand für die Erhaltung der Master-Version, die dann nur einmal dazu addiert wurde.

Ausgegangen wurde von drei Wartungstypen (Fehlerkorrekturen, Änderungen und Erweiterungen) und der Änderungsrate der Software pro Kilo Anweisungen pro Jahr. Für die Wartungseingriffe wurde durchschnittlich 1,5 PT berechnet, einschließlich Test. Dies basiert auf der Erfahrung in den ersten zwei Jahren. Daraus ergaben sich folgende Aufwände:

	Justierte Größe (Stmts)	Eingriffe	Aufwand (PT)
System-A	145.575	436	654
System-B	154.255	462	693
System-C	213.250	639	958
System-D	245.969	738	1.107
System-M	134.912	270	405

Wenn jedes der vier Werke allein für sich gepflegt wird, werden 3412 Personentage pro Jahr benötigt, bzw. 17 Personen in den 4 Werken.

Wenn die Software zentral gepflegt wird, werden nur 2197 Personentage pro Jahr benötigt.

$$\{ \Sigma (Werksaufwände) - Masteraufwand \} + Masteraufwand$$

Das ergibt 11 Wartungsingenieure für die vier Werke, ein Ersparnis von 6 Personenjahren. Je mehr Werke dazukommen und je größer der Anteil der Master-Version am gesamten Source-Code, desto höher das Ersparnis. Dies spricht für einen möglichst großen, wiederverwendbaren Codeanteil.

Der Unterschied wird noch krasser, wenn wir die Managementkosten berücksichtigen. falls jedes Werk die Wartung seines Systems selber betreibt, muss es auch die Overhead-Kosten tragen. Diese wären für

```
System-A      654 / 2 =   327 PT
System-B      693 / 2 =   346 PT
System-C      958 / 2 =   479 PT
System-D    1.107 / 2 =   554 PT
Insgesamt   3.412 / 2 = 1.706 PT Management-Overhead.
```

Zusammen mit den Overhead-Kosten kommt einen Aufwand von 4512 Personentagen, bzw. 226 Personenmonate, zustande. Daraus leitet sich eine Kopfzahl von 25 Personen ab.

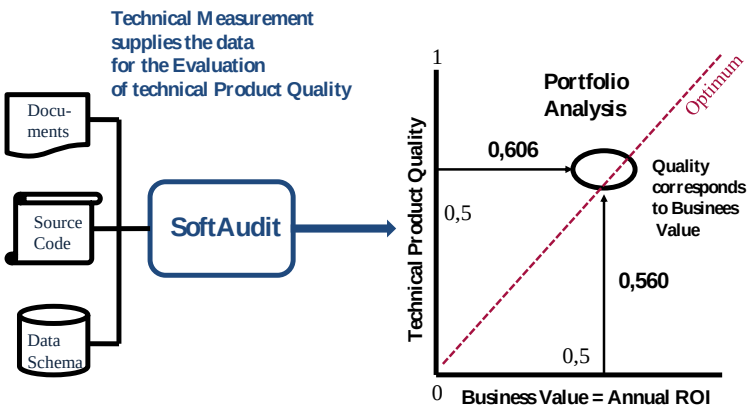
3412 PTs + 1706 PTs = 5118 PTs pro Jahr bzw. 25 Personen.

Die zentrale Lösung würde nur 1300 Personentage Overheadkosten verursachen.

```
System-A    (654 - 405) / 2 =   125 PT
System-B    (693 - 405) / 2 =   144 PT
System-C    (958 - 405) / 2 =   277 PT
System-D (1.107 - 405) / 2 =   551 PT
System-M           405  / 2 =   203 PT
Insgesamt                     1.300 PT Management-Overhead.
```

Dazu kämen 2.197 PT für die Wartungseingriffe. Gesamtkosten für die zentrale Lösung wären also 2197 + 1300 = 3497 Personentage. Das sind nur 18 Personen, bzw. 2/3 der Kosten der verteilten Erhaltung. So gesehen war die zentrale Wartung der verteilten Werkssysteme eindeutig die wirtschaftlichere Lösung. Die richtige Schätzung der zu erwartenden Wartungskosten ist eine entscheidende Voraussetzung für eine nachhaltige Softwareevolution. [Sned12].

Zuordnung der technischen Qualität zum betriebswirtschaftlichen Nutzen



11.6 Der Software Turm von Babel

In einem Zeitrahmen von vier Jahren von 2009 bis 2013 habe ich insgesamt fünf DotNet Systeme für die Österreichische Wirtschaftskammer vermessen – das ursprüngliche Gebührenabrechnungssystem, das sanierte Gebührenabrechnungssystem, ein Lehrling-Ausbildungssystem, ein Wahlausführungssystem und ein betriebliches Webportal. Alle fünf Systeme liefen in einer DotNet Umgebung und waren in der Sprache C# implementiert. Dazu kamen die XML und WSDL Schnittstellen sowie die SQL Datenbanken. Jeder diese Sprachen hat einen eigenen Syntax und Semantik und muss extra vermessen werden. Denn es nutzt nicht nur die Sourcen in der Kernsprache wie hier C# zu analysieren. Wie bei den anderen Messprojekten, müssten die GUI Templates, die Schnittstellendefinitionen und die Datenbankschemen herangezogen werden, denn auch diese Elemente gehören zum System und tragen zur Systemgröße bei. Ergo braucht der Vermesser nicht einen, sondern mehrere Werkzeuge, eins für jede Sprache, die er zu verarbeiten hat. Für diese Messprojekte benutzte ich außer dem C# Analysator auch einen SQL Analysator, einen XML/XSL Analysator und einen WSDL Analysator.

Moderne Webanwendungen sind besonders schwer zu messen. Da kommt alles Mögliche zusammen – C#, PHP, Java, Java Script, HTML, XML, XSL und SQL. Wenn auch noch Web Services benutzt werden hat man mit WSDL oder WADL Schnittstellen zu tun. Erschwerend für die Function-Point Zählung ist das die Software in mehreren Schichten aufgeteilt ist. Es gibt eine konventionelle Frontend mit DotNet oder Java Swing, eine Web Frontend mit PDF HTML und Java Script, eine Backend mit Java oder C# und eine Datenzugriffsschicht mit SQL. Es ist der wahre Turm vom Babel. Wer soll daraus einen Reim machen. Das Ganze zu integrieren ist eine wahre Herausforderung über die ich 2003 zusammen mit meinem Sohn Stephan ein Buch geschrieben habe [SnSn03].

Die Frage bei den Function-Points ist, welche Ein- und Ausgaben sollte man zählen – nur die zwischen dem Endbenutzer und dem Frontend oder auch die zwischen Frontend und Backend. Eine Ausgabe des Frontendes ist eine Eingabe für das Backend und eine Ausgabe des Backendes eine Eingabe für das Frontend. Die COSMIC Function-Point Methode ist in dieser Hinsicht gründlicher weil sie näher zur Technik ist, aber viele Anwender beharren weiterhin auf die IFPUG Methode, die eigentlich schon längst überholt ist, weil ihre bisherigen Zählungen auf dieser Basis gemacht wurden. Am besten wäre es eine völlig andere Zähleinheit für moderne servicebasierte IT-Anwendungen zu finden [Sned15].

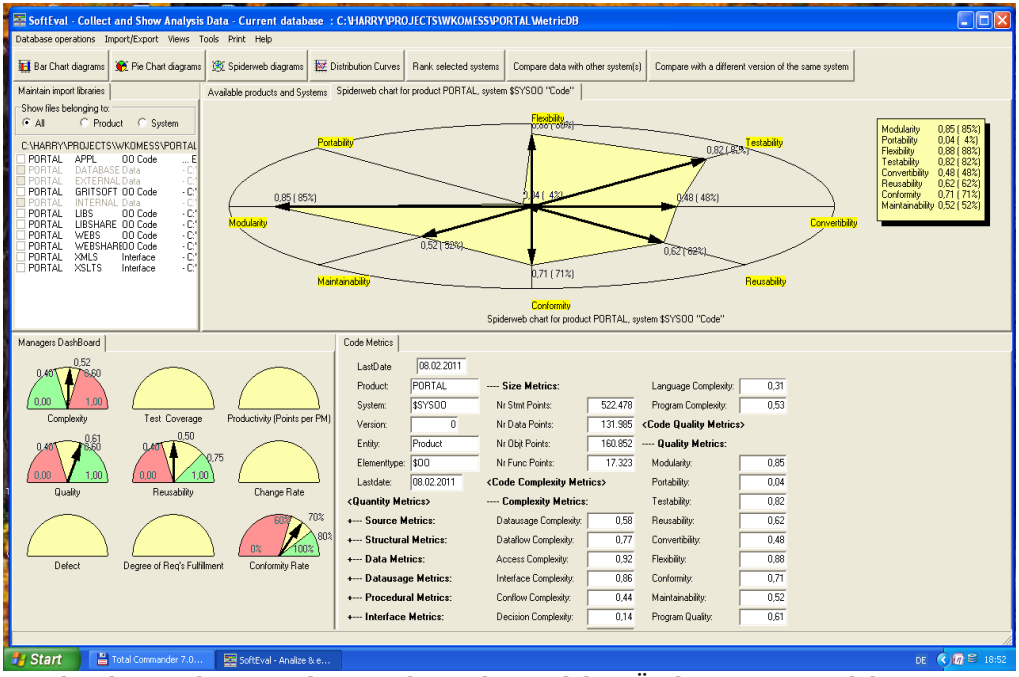
Das Schwierigste bei der Vermessung ist die Weiterentwicklung und das Tuning der eigenen Messwerkzeuge. Das nimmt viel Zeit in Anspruch, Zeit, die selten vom Kunden bezahlt wird. Man kann es auch nicht erwarten. Kunden bezahlen für ein Service wie ein Test, eine Sanierung oder eine Messung. Wie das Service geleistet wird, ob mit Tools oder per Hand, ist Sache des Dienstleisters. Die ganze Toolentwicklung muss also parallel zum Service habe ich immer auf eigene Kosten gemacht abends und unterwegs in dem Zug. Das war schon immer der Preis dafür dass ich überhaupt solche Projekte bekommen konnte. Oft wurde ich gefragt, warum ich nicht kommerzielle Werkzeuge vom Markt benutze. Ich hatte das schon einmal probiert. Wenn Fehler in den fremden Werkzeugen auftreten, was bei der Verarbeitung solcher Codemengen in diversen Sprachen unvermeidlich ist, ist man auf den Toollieferant angewiesen. Es kann Wochen dauern bis er reagiert. Die Messprojekte müssen aber in wenigen Tagen durchgeführt werden – ich hatte keine Zeit zu warten. Mit den eigenen Werkzeugen konnte ich die Fehler selber beseitigen, auch wenn es eine ganze Nacht dauerte. Nicht selten habe ich im Hotel irgendwo die Nacht damit verbracht, einen Fehler zu beseitigen, damit ich am nächsten Tag weiterarbeiten konnte. Nie hatte ich ein Mess- oder Testprojekt wegen der Werkzeuge abbrechen müssen. Die Probleme habe ich immer selber beseitigen können. Dies ist der Preis dafür, dass man solche Dienstleistungen wie Softwarevermessung überhaupt anbieten kann.

11.7 Weiterentwicklung der Messwerkzeuge

Am Ende bin ich mit den eigenen Werkzeugen nicht schlecht gefahren. Ob beim Testen, Prüfen, Messen oder Transformieren kam ich immer zu einem Ergebnis. Über die Qualität der Ergebnisse könnte man streiten, aber das ist bei Software immer so. Qualität ist relativ und wenn einer keine Alternative hat, ist ein qualitativ minderwertiges Ergebnis immer besser als gar kein Ergebnis. So schlecht waren die Werkzeuge auch nicht. Soviel ich weiß, gibt es auf der Welt keine andere Werkzeuge, die so viele verschiedene Sprachen verarbeiteten, wie die meine. Zu diesem Zeitpunkt konnte ich schon fünf prozedurale Sprachen - Assembler, PL/I, RPG, C und COBOL – fünf 4GL Sprachen – Natural, CSP, Delta, Powerbuilder und ABAP – sowie fünf objektorientierte Sprachen – C++, C#, Java, JavaScript und PHP verarbeiten und die Ergebnisse mehr oder weniger auf einen Nenner bringen. Für die Messung war das besonders wichtig, denn ich musste prozedurale, 4GL und objektorientierte Systeme miteinander vergleichen. Nach der Analyse jeder einzelnen Sprache gilt es die Sprachergebnisse auf ein gemeinsames Nenner bringen. Das Ziel ist die Gesamtgröße und die mittlere Komplexität und Qualität zu ermitteln. Für Aufwandsschätzungen wird die Größe durch die Komplexität und Qualität justiert. Die justierte Größe lässt sich dann über die Produktivitätswerte in Sollaufwand umsetzen. Alles setzt auf der Messung der

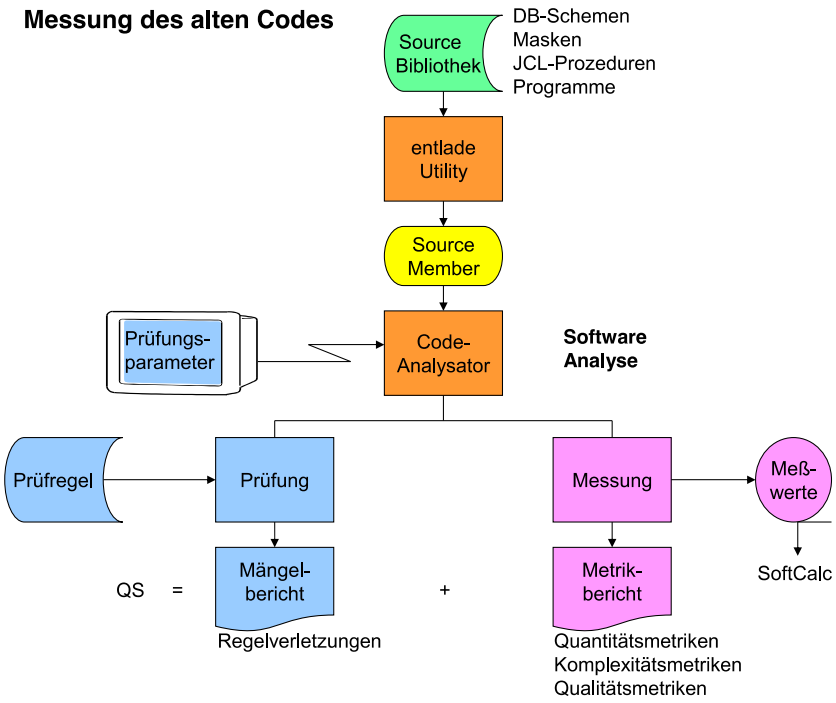
Software auf. Nicht nur den Code, auch den Test, den Entwurf und die Anforderungsspezifikation konnten voll automatisch gemessen werden.

Visualisierung der Auditergebnisse



Darüber hinaus konnte ich ausrechnen, bei welcher Änderungsrate welcher Wartungsaufwand erforderlich sei. Für die Schätzung der Entwicklung habe ich die Größe in Function-Points, Data-Points und Object-Points benutzt, für die Wartungsschätzung die Größe in Anweisungen. Codezeilen habe ich nie benutzt, weil ich weiß, wie fragwürdig diese Zählung ist. Einer, der sich sehr viel mit Code in verschiedenen Sprachen befasst, merkt bald, wie ungleich Codezeilen sind. Es hängt von der Sprache, von dem Editor und von dem jeweiligen Programmierer ab, wie viele Zeilen er wo schreibt. Viel stabiler sind die Anzahl Anweisungen, da eine Anweisung in allen Sprachen in etwa das Gleiche ist, sowie die Sätze in natürlichen Sprachen. In den Anforderungstexten habe ich Sätze, Anforderungen, Regel, Datenobjekte, Systemschnittstellen und Anwendungsfälle gezählt, in den Entwurfsmodellen Entitäten – Klassen, Methoden, Objekte, Zustände, Aktivitäten, Komponente und deren Beziehungen gezählt, in den Codetexten die Anweisungen, Daten, Module, Methoden, Variabel, Verweise, Bedingungen und Datenbankzugriffe, in den Datenbankschemen,

die Tabellen, Schlüssel, Indexes und Attribute, und in dem Test die Testfälle, die Testprozeduren und die Testdaten gezählt. Alle dieser Zählungen flossen in eine einzige große Metrik-Datenbank hinein und wurden dort von dem Tool SoftEval statistisch ausgewertet.



Damit war ich nicht weit entfernt von den ursprünglichen Ziele des EU MetKit Projektes. Es hat mich gefreut mit meinen Messwerkzeugen einen wichtigen Beitrag zur Förderung der Software-Messtechnik in Europa zu leisten. Auch in fernem Finnland hat eine Forschungsgruppe meine Messwerkzeuge – SoftAudit, SoftEval und Softcalc - benutzt um Wartungskosten zu projektieren [KKSM08].

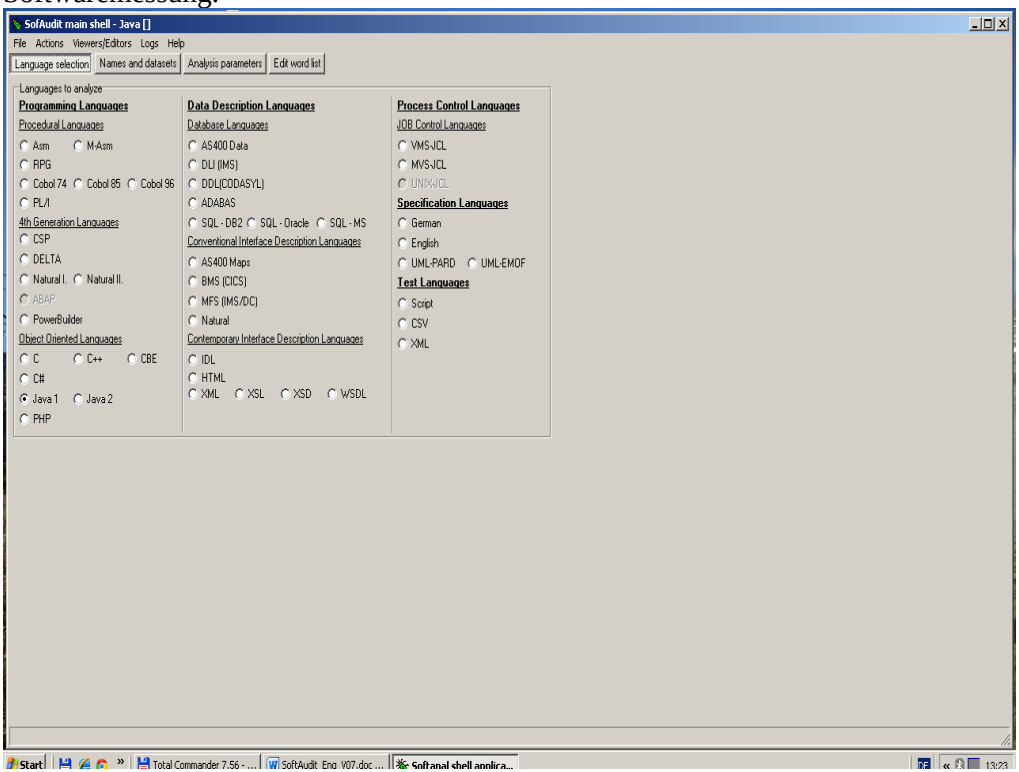
Aufwandschätzung einer Auftragsbearbeitung

```
Kalkulationsergebnisse / F u n c t i o n - P o i n t - M e t h o d e
Einzelparameter:
Overhead: 0,15   Änderungsrate: 0,12   geforderte Laufzeit:   8 Monate

Ergebnisse:

Aufwand in Mannmonaten.......9,26     FPT unjustiert..............219
Computer-Stunden.............0,00     FPT Qual.-justiert...........216
Kosten..................92561,30     FPT justiert.................205


Qualitäts-Rate..............0,68     Anzahl Datenobjekte...........26
Qualitäts-Faktor............0,98     Anzahl Schnittstellen.........42
Einflussfaktor..............0,95     Anzahl Prozesse..............10


        Minimale Projektdauer..................5,8 Monate
        optimale Besetzung.....................2 Personen
        geschätzter Wartungsaufwand............1,2 Mannmonate pro Jahr
```

11.8 Das Ziel ist die Vergleichbarkeit

Immerhin ist es mir gelungen, vergleichbare Maßzahlen aus sehr unterschiedlichen Systemen abzuleiten und auszuwerten. Mit den Werkzeug SoftEval konnte ich eine Metrikdatenbank aufbauen und die darin enthaltenen Messwerte auswerten. Mit dem Werkzeug SoftCalc konnte ich aufgrund der gewonnenen Messwerte Aufwände nach sechs verschiedenen Schätzmethoden kalkulieren – COCOMO, Data-Point, Function-Point, Object-Point, UseCase-Point und Test-Point. Ich war immer sehr bemüht die diversen Ansätze einzubeziehen, um mehrere Sichten auf die Schätzproblematik anzubieten. Die Sicherheit liegt in der Redundanz. Ob das die Kunden zu schätzen wussten, mag ich bezweifeln. Mein Eindruck ist, dass die wenigsten Software Anwender die Komplexität ihrer Softwaresysteme begreifen. Den Entwicklern ist es egal. Sie leiden nur darunter. Die Führungskräfte sind meistens von dem Code meilenweit entfernt und wollen darüber nichts wissen. Sie suchen nach einfachen, leicht erklärbaren Lösungen. Leider konnte ich damit nicht immer dienen. Ein komplexes

Thema lässt sich nicht ohne weiteres einfach erklären. Dies ist das Dilemma der Softwaremessung.



12 Das Ende in Wien

12.1 Umsetzung von BULL-COBOL in Java für den Wiener Flughafen

Im Frühjahr 2009 bekam ich von der ANECON den Auftrag ein altes Honeywell-Bull System am Wiener Flughafen zu messen. Es war mein letztes Jahr in der Pension in Oberlaa. Es war geplant die Pension dort am Kurpark abzureisen in der ich seit 1998 unter der Woche wohnte. Das machte mich sehr traurig denn ich hatte mich daran gewohnt mit dem Rad über den Laaberg durch Favoriten in die Stadt zu fahren. Ich brauchte knapp anderthalb Stunde. Noch konnte ich von Oberlaa aus durch Schwechat zum Flughafen radeln. Das dauerte gut eine Stunde. Auf dem Rückweg gab es einen Heurigen am Liesingbach wo ich einkehren konnte. Wie oft saß ich dort in der Laube am Abend bei einem Glas Wein und sinnierte über die Ungerechtigkeit des Lebens. Das brachte mich aber nicht weiter, man muss sich seinem Schicksal stellen. Jetzt hatte ich eine sehr schwierige Aufgabe zu bewältigen. Bereits zwei große Softwarehäuser hatten ihre Zähne an dem alten Flughafen System ausgebissen. Jetzt sollte ich etwas bewirken. Nichts wie Augen zu und durch. In meinem Alter hatte ich nichts mehr zu verlieren.

12.1.1 Die Ausgangssituation

Wie so viele alte Systeme war das System für die Steuerung der Airport Operationen in den 80er Jahren mit der Technologie jener Zeit implementiert. Nur in diesem Fall war die Technologie schon damals ein Exote. Aus welchen Gründen auch immer hat das Flughafen Management damals sich für einen Bull Rechner aus Frankreich entschieden. Die Maschine hatte manches gemeinsam mit dem Unisys Rechner bei der UBS. Also war mir seine seltsamen 30-Bit Wortstruktur nicht unbekannt. Auch meinen ungarischen Mitarbeiter Majnar Rudi kannte sie gut. Er hatte damals die Univac Assembler Programme umgesetzt.

Das Projekt begann wie üblich mit einer Messung der Programme und Datenbanken. Für beides musste ich meine Messwerkzeuge anpassen. Die Datenbanken waren nach dem CODASYL DDL Schema aus dem Jahre 1968 organisiert mit Areas, Sets, Owners, Members and Records. Ich hatte schon IDMS Datenbanken analysiert aber noch keine IDS Datenbanken. Es gab nur geringfügige Unterschiede aber genug um meinen CODASYL Parser ändern zu müssen. Es gab insgesamt 59 Datenbanksche-

men mit 27.268 Schemazeilen. Wegen der Anpassungen zum CODASYL Mess-Tool dauerte allein die Vermessung der Datenbanken über einen Monat. Es ergaben sich zum Schluss folgende Zahlen: 38 Areas, 1475 Satztypen, 44 Sets, und 19973 Datenfelder davon 51 Schlüsselfelder. Die Datenkomplexität war wegen der komplexen Struktur mit 0,628 relative hoch und die Datenqualität wegen der hohen Anwendungsabhängigkeit mit 0,404 zu niedrig. Was war dazu zu sagen. Die Datenbankstruktur ist über die Jahre degeneriert als sie mehrfach ergänzt und erweitert wurde. Anwender wollen es nicht wahr haben, aber auch ihre Daten veralten.

C O D E Q U A N T I T Y M E T R I C S		
Number of Sources analyzed	=======>	1662
Number of Source Lines in all	=======>	1090755
Number of Genuine Code Lines	=======>	971709
S T R U C T U R A L Q U A N T I T Y M E T R I C S		
Number of Modules	=======>	630
Number of Copy/Includes	=======>	1032
Number of Copy/Include References	=======>	7483
Number of Entry Points	=======>	630
Number of Exit Points	=======>	662
Number of Sections/Procedures	=======>	10443
Number of Labels/Paragraphs/Code Blocks	=======>	72143
Number of Reusable Code Blocks	=======>	6539
Number of Data Structures/Objects	=======>	31008
Number of Reusable Data Objects	=======>	11976
D A T A Q U A N T I T Y M E T R I C S		
Number of Panels processed	=======>	445
Number of Reports produced	=======>	169
Number of Files declared	=======>	928
Number of Data Bases accessed	=======>	431
P R O C E D U R A L Q U A N T I T Y M E T R I C S		
Number of Statements	=======>	890486
Number of Procedural Statements	=======>	647302
Number of Call Statements	=======>	9007
Number of Perform Statements	=======>	33244
Number of Selections (If & Case)	=======>	108929
Number of Loop Statements (Until/While)	=======>	30113
Number of GOTO Branches	=======>	90263
Number of all Control Statements	=======>	232442
Number of Function-Points	=======>	13876

Für die COBOL Codeanalyse musste ich an dem entsprechenden Werkzeug nicht allzu viel ändern. Die DML Anweisungen für die Datenbankzugriffe hatte ich schon bei der Warburg Bank in Zürich hinlänglich getestet. Was neu dazu kamen waren die Bull-spezifischen Datentypen sowie die Bull-spezifischen TP Operationen Für sie musste ich meinen Parser ausbauen. Nach drei Wochen konnte ich die gesamte Codeanalyse durchführen. Die wichtigsten Zahlen waren: 1662 Sources, 971.709 echte Codezeilen, 630 Module, 31008 Datengruppen, 445 Benutzerschnittstellen, 72.143 Codeblöcke mit 647.302 prozeduralen Anweisungen, davon 90263 GO TO Anwei-

sungen. Die Ablaufkomplexität war deshalb besonders hoch und die Codequalität äußerst niedrig. Nicht nur wegen der vielen GOTO Verzweigungen sondern wegen der zu großen Modulen und der hohe Anzahl Daten aus unterschiedlichen Quellen. Der ursprünglichen Code war scheinbar von alten Assembler Programmieren geschrieben wurden, denn die Namen der Daten waren alle verstummelt und die Namen der Prozeduren nummeriert.

Ausschnitte aus einem BULL-COBOL Programm

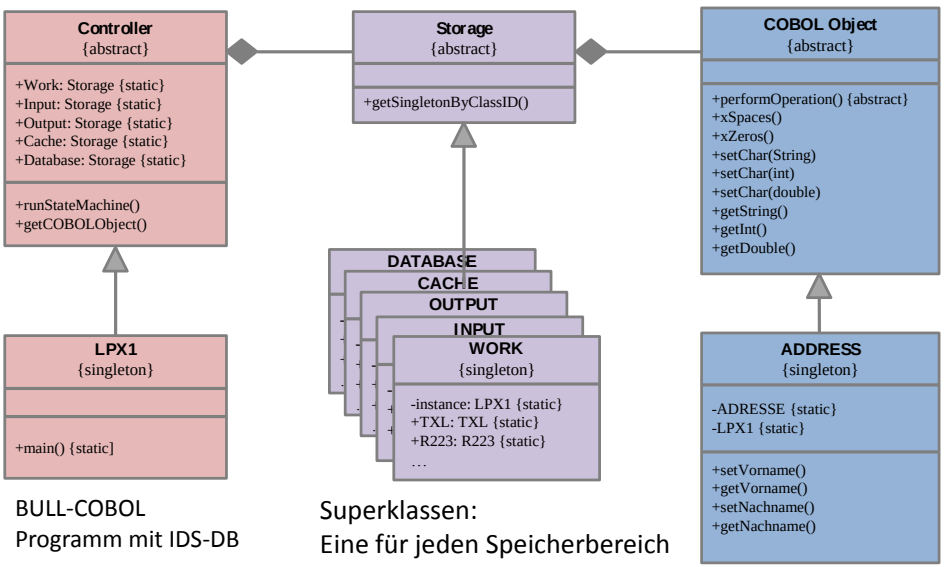
```
029140    IF   T (W - 1) NOT < “”33“”
029150                         GO TO 1212P.
029160                         GO TO 1270P.
029170 1230P.
029180    IF   T (W) NOT = "S"
029190                         GO TO 1212P.
029200    IF   T (W + 1) NOT = "O"
029210                         GO TO 1212P.
029220    IF   T (W + 2) NOT = "M"
029230                         GO TO 1212P.
029240    IF   T (W + 3) NOT < ""33""
029250                         GO TO 1212P.
029260*   IF   T (W - 1) NOT < ""33""
029270*                        GO TO 1212P.
029280                         GO TO 1270P.
029290 1231P. ← Keinerlei Kommentare
029300    IF   T (W) NOT = "B"
029310                         GO TO 1212P.
029320    IF   T (W + 1) NOT = "K"
029330                         GO TO 1212P.
029340    IF   T (W + 2) NOT = "G"
029350                         GO TO 1212P.
029360*   IF   T (W + 3) NOT < ""33""
029370*                        GO TO 1212P.
029380*   IF   T (W - 1) NOT < ""33""
029390*                        GO TO 1212P.
029400                         GO TO 1270P.
031400    IF   T (W + 7) NOT = TXREGI (8)
031410         GO TO 1245P.
31420     GO TO 1270P.
31421  1245P.
031440    SET  W UP BY 1.
031450    IF   T (W) = "N“
031460       IF   T (W + 1) = "N"
031470          IF   T (W + 2) = "N"
031480             IF   T (W + 3) = "N"
031490                      GO TO 1260P.
031500    IF   W > 1000
031510                      GO TO 1260P.
031520   GO TO 1243P.
031530 1260P.
031540    ADD  1 TO CHECKCT.
031550    IF   CHECKCT > 100
031560                      GO TO 1400P.
31570     IF   TXSW = 53 031580
31571        IF   CHCPCT > 30
031590                      GO TO 1265P.
031600    IF   WA102 = "+"
031610       COMPUTE 39TXNR = 39TXNR + 1
31620     ELSE
31621        COMPUTE 39TXNR = 39TXNR - 1.
031630    MOVE 39TXNR TO WA104.
031640    IF 39TXNR NOT < 1
031650                      GO TO 1208P.
```

Es war jedenfalls klar dass dieser Code nicht 1:1 umsetzbar in Java wäre. Der Java Code musste neu geschrieben werden aber wie. Um den neuen Code zu schreiben brauchten die Java Entwickler eine Beschreibung des alten Codes. Ich habe mit den zuständigen Manager vom Flughafen über die verschiedenen Alternativen der Nachdokumentation gesprochen. Am Ende schien die beste Lösung zu sein, den Code doch in Java umzusetzen und den generierten Java Code als Vorgabe für eine Reimplementierung zu verwenden. Mein ältester Sohn Stephan unterbrach sein Studium an der LMU München um für uns eine Java Architektur zu konzipieren. Die Architektur entsprach weitgehend der bestehenden IDL Netzwerkstruktur mit vier Architekturebenen und einem Objekt für jeden COBOL Satz. Die Datenfelder sollten dort blei-

ben wie sie waren innerhalb der Objekte mit Get und Set Operationen um sie nach Bedarf zu verarbeiten. Jedes Objekt soll in einer Klasse zusammen mit den Operationen auf das Objekt gekapselt werden. Daneben soll es einen Scheduler geben, der die Ablauffolge der Operationen von außen in einer Steuerungsklasse bestimmt. Stephan stellte sein Konzept vor dem Flughafen IT-Management in November vor und im Dezember bekamen wir einen Auftrag den kompletten COBOL Code in Java zu transformieren und zwar zu einem festen Preis von € 40.000. Eigentlich war es nie beabsichtigt den generierten Java Code in die Produktion zu übernehmen. Er sollte nur dazu dienen den Inhalt des alten Systems in einer Sprache zu dokumentieren die Java Entwickler verstehen können. Die Transformation musste innerhalb vier Monaten abgeschlossen sein. Ich habe mich zusammen mit Stephan, Rudi und Kati gleich rangemacht. Bis Mai 2010 sollten wir fertig sein [Sned09].

12.1.2 Die Objektorientierte Transformationsstrategie

Die Transformationsstrategie sah vor, dass die alten Datenstrukturen soweit wie möglich beizubehalten. Dadurch sollte die Vergleichbarkeit gegeben werden. Wir wollten die Datenergebnisse der neuen Java Programme mit denen der alten COBOL Programme abgleichen. Aus den alten IDL Records, den Arbeitsdatenstrukturen und den Nachrichten sollten Java Klassen erzeugt werden. Die Data Division der COBOL Programme war in fünf Speicherbereiche aufgeteilt – einen Datenbankbereich, eine Daten-Cache, einen Eingabebereich, bzw. Linkage Section, einen Ausgabebereich, bzw. Communication Section und einen Arbeitsbereich, bzw. Working-Storage. Die Prozedur-Division bestand aus mehreren hundert kleinen Codeblöcken, bzw. Absätzen, bestehend aus 3 bis 50 Anweisungen, die auf die Felder in der Data Division beliebig zugriffen. Ein Durchschnittsprogramm hatte circa 5 Kilo Codezeilen, davon 2 bis 3 Kilo Datendefinitionen, von denen die Meisten nie benutzt wurden. In der Regel allerdings verarbeitete ein Codebaustein die Daten von einem einzigen Datensatz. Dies ergab den Anlass die Codebausteine jenen Klassen zuzuordnen die sie am meisten benutzten. Es sollte möglich sein, die Komponente ohne Verlust wertvoller Detailinformationen von Java Spezialisten überarbeiten und optimieren zu lassen. Ein wesentlicher Anteil der Reimplentierungsarbeit war die Entfernung redundanter Klassen und Methoden. D.h. Klassen die in mehreren Komponenten vorkamen, wurden zu einer Klasse zusammengefasst und gleiche Methoden, die in mehreren Komponenten vorgekommen sind in übergeordneten Klassen hochgezogen. Im Grunde genommen lief die Transformation auf ein Refaktoring-Projekt hinaus. Leider waren wir in dieser Angelegenheit nicht konsequent genug gewesen. Zum Schluss gab es doch jede Menge redundanter Klassen, also Klassen die in verschiedenen Programmen, bzw. Paketen, vorkamen. Dies war der Preis der funktionalen Äquivalenz [Sned10a].



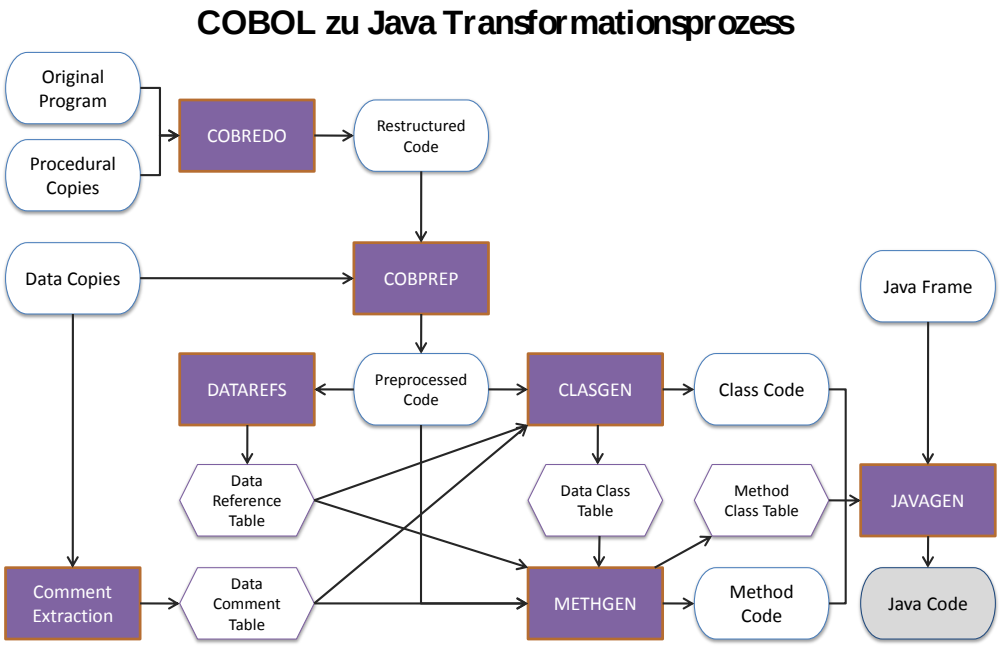
12.1.3 Der Code Transformationsprozess

Der Code Transformationsprozess vollzog sich in sieben aufeinander folgende Schritte. Im ersten Schritt wurde nach dem Motto, erst sanieren, dann migrieren, der COBOL Code von dem Tool COBREDO bereinigt und restrukturiert. Der Zweck der Sanierung war den alten Code auf einen minimalen Qualitätsstand zu bringen. Die Anweisungen wurden gespalten – eine pro Zeile, die If Anweisungen mit End-Ifs abgeschlossen, den verschachtelten Code eingerückt, die Perform Thrus und GoTo Depending Ons eliminiert und festverdrahtete Daten in Tabellen ausgelagert. Außerdem wurden Kommentare, die vor den Absätzen standen, in die Absätzen verlagert damit sie nicht verloren gingen.

Im zweiten Schritt wurde der Source-Code eines jeden Programmes um die Copy Strecken expandiert und die Daten restrukturiert. Einzelne Datenfelder wurden in einer globalen Datenstruktur gesammelt. Sofern sie in den Datenbankschemen vorhanden waren, wurden die kurzen Datennamen durch lange sprechenden Namen er-

gänzt. Das Gleiche galt für die Prozedurnamen. Dafür wurde einen Namensaustauschtabelle von den Flughafen Entwicklern bereitgestellt.

Im dritten Schritt wurde die Datenverwendung analysiert und eine Datenquerverweistabelle erstellt. Darin wurde für jeden COBOL Codeblock seine Ein- und Ausgabendaten festgehalten. Mit Hilfe dieser Tabelle sollten die Codeblöcke als Java Methoden den Klassen zugeordnet werden.



Im vierten Schritt wurde für jede verwendete COBOL Datenstruktur eine Java Klasse erzeugt. Das Objekt war ein Character Array. Jedes Feld war ein Abschnitt dieser Zeichenfolge, identifiziert über sein Startposition, Typ und Länge. Damit wurde das Problem der Daten-Redefinitionen umgangen von denen es in diesem Code besonders viele gab. Für jedes Datenfeld wurde eine „set“ und eine „get“ Methode erzeugt. Für 88 Felder wurde eine Enumerierungsklasse generiert. Für jedes Objekt wurde eine Konstruktor und eine Initialisierungsmethode angelegt. Das Ergebnis war ein Klassenrahmen für jedes Objekt.

Im fünften Schritt wurde für jeden prozeduralen Codeblock eine Java Methode erzeugt. In einem Vorkommentar wurden die Ein- und Ausgabedaten als XML Elemente dokumentiert. Nach jedem Codeblock wurden in einem Nachkommentar die Nachfolgemethoden ebenfalls als XML Elemente dokumentiert. Dazwischen war jede COBOL Anweisung in eine entsprechende Java Anweisung umgesetzt. Die GOTO Anweisungen wurden durch eine Zuweisung der Methodenname zur Label-Variabel mit Return ersetzt. Die wenigen Anweisungen, die nicht automatisch konvertiert wurden, wurden kommentiert. In der Regel wurden über 95% der COBOL Anweisungen vom Tool konvertiert.

Im sechsten Schritt wurden die Methoden mit den Klassen zusammengeführt. Außerdem wurden hier sämtliche Methoden durch ihre Klassennamen qualifiziert damit alle Methodenaufrufe eindeutig waren. Schließlich wurden Container Klassen für die Section Aufrufe erzeugt, in denen Methoden-Aufrufssequenzen enthalten waren.

```
//* CUTE-Geraete haben ihre spez. User-ID
//  IF TXL1 = "U"
   if (IAB6.INPUT.INP_RTX_STORAGE.getTXL1().compareTo("U") == 0) {
//    MOVE TXL1-3 TO USR-IDENT USR-ID R223-LID
            IAB6.WORK.WRK_USR_IDENT.setUSR_IDENT(IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3());
            IAB6.WORK.WRK_IAB6_WORKING.setUSR_ID(IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3());
          IAB6.WORK.WRK_IAB6_WORKING.setR223_LID(IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3());
//    GO XAUT-1
     xNextMethod = " IAB6.WORK.WRK_IAB6_WORKING.IAB6_XAUT_1";
     return xNextMethod;
//  END-IF.
   }
//*  suche LID-Anmeldungssatz   (IIII----)
//  IF TXL1-2 = "SV" OR TXL1-3 = "C@96@F"
if (IAB6.INPUT.INP_RTX_STORAGE.getTXL1_2().compareTo("SV") == 0 ||
   IAB6.INPUT.INP_RTX_STORAGE.getTXL1_3().compareTo("C@96@F") == 0) {
//      MOVE "FD"      TO R223-LID USR-ID
       this.setR223_LID("FD");
//  ELSE
   }
   else {
//      MOVE TXILID    TO R223-LID
       this.setR223_LID(IAB6.INPUT.INP_RTX_STORAGE.getTXILID());
//      MOVE ALL "-"   TO R223-MSID
       this.setR223_MSID("-"[*]);
//      FIND ANY R223AUTY
       DB_STATUS = IDSDB.IDS_R223AUTY.find(R223AUTY, "ANY");
//*   Dieser Satz enthaelt die User-ID
//      IF DB-STATUS NOT ZERO
         if (!(IAB6.WORK.WRK_IAB6_WORKING.getDB_STATUS().matches(Zeros))) {
//          GO 9875F
           xNextMethod = " IAB6.WORK.WRK_IAB6_WORKING.IAB6_9875F";
           return xNextMethod;
//      ELSE
       } else {
//          GET
          DB_STATUS = IDSDB.IDS_R223AUTY.getnext(R223AUTY, this);
```

Im siebten und letzten Schritt wurden die HTML Links zwischen den Daten und Methoden –Datenfluss - sowie zwischen den Methoden und Methoden – Ablauffluss – hergestellt. Damit wurde die JavaDoc Dokumentation vollendet.

12.1.4 Das Transformationsergebnis

Das Ergebnis der Code-Transformation waren 5545 Java Klassen, eine für jede Stufe 1 COBOL Datenstruktur, die von einer gemeinsamen Superklasse COBOLObjekt geerbt haben. Jede Klasse hat eine Konstruktormethode um das singleton Objekt zu Beginn eines Anwendungsfalles zu generieren und eine Initialisierungsmethode um das Objekt mit den Initialwerten zu belegen. Diese stammen aus den Value Klauseln der COBOL Datenvereinbarungen. Das Objekt selbst wurde als „Character Array" definiert. Für jedes benutzte Feld folgen eine Get und eine Set Methoden. Bei alphanumerischen Zeichenfeldern wird ein Objekt vom Typ String erzeugt bzw., in die Character Array abgelegt. Bei numerischen Feldern wird entweder ein Integer oder ein Double Wert erzeugt, bzw. abgelegt.

Auch in jeder Klasse gab es eine Steuerungsmethode die die über die Label-Variabel „XNextMethod" die betroffene Methode invokiert. Diese wurde von der Controllerklasse überreicht. Innerhalb der Verarbeitungsmethoden wurde diese Variable anhand der GOTO und PERFORM Anweisungen gesetzt. Ein Zustandsautomat steuerte den Ablauffluss über den Wert der Label-Variabel. Stephan hat den Controller als recursive Algorithmus implementiert.

Die Mitgliedsklassen bildeten eine Aggregation unter der Kontrollerklasse. Sie waren wie geplant in fünf Speicherbereiche entsprechend den Bereichen im COBOL Programm aufgeteilt:

- Database
- Cache
- Work
- Input
- Output

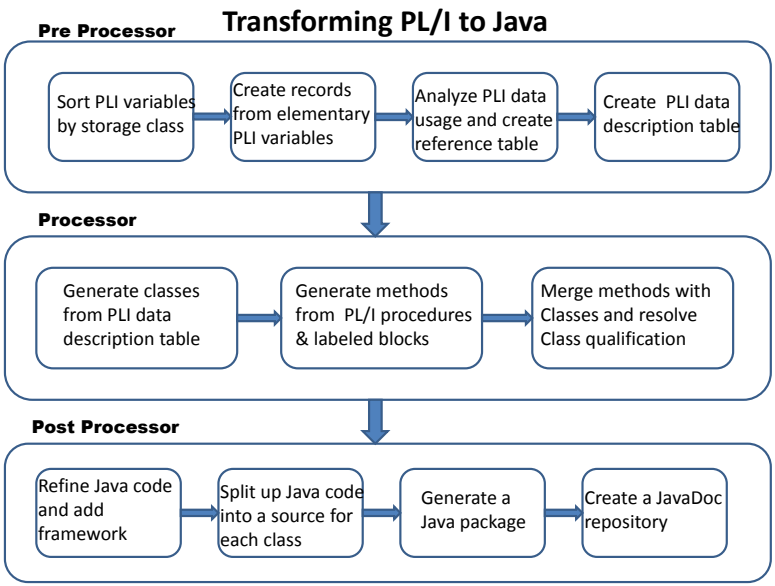
Der Database Bereich enthielt die Sichten auf die Datenbank. Sie wurden von den Verarbeitungsmethoden aufgefordert die Daten aus der Datenbank bereitzustellen. Der Cache Bereich enthielt alle Objekt die aus der Datenbank stammten. Sie wurden dort solange behalten wie sie verarbeitet wurden. Der Input Bereich enthielt die Eingangsnachrichten, die von den diversen Benutzerschnittstellen stammten. Der Output Bereich enthielt die Ausgangsnachrichten, die an die Benutzer zurückgesendet wur-

den. Der Work Bereich enthielt die lokalen Hilfsvariabel sowie die Konstanten – Texte und Nummern, die vom COBOL Programm verwendet wurden. Das Ganze wurde in einer großen Javadoc Bibliothek mit Indexes und Querverweisen dokumentiert.

Dieses Ergebnis mit:

- 2,5 Million Java und Javadoc Codezeilen
- 104 Komponenten
- 4545 Klassen
- 36.750 Methoden
- 814.000 Anweisungen und
- 12.794 Function-Points

wurde Anfang Mai 2010 pünktlich abgegeben. Einige der Komponenten konnten sogar vorher in einer simulierten Umgebung getestet. Wir hatten unseren Auftrag innerhalb der vorgesehenen vier Monate erfüllt. Es sollte aber noch zwei Jahre dauern bis die Reimplementierung abgeschlossen war. Das hat ein Team von der TU-Wien durchgeführt. Über dieses Projekt habe ich auf der internationalen Software Maintenance Konferenz in Temesvar, Romania berichtet. Das Interesse war sehr groß weil seit vielen Jahren sowohl Forscher als auch Praktiker das Problem der Transformation prozeduraler Programme in ein objektorientiertes System zu lösen versuchen. Unsere Lösung war zwar korrekt aber weit entfernt davon optimal zu sein [Sned10b].



Zu dieser Zeit habe ich noch an der Uni Koblenz Reengineering unterrichtet. Eine meiner Studenteninnen war Ellen Wolf, die ihre Diplomarbeit über Softwaremigration schreiben wollte. Wir haben entschieden aus der Diplomarbeit ein Buch zu machen. Die Erfahrung aus dem Flughafenprojekt ist in das Buch „Softwaremigration in der Praxis“ eingeflossen. Das Buch ist im Herbst 2010 erschienen. Meine Mitautorin ist nach ihrem Studium Migrationsberaterin geworden.

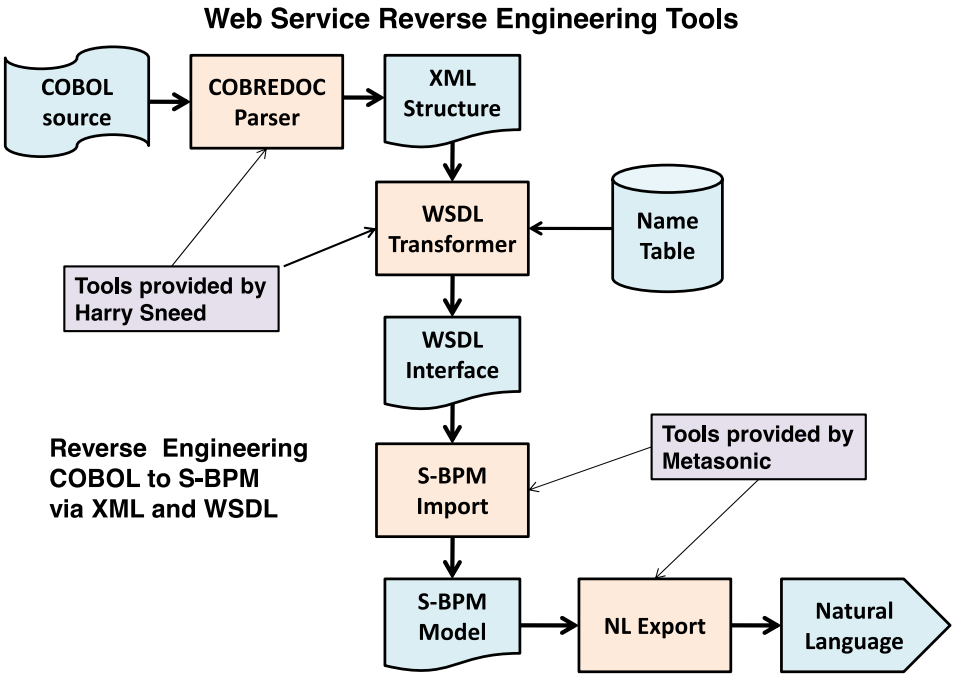
Ich habe die Arbeit an der Migration zu Java fortgesetzt und auch noch ein Tool für PL/I zusammen mit Erdös, Kati entwickelt. Sie hatte früher PL/I in dem SzAMOK Institut gelehrt und kannte sich damit gut aus. Wir haben den gleichen Ansatz verfolgt wie bei COBOL, nur mit anderen Basisklassen für die verschiedenen PL/I Speicherarten. Da es in PL/I keine Paragraphen gibt, haben wir die internen Prozeduren und die DO, bzw. BEGIN Blöcke als Methoden verwendet. Die GoTo Anweisungen haben wir mit einer Label-Variabel ersetzt. Wir sind auch von den statischen Objekten mit festen Adressen für die Attribute abgerückt und haben die Objekte dynamisch generieren lassen, sowie es sich gehört. Das Ergebnis war sogar besser als mit COBOL, es gab nicht so viele kleine Methoden. Ich habe 2011 auf der europäischen Konferenz für Software Maintenance and Reengineering in Oldenbourg darüber berichtet [Sned11].

12.2 Die Umstellung auf Web Services

Schon im Jahre 2000 habe ich begonnen mich mit Web Services zu befassen. Ich war der Überzeugung dass die Anwendungssysteme der Zukunft Service-basiert sein würden. Eine Möglichkeit Services zu erstellen war sie aus dem bestehenden Legacy-Code zu gewinnen. Man sprach von Service-Mining. Ich habe begonnen Tools dafür zu entwickeln – SoftWrap – und Papers darüber zu schreiben. Eine Möglichkeit Wrapping als Migrationsalternative anzuwenden hatte ich bei einem Inkasso Betrieb im Badischen gehabt. Sie war dabei von den Sprachen Forte und COBOL in Java umzusteigen. Die Forte Programme haben sie von einem indischen Dienstleister in Java umsetzen lassen. Die Cobol Programme haben sie gekapselt. Das habe ich zusammen mit dem Team von Frau Nyary für sie bewerkstelligt. Es hat geklappt und wir waren bereits auf einem guten Wege als die Firma verkauft wurde. Daraufhin wurde das Migrationsprojekt gestoppt. Wir sind mit dem Wrapping aller COBOL Programme nie fertig geworden aber wir hatten demonstriert dass es möglich ist, alter Code in eine Java Umgebung einzubinden [Sned05].

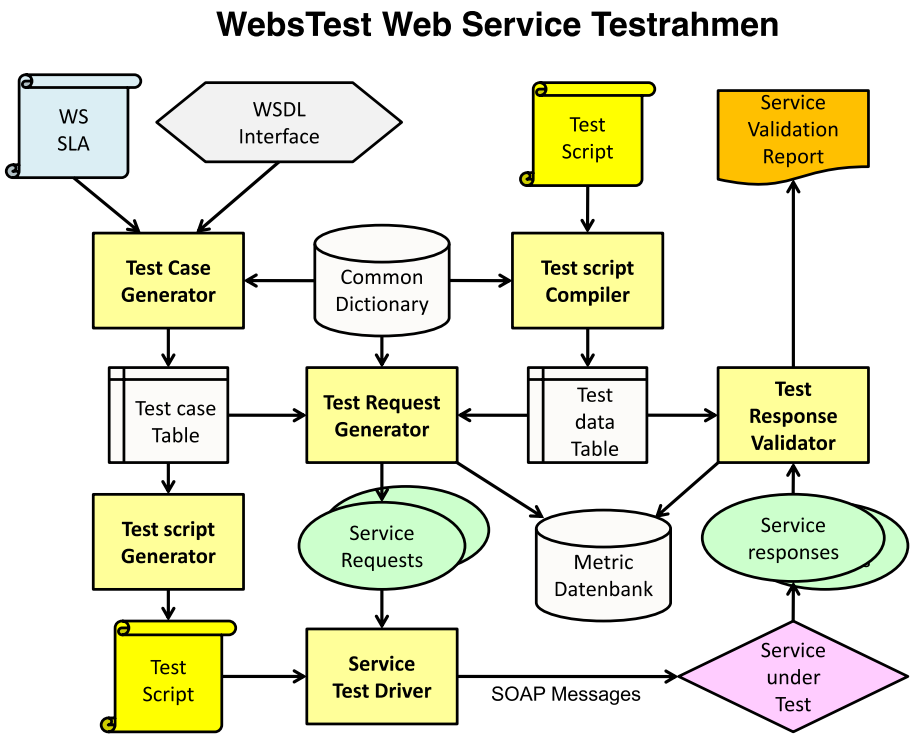
Leider war es nicht einfach solche Projekte in der Industrie zu bekommen und wenn, waren es meistens nur Probeprojekte wie das was ich von einer Versicherungs-

firma in Deutschland bekommen habe. Das Projekt sollte zeigen ob es möglich ist aus alten COBOL Programmen Web Services automatisch abzuleiten. Der Kunde wollte diesen Ansatz als Alternative zu einer komplett neuen Entwicklung erproben. Ich hatte bereits die Werkzeuge dafür – SoftWrap, und müsste sie nur an der besonderen IMS-DB/DC Umgebung der Kundenprogramme anpassen. Ich musste sämtliche Bildschirmoperationen durch Serviceaufrufe ersetzen. Bis auf einige geringe Syntax-Fehler ist dies mir mit Hilfe von Majnar, Rudi gelungen. Fünf ausgewählte IMS/COBOL Programme wurden automatisch gekapselt. Aber dann folgte der Test der gekapselten Programme. Dies hat sich als äußerst aufwendig erwiesen, da wir die alten Bildschirmeingaben nicht wiederverwenden konnten. Wir müssten neue Testdaten für eine Web-Service Schnittstelle schaffen. Der Test der widergewonnen Services war so aufwendig, das der Anwender das Vorhaben aufgab. Daraufhin habe ich mich entschlossen ein Tool für den Test von Web Services zu entwickeln [Sned09].



Das Thema Service-orientierte Architektur lies mich trotz der ersten Rückschläge nicht los. Die neuen Tools – neben SoftWrap gab es auch ein SoftReus – haben alle darauf gezielt aus bestehenden Legacy-Code Web Services herauszuschneiden und in

einer Standard Web Service Bibliothek bereitzustellen. Ich begann mit COBOL und PL/I und habe den Ansatz auch auf C++, C# und Java ausgedehnt. Ich habe sogar meinen Sohn Stephan in diese Arbeit einbezogen um die Verbindung zu den geplanten Geschäftsprozessen herzustellen. Im September, 2012 haben wir unseren Ansatz auf der internationalen Maintenance Konferenz in RivadelGarde vorgestellt [SnSn12]. Ein Jahr später setzte ich das Thema auf derselben Konferenz in Eindhoven fort. Diesmal habe ich meinen langjährigen Forschungspartner und künftigen Doktorvater Chris Verhoef von der Freien Universität Amsterdam einbezogen. Unser Beitrag hieß „Migrating to Service-oriented Systems“ [SnVe13]. Ich war eben besessen Legacy Code als Quelle für die neuen Services zu gewinnen. Leider bekam ich dafür keine Unterstützung von meiner damaligen Arbeitgeber ANECON und ich selbst hatte nicht mehr die nötigen Kontakte zu den Anwendern in Deutschland. Meine Zeit als Industrieberater in Deutschland war vorbei und in Österreich waren die Anwender nicht so weit. Also blieb ich auf meiner Forschung sitzen.



12.3 Mein letztes Projekt für die ANECON

Die Schätzung der Wartungskosten für das Planungs-Produktions-und-Steuerungssystem in der Automotivindustrie im Jahre 2012 war mein vorletztes Messprojekt für die ANECON. Es folgte im Jahre 2013 nur noch zwei Migrationsplanungsprojekte –eins für einen AS400 Anwender das ich mit Erdös, Kati in einer Industriefirma in der Nähe von Linz durchgeführt habe [SnEr13] und eins für ein Wiener Finanzinstitut, das gleiche Institut mit dem ich im Jahre 1978 meine Karriere als Tester in Wien begonnenen hatte. Inzwischen war viel Wasser der Donau runtergeflossen. Jetzt stand der IT-Dienstleiter dieses Unternehmens vor der Notwendigkeit ein altes Assembler-System von damals zu modernisieren.

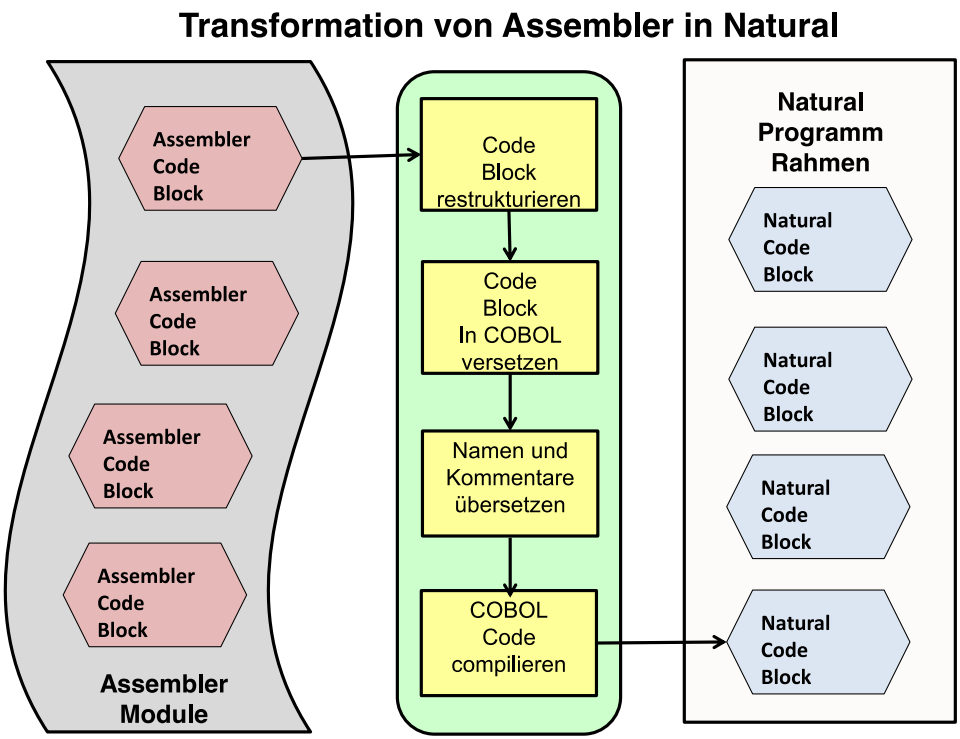
Das System um das es sich handelt war wie viele alte Systeme dieser Art Ende der 70er Jahre in Assembler aus Gründen der Performanz implementiert wurden. Es ging um die Offline Buchung von Kontenbewegungen. Die Bewegungen wurden während des Tages Online erfasst und nachts im Batchbetrieb gegen die Kontensätze gebucht. Die ursprünglichen Online-Transaktionen waren mit COBOL-CICS implementiert. Später in den 90er Jahren wurden die CICS Online-Programme durch Natural-Adabas Applikationen abgelöst, aber die Assembler Batchlösungen blieben. Erst nach vielen Jahren wurde entschieden die Assembler Teile des Gesamtsystems auch abzulösen. Da der Rest des Systems weiterhin aus Kostengründen in Natural bleiben sollte, sollten auch die Assembler Teile in Natural umgesetzt werden.

Es gab mehrere mögliche Wege zu diesem Ziel. Ein Weg war die vollständige Neuentwicklung auf der Basis eines neuen Fachkonzeptes. Dieser Weg schien dem Anwender aber zu teuer und zu risikoreich. Ein anderer Weg war die automatische Nachdokumentation und die manuelle Reimplementierung des Codes in der neuen Sprache. Gegen diesen Weg sprach die Unzulänglichkeit des Assembler-Reverse-Engineering. Es ist sehr schwer brauchbare Information aus Assembler-Code abzuleiten. Ein dritter Weg war die manuelle Dokumentation des Systems verbunden mit dem manuellen Umschreiben des Codes. Ein vierter Weg war das einfache, 1:1 manuelle Umsetzung des Codes ohne Dokumentation durch einen Offshore Partner. Dieser Weg ist letztendlich begangen worden, aber erst nachdem versucht wurde den Code automatisch umzusetzen.

12.3.1 Automatische Transformation des Assembler Codes

Da die Lücke zwischen Assembler und der Natural-II Sprache besonders breit ist – Natural II ist eine strukturierte Sprache ohne GOTO Anweisung, die Assemblerprogrammsteuerung basierte hingegen auf GOTO Sprüngen- habe ich entschieden den Assembler Code erst in COBOL umzusetzen, den COBOL Code zu restrukturieren und anschließend die COBOL Programme in Natural zu transformieren. Die Sprache

COBOL sollte dazu dienen dem Code eine Struktur zu verleihen und in kleinere, handhabbare Codeblöcke zu zerlegen. Gleichzeitig sollten die Assembler Kurznamen in sprechende Langnamen verwandelt werden um die Lesbarkeit des Codes zu erhöhen.



Die Transformation des Assembler Codes in COBOL vollzog sich in drei Schritten mit drei verschiedenen Werkzeugen. Im ersten Schritt wurde mit dem Tool *AsmRedo* der Assembler Code restrukturiert. Die Restrukturierung unstrukturierten Assembler-Codes setzt mehrere Pässe durch den Code voraus. Zunächst müssen alle Sprungbefehle und alle Ansprungs-Adressen in einer Tabelle gesammelt werden. Daraus wird ein abstrakter Syntaxbaum – AST – gebildet.

Im zweiten Schritt wurde mit dem Tool *AsmTrans* der umstrukturierte Assembler Code in unstrukturierte COBOL umgesetzt. Dabei wurde die Syntax verändert.

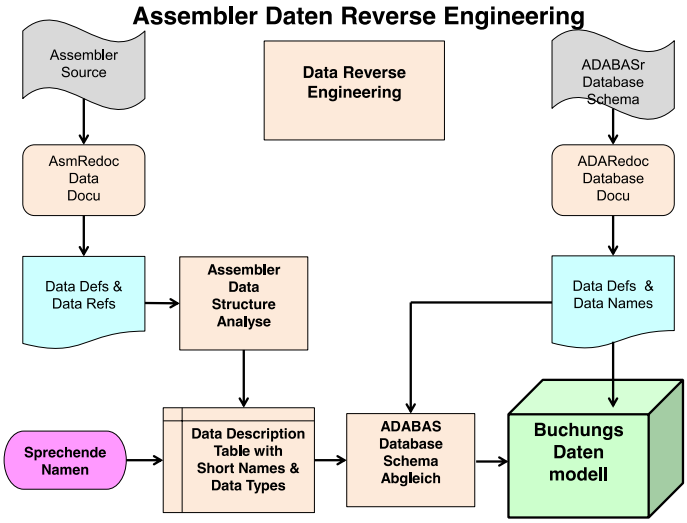
In dem dritten Schritt wurden mit dem Tool COBRedo der COBOL Code restrukturiert, die Literale durch Variable ersetzt und die Datennamen, soweit bekannt, umgetauscht.

Die Assembler Register wurden als Arbeitsfelder in der Data Division des COBOL Programmes deklariert und zwar sowohl als Binärnummer als auch als Zeichenfolge, damit sie als Strings und als Zahlen behandelt werden konnten. Die Konstanten, die in den prozeduralen Anweisungen vorkamen wurden als symbolische Konstanten mit festen Werten vereinbart. Die anderen einzelnen Daten wurden 1:1 als Stufe 1 Arbeitsfelder in die Data Division übernommen.

Die als Datensätze erkannten Datenstrukturen wurden wiederum in COPY Members zusammengelegt, wo sie von allen Programmen gemeinsam benutzt werden konnten. Diese Sätze hatten eine dreistufige Struktur.

12.3.2 Aufbau einer Assembler/Natural Data Dictionary

Ein besonderes Problem bei der Konvertierung vom Assembler Code ist die Benennung der Daten. Assembler Variabel haben nur maximal 8-stellige Namen, aber oft sind längere Namen in Kommentaren hinter der Datendefinition. In einem Nebenprojekt wurden alle Datennamen gesammelt zusammen mit ihren Kommentaren und in eine Datentabelle abgelegt. Diese Tabelle wurde anschließend verwendet um bei der Umsetzung des Codes die Kurznamen durch Langnamen auszutauschen. Der ursprüngliche Kurzname wurde als Präfix zu dem neuen langen Namen beibehalten um die Verbindung zu dem Assembler Code zu erhalten. Auf dieser Weise wurden von Frau Erdös mehr als 2000 Datentypen in einer Repository gesammelt und umbenannt.



12.3.3 Probleme mit der Adressierung

Die zwei größten Stolpersteine in diesem Projekt waren die dynamisch veränderbaren Anweisungen und die überlagerten Datenstrukturen. Dynamisch veränderbare Anweisungen sind Sprungbefehle deren Ansprungsadresse zur Laufzeit überschrieben wird. Solche Befehle müssten manuell umgeschrieben werden, da eine automatisierte Lösung nicht gefunden werden konnte. Diese Befehle machen zwar weniger als 2% des Codes aus, aber das ist genug um eine vollständig automatisierte Transformation zu verhindern. Überlagerte Datenstrukturen sind mehrere logische Datengruppen die den gleichen physischen Speicherbereich besetzen. Die Entwickler hatten begonnen saubere Datenstrukturen als DSects zu definieren, aber im Laufe der Entwicklung sind sie dazu übergegangen physische Anfangszeiger zu verwenden. Statt über ihre Namen wurden die Daten über ihre Distanz von dem Anfangszeiger adressiert. D.h statt Datennamen waren in dem prozeduralen Code nur Zeiger, Absetzungsdistanz und Feldlänge zu finden z.B. KUENDIGUNG(3:8). Daran ist dieser Ansatz letztendlich gescheitert. In dem Moment in dem händisch in den Code eingegriffen werden muss, gehen die Kostenvorteile der Automation verloren. Man braucht einen Entwickler der sowohl die Quellsprache als auch die Zielsprache versteht.

12.3.4 Die Grenze der Automation

In Anbetracht der physischen Addressierung und der vielen Ablaufverzweigungen wäre es nicht möglich gewesen den Code automatisch in Natural umzusetzen, vor

allem deshalb weil Natural keine GOTO Anweisung kennt. Der einzige Weg war den Code erst in COBOL umzusetzen und anschließend den COBOL Code in Natural zu übersetzen. Mit Majnar, Rudi habe ich ein Pilot Projekt mit zwei Moduln gemacht. Davon wollte aber der Kundenprojektleiter nichts wissen. Er wollte die Programme direkt in Natural umschreiben. Wir haben versucht Natural Code zu generieren aber er war voller Lücken. Die automatische Umsetzung dauerte nur Minuten aber der Nachbearbeitungsaufwand für die zwei Pilotmodule betrug 12 Personentage = € 8.640. Das war pro Modul € 4.320, bzw. € 1,5 pro Zeile. Die Inder baten an, den Code für € 1 pro Zeile manuell zu konvertieren. Das machte in diesem Falle € 2.828 pro Modul. Damit waren wir aus dem Rennen. Wie damals in der Schweiz hatte ich trotz eines großen Aufwandes meinerseits wieder gegen die Inder verloren [Sned14].

12.3.5 Das Ende meiner Arbeit für die ANECON

Mein Versagen in diesem Projekt hat mein Schicksal in der ANECON besiegelt. Die Geschäftsführer haben erkannt dass ich nicht mehr zu gebrauchen war. Sie hätten das Projekt gerne als Entwicklungsprojekt gewonnen. Mit meinem technisch motivierten Beharren auf einen Umweg über COBOL hatte ich den Kundenprojektmanager verärgert, dann kam mein Versagen bei der automatischen Umsetzung. Das die Kosten für die manuelle Umschreibung so hoch waren, lag nicht an mir aber ich hatte sie zu verantworten. Hinzu kam, dass ich zu Beginn des Jahres schon einmal versagt habe. Ein mittelständisches Unternehmen in der Nähe von Linz hatte einen Cobol Entwickler für den AS400 gesucht. Da ich keine andere Arbeit hatte, habe ich mich dafür gemeldet. Das war ein Fehler. Ich hatte seit dem Wella Projekt im Jahre 1995 nichts mehr mit der AS400 zu tun gehabt. Ich brauchte Zeit um mich wieder einzuarbeiten. Es gab aber keine Zeit dafür. Der Kunde wollte sofort ein neues Programm haben und zwar innerhalb einer Woche. Als ich das nicht schaffen konnte, hat der Entwicklungsleiter mich prompt vor der Tür gesetzt. Ich war dort nicht zu gebrauchen.

So kommt es, dass jeder Mensch wenn er nicht rechtzeitig aussteigt irgendwann mal seine Nützlichkeit überlebt. Für die ANECON war dieser Punkt bei mir schon erreicht. Wie man im englischen sagt „I had reached the end of the line“. Die Leitung brauchte nur einen Anlass mich weg zu schicken. Diesen Anlass habe ich ihnen im darauf folgenden Jahr geliefert.

12.4 Zwischen drei Städten

In den letzten Jahren bei der ANECON bin ich immer nur hin und her zwischen den drei Städten – München, Wien und Budapest – gependelt. In Budapest habe ich die Werkzeuge mit Hilfe meiner ungarischen Mitarbeitern weiterentwickelt. In Wien

habe ANECON Kunden beraten und die Werkzeuge eingesetzt um diverse Aufgaben zu erledigen – prüfen, messen, dokumentieren, sanieren und konvertieren. In Bayern war ich nur gelegentlich zuhause. An der Uni Regensburg habe ich jeden Freitag Software Engineering für Wirtschaftsinformatiker gelehrt. In Hagenberg bei Linz habe ich dort auf der Fachhochschule Kurse für Software-Produktmanagement und Evolution gehalten. Später fing ich an abends unter der Woche Softwaremessung und Testautomation an der Fachhochschule Wien zu unterrichten. Noch zwei Bücher „Software in Zahlen“ im Hanser Verlag und „Softwareevolution“ in dpunkt Verlag habe ich in dieser Zeit mit der Unterstützung von Richard Seidl herausgebracht. An zwei weiteren Büchern „Der Integrationstest“ und „Agile Testing “ habe ich mitgewirkt. Einen Großteil meiner Zeit habe ich auf der Bahn zwischen den Orten an der Donau verbracht. Für mich wurden die Projekte immer seltener. Ich war inzwischen über 70 und als Tester kaum noch zu gebrauchen. Was blieb war die Lehre und die Softwareanalyse.

Nach dem letzten erfolglosen Versuch das Assembler System zu konvertieren waren meine Tage bei der ANECON gezählt. Das war im Oktober 2013. Ein Monat später habe ich mein 10 jähriges Jubiläum bei der Firma gefeiert. Mein Abteilungsleiter bedankte sich für meinen langjährigen Beitrag und schenkte mir zwei Flaschen Wein. Das wäre der richtige Augenblick mich zu verabschieden, aber mir fehlte der Mut dazu. Ich hatte Angst vor dem was auf mich zukommen würde. In Wien hatte ich ein Zimmer bei einem alten Ehepaar am Rande des Wienerwaldes oberhalb von Neuwaldegg. Von dort aus konnte ich im Wienerwald herum wandern. Es war mühselig dort zu Fuß von der letzten Bushaltestelle hinaufzugehen, aber es tat mir gut. Ich hatte mich an dem Wanderleben zwischen drei Wohnorten in drei verschiedenen Ländern gewöhnt und wollte damit nicht aufhören.

Neben meiner Tätigkeit bei der ANECON hatte ich noch die Kurse an den Fachhochschulen in Österreich und meinen Lehrauftrag an der Uni Regensburg. Dazu kam ein alljährliches Seminar für Doktoranden an der Universität Szeged. In Regensburg war die Zahl meiner Zuhörer im Laufe der Jahre immer weniger geworden, seit dem ich in Englisch vortragen musste. Es waren meistens ausländische Studenten. Mehr Zuhörer hatte ich an den Fachhochschulen in Wien und Hagenberg. Im Frühjahr 2013 bekam ich einen Auftrag Software Reengineering an der TU Dresden zu lehren. Der Kurs dauerte vier Wochen mit drei Tagen in der Woche. Es war zwar weit ab von meinem gewöhnlichen Wege aber ich konnte Dresden mit dem Nachtzug aus Budapest bzw. aus Wien über Prag erreichen. In Dresden konnte ich auch meine Forschungsarbeit fortsetzen. Zusammen mit meiner Chefin dort – die Frau Dr. Birgit Demuth - habe ich einige wissenschaftliche Beiträge verfasst. Dresden war bemüht

Elite Universität zu bleiben und es zählte möglichst viele Veröffentlichungen herauszubringen. Ich konnte dazu beitragen.

Ich hatte also genug zu tun gehabt auch ohne die Arbeit für ANECON. Dennoch war für mich der Kontakt mit der industriellen Praxis sehr wichtig. Er war für mich die Quelle meiner Inspirationen. Die Lehranstalten waren für mich eine andere Welt, eine Scheinwelt fern ab von der Realität des wirtschaftlichen Lebens. Dort hat man die Möglichkeit sich selber Scheinaufgaben stellen und für diese Scheinaufgaben Scheinlösungen ausarbeiten. Man steuert die Welt in der man arbeitet. Die meisten Hochschulprofessoren, auch wenn sie auf ihrem Gebiet hervorragende Ansätze hervorbringen, leiden darunter dass sie nicht unter Zugzwang stehen die Ansätze in echten Projekten durchzuziehen. Somit können die Ansätze in der Praxis nie reifen. Sie bleiben theoretische Modelle. In der Welt der Wirtschaft ist das anders. Man wird von der Welt in der man arbeitet gesteuert. Es ist selten so wie man es sich wünscht, es geht mehr um die Macht als um die Sache, aber es ist die Realität. Die Projektwelt ist chaotisch, brutal, unwissenschaftlich und ungerecht. Dieses Buch ist der beste Beweis dafür. Trotzdem wollte ich mit einem Fuß drin bleiben.

Das sollte nicht lange halten. Nach dem letzten erfolglosen Projekt habe ich nur noch unterrichten können. ANECON hatte begonnen, wie viele andere Testberatungshäuser Kurse für die Zertifizierung von Testern anzubieten. Dieses Zertifizierungsprogramm ist an sich eine gute Sache. Es ist wichtig, dass in einem Tätigkeitsbereich wie Testen alle dieselbe Begriffe verwenden und dieselbe Ansätze anwenden. Es ist aber falsch zu denken, bloß weil einer die Texte aus dem Syllabus auswendig gelernt hat, ist er zum Testen befähigt. Es gehört mehr dazu. Das hat mich gestört als ich von der ANECON beauftragt wurde Tester auszubilden.

12.5 Mein Abgang von der ANECON

Nach dem misslungen Versuch wieder in ein Projekt hineinzukommen, durfte ich bei der ANECON nur noch unterrichten, nach dem Motto „if you can't do it, teach it". Meine Aufgabe bestand darin, die Grundlagen des Testens zu lehren so wie sie in den Textbücher geschildert sind. Ich musste endlose Aufzählungen durchgehen und Begriffe erklären, die jeder angehende Tester kennen sollte, aber wofür der Frontalunterricht nicht gerade die beste Methode ist sie beizubringen. Hierfür wären Selbstlernprogramme besser geeignet, aber die Kundenfirmen haben für die Frontalunterricht bezahlt. Also habe ich versucht die Unterricht mit Geschichten aufzulockern. In einer dieser Geschichten ging es um den Bau einer Chip Fabrik in meinem Heimatland Mississippi. Die Geschichte sollte zeigen, dass es zu einem erfolgreichen Projekt die

richtigen Menschen erforderlich sind. Prozesse können Menschen nicht ersetzen. In den 80er Jahren wollte Intel dort versuchen Chips herzustellen. Es gab dort viele arme Menschen vor allem Schwarze und Intel bekam für das Projekt eine Förderung von der Bundesregierung. Es hat sich aber bald herausgestellt, dass die Menschen dort für diese Art Arbeit nicht geeignet waren. Die Fingerfertigkeit und die Konzentrationskraft waren nicht ausreichend. Bis dahin hatten hauptsächlich asiatische Frauen in Kalifornien diese Art Arbeit verrichtet. Intel hatte in Mississippi das Projekt nach kurzer Zeit aufgegeben. Ich wollte damit sagen, dass die erste Voraussetzung für ein erfolgreiches Projekt ist die geeigneten Menschen zu finden. Bis dahin waren die Schwarzen in Mississippi hauptsächlich mit Baumwollpflücken beschäftigt. Der Schritt von „cotton balls“ zu „integrated circuits“ war eben zu groß. Die gleiche Situation finden wir in vielen IT Projekten. Mancher Anwenderbetrieb versucht ein Projekt anzupacken mit Mitarbeitern, die noch nicht soweit sind. Sie haben weder die Kenntnisse noch die Erfahrung die erforderliche Arbeit zu verrichten. Dann nutzen alle Prozesse, Methoden und Werkzeuge nichts. Die Menschen müssen in der Lage sein sie richtig einzusetzen.

Deshalb ist hohe Qualität in der Herstellung von Hardware und Software das Ergebnis eines langjährigen Reifeprozesses. Die beteiligten Mitarbeiter brauchen eine Kombination von Kenntnissen, Fähigkeiten und Erfahrungen hochwertige Produkte herzustellen. Die Deutschen sind stolz auf ihre Marke „Made in Germany“ weil die Menschen hier die Voraussetzungen erfüllen. Sie haben sie über Generationen einer Industriegesellschaft erworben. In Mississippi fehlen diese Voraussetzungen. Da Mississippi schon immer eine reine Agrarwirtschaft war, ist sie nicht gerade der beste Platz um Hightech Betriebe auf die Schnelle einzurichten. Solche Fähigkeiten die in einer industriellen Gesellschaft wie Deutschland vorhanden sind, brauchen Generationen heranzureifen. Das heißt noch lange nicht, dass die Deutschen die besseren Menschen sind. Es heißt, dass sie dem Prozess der Industrialisierung viel länger ausgesetzt sind. Ein komplexer Produktionsprozess mit anspruchsvollen Produktionsmitteln setzt hoch qualifizierten Produzenten voraus. Das gleiche gilt für Entwicklungsprozesse in der IT-Industrie. Ohne qualifizierte Mitarbeiter kann es keine Qualität geben, egal welche Prozesse und Tools man einsetzt.

Einige Kursteilnehmer haben mein Beispiel übelgenommen. Die Zeiten hatten sich geändert: früher konnte ich diese Geschichte ohne weiteres erzählen, keine fand Anstoß daran. Jetzt aber, im Zuge der politischen Korrektheit und der zunehmenden Empfindlichkeiten klang sie rassistisch. Nach Ansicht einer Teilnehmerin wollte ich damit sagen: die Schwarzen gehören in den Baumwollfeldern, die Weißen in den Büros und Fabriken. Abgesehen davon, dass die Weißen in Mississippi auch nicht

viel fähiger sind, auch sie sind in dieser High-Tech Welt zurückgeblieben, hat diese Geschichte vielleicht doch einen rassistischen Unterton. Ich hätte sie in dieser Form nicht erzählen dürfen.

Als die Teilnehmer sich bei der Bewertung des Kurses beschwert hatten, blieb der ANECON, die Wert darauf legte als politisch korrektes Unternehmen zu erscheinen, nichts Anderes übrig als mich zu entlassen. Ich hatte die Grenze des Anstands überschritten.

Die Entlassung aus der ANECON habe ich widerstandslos hingenommen. Ich hatte sowieso keinen Nutzen mehr für sie und in Vergleich zu aller früheren Entlassungen war dieser wegen Rassismus am ehesten gerechtfertigt. Präsident Obama hatte gerade in einer Ansprache auf Anlass einer Beerdigung von schwarzen Bürgern die in einer Kirche in South Carolina erschossen wurden geäußert, die weißen Südstaatler würden Rassismus in ihren Genen haben. Es würde Generationen dauern um diese Krankheit auszutreiben. Als weißer Südstaatler gehöre ich eben dazu. Man kann seine Wurzeln nicht verleugnen.

12.6 Wohin mit den älteren Softwareentwicklern

Es gab früher ein Lied über alte Soldaten „old soldiers never die, they just fade away“. Das Gleiche gilt für alte Softwareentwickler. Die wenigsten Menschen werden in der IT-Welt alt. In den 45 Jahren, die ich hier geschildert habe, sah ich viele Menschen in der IT-Welt kommen und gehen. Wer als Fachkraft bleibt, wird technologisch bald überholt. Er verbringt seine letzten Jahre in der IT mit der Erhaltung jener Systeme, die er früher mal entwickelt hat. Solange diese Systeme leben, hat man bei der Firma eine Daseinsberechtigung. Wenn diese Systeme ausgemustert werden, werden die Mitarbeiter, die sie jahrelang gepflegt haben, überflüssig. Die Chance, ein neues Projekt zu bekommen ist gering, weil die Kenntnisse für das alte System nicht mehr verlangt werden. Viele Entwickler der alten Systeme werden zum Tester der neuen Systeme. Wenn sie das schaffen, können sie noch eine Weile als Fachkraft bestehen. Ansonsten geht man in die Fachabteilung und spezifiziert die Anforderungen für die nächsten Systeme. Es besteht zwar die Möglichkeit eine Technische Führungskraft zu werden, aber das können nur die wenigsten. Es gibt schon jetzt in den IT-Abteilungen zu viele Manager. Sie werden nicht benötigt und stehen den anderen nur im Wege. Im Zuge der agilen Selbstorganisation werden immer weniger Manager gebraucht.

Eine weitere Option für die älteren Mitarbeiter ist die Projektunterstützungsarbeit. Es gibt ein Bedarf an Sachbearbeitern für administrative Aufgaben um die operativen Tätigkeiten herum, Aufgaben wie Reviews der Dokumente, Prüfung und Messung der Zwischenergebnisse, Verfolgung und Aufzeichnung des Projektfortschrittes, Erstellung von Testdaten und Kontrolle der Testergebnisse. Sogar in der agilen Entwicklung gibt es neben dem Tester im Team auch der Projekt-Tracker. Der Projekt-Tracker führt die Fehlerstatistik und verwaltet den Backlog. Dies ist eine ideale Arbeit für einen älteren Mitarbeiter. Für solche administrative Tätigkeiten brauchen ältere Mitarbeiter nicht die neuesten technischen Kenntnisse über Cloud Services und Big Data.

Die eine Frage ist, ob ältere Mitarbeiter in der Softwareentwicklung bereit sind diese Hilfstätigkeiten zu übernehmen. Sie würden weniger verdienen als vollwertige Entwickler, aber sie hätten einen Job. Die Alternative dazu wäre wie bereits erwähnt als Systemanalytiker, Prozessmodellierer oder fachlicher Tester in der Fachabteilung zu arbeiten. Jedenfalls gebe es genug zu tun, um die Qualität der Softwareprodukte zu sichern.

Die andere Frage ist ob die Betriebe – Anwender oder Hersteller von Software – bereit sind diese zusätzlichen Kosten für die Qualitätssicherung zu tragen. Wer die Qualitätssicherung ernst nimmt, müsste dafür mindestens so viel ausgeben, wie für die Entwicklung und Wartung. In der Vergangenheit waren die wenigsten IT Anwender dazu bereit. So wollten eher noch mehr Systeme schaffen, statt die vorhandenen besser zu machen, d.h. neue Funktionalität und schnelle Erfolge hatten Vorrang vor Qualität und Nachhaltigkeit. Hier ist ein großer Nachholbedarf. Die alten Systeme müssten saniert und modernisiert werden. Das erfordert eine signifikante Investition. Dafür würden solche Mitarbeiter benötigt, die die alten Technologien noch beherrschen. Es gebe demnach Grund genug die älteren Mitarbeiter weiter zu beschäftigen.

Das dies nicht geschieht oder nicht im ausreichenden Maße, liegt zum Teil an der Kurzsichtigkeit des IT-Managements. Es erkennt nicht den potentiellen Nutzen der älteren Mitarbeiter. Was alt ist, ist relativ, aber es kann schon mit 40 beginnen wenn die Mitarbeiter nicht mit den neuen Technologien mitkommen. Zum anderen Teil liegt es an älteren Mitarbeitern selbst wenn sie nicht gewillt sind sich den jüngeren Kollegen unterzuordnen und eine minderwertigere Tätigkeit auszuführen. Diese müssten erkennen dass ihre Erfahrung, Kenntnisse und Fähigkeiten nur einen begrenzten Wert haben. Erfahrung in der technischen Welt hat nur einen Wert wenn sie einen Bezug zur verwendeten Technik hat. Das müssen die Älteren einsehen. Es ist doch besser eine geringer bezahlte Arbeit zu haben als gar keinen.

Wenn die Qualitätssicherung billiger wird, wird sie auch intensiver betrieben und alle profitieren davon. Es liegt an den Betrieben die Voraussetzungen dafür zu schaffen.

12.7 Fünfzig Jahre Software Engineering

Software Engineering wurde als Begriff auf der NATO Konferenz in Garmisch im Jahre 1968 erstmals geprägt, zwei Jahre vor dem Beginn meiner Zeit als Softwareentwickler in Deutschland. Prof. Fritz Bauer von der TU München meinte, wir würden ein Engineering Disziplin brauchen, um den Chaos in der Software Entwicklung Herr zu werden. Dieser Anstoß aus der deutschen akademischen Welt wurde von der IT-Industrie - damals noch DV-Industrie – mit Begeisterung aufgenommen. Es entstand eine neue Lehrdisziplin an den Hochschulen und ein neuer Berufszweig in der Industrie [Broy06]. Die Bezeichnung Software Engineer setzte sich durch lange ehe es ein international anerkannter Studienweg zu diesem Berufstitel gab. Ich selbst habe Software Engineering propagiert und gelehrt obwohl ich sie nie studiert habe. Deshalb habe ich mich niemals aus Respekt vor dieser Disziplin als Software Engineer bezeichnet. Ich war Programmierer, Analytiker, Tester und gelegentlich Manager. Heute gibt es Studiengänge für dieses Fach und man kann ein Titel als Master of Software Engineering erwerben. Was genau zu diesem Fach gehört, wird überall etwas anders ausgelegt, aber im Großen und Ganzen herrscht Einigkeit über das was ein Software Engineer zu beherrschen hat. Das Software Engineering Programm der Carnegie Mellon Universität in Pittsburgh, U.S.A. dient als Vorbild. Dort ist auch das amerikanisches Software Engineering Institut situiert, wo die Normen für Software Engineering Prozesse in der U.S. Industrie entstanden sind [Glas02].

Vierzig Jahre nach der Entstehung dieser Disziplin veröffentlichte der weltbekannte IT-Autor und Dozent – Tom DeMarco einen Artikel in der IEEE Software Magazine mit dem provokativen Titel

„Software Engineering – An Idea whose Time has Come and Gone?“

In diesem umstrittenen Beitrag behauptete DeMarco, das Software Engineering nicht mehr gültig ist, denn Engineering setzt Kontrolle voraus und moderne Software Projekte lassen sich nicht kontrollieren. In Anspielung auf die agile Entwicklung meint er heute, Softwareentwicklungsprojekte sind eigentlich Forschungsprojekte und als solche nicht im Sinne von Engineering beherrschbar. Er setzt damit Software Engineering mit systematischer Prozesskontrolle bzw. mit Projektmanagement gleich. Wenn es nicht möglich ist ein Software Projekt exakt zu planen und zu steuern so wie dies in einem konventionellen industriellen Fertigungsprozess der Fall ist, kann von Engineering keine Rede sein. Also mit Software Engineering reiten wir auf einem toten Pferd. [DeMa09]

DeMarco endet seinen Diskurs mit der Feststellung „I am gradually coming to the conclusion that software engineering is an idea whose time has come and gone. I still believe it makes excellent sense to engineer software. But that is not what software engineering has come to mean. The term encompasses a specific set of disciplines including defined processes, inspections, walkthroughs, requirement engineering, traceability, metrics, walkthroughs, requirement engineering, traceability, metrics, precise quality control, rigorous planning and tracking, coding and documentation standards. All of these strive for consisting of practice and predictability…”

Lisbon, 29 November 2013

2013 International Software Testing Excellence Award

Dear Mr Sneed,

On behalf of the ISTQB® (International Software Testing Qualifications Board) we are delighted to present you with the International Software Testing Excellence Award for 2013 in acknowledgement of your valuable contribution to the profession.

You have been chosen from a large group of highly qualified applicants through a rigorous selection process of balloting and discussion by a panel of ISTQB® software testing experts. The panel feels that you have made a significant and lasting contribution to the software testing profession at a level that is worthy of this award over the course of a long and productive career.

Therefore, it is with the utmost respect and appreciation that the Executive Committee of the ISTQB® and the ISTQB® Award Panel thank you for your contribution to our profession and congratulate you on winning the Award.

Kind regards,

Chris Carter
ISTQB® President

Gualtiero Bazzana
Head of ISTQB® Marketing Working Group

About ISTQB®

The ISTQB® (International Software Testing Qualifications Board) is a non-profit organization, headquartered in Brussels, Belgium, whose mission is to advance the software testing profession by managing the most successful tester certification program in the world. It is supported in its operations by 46 member boards, representing over 70 countries.

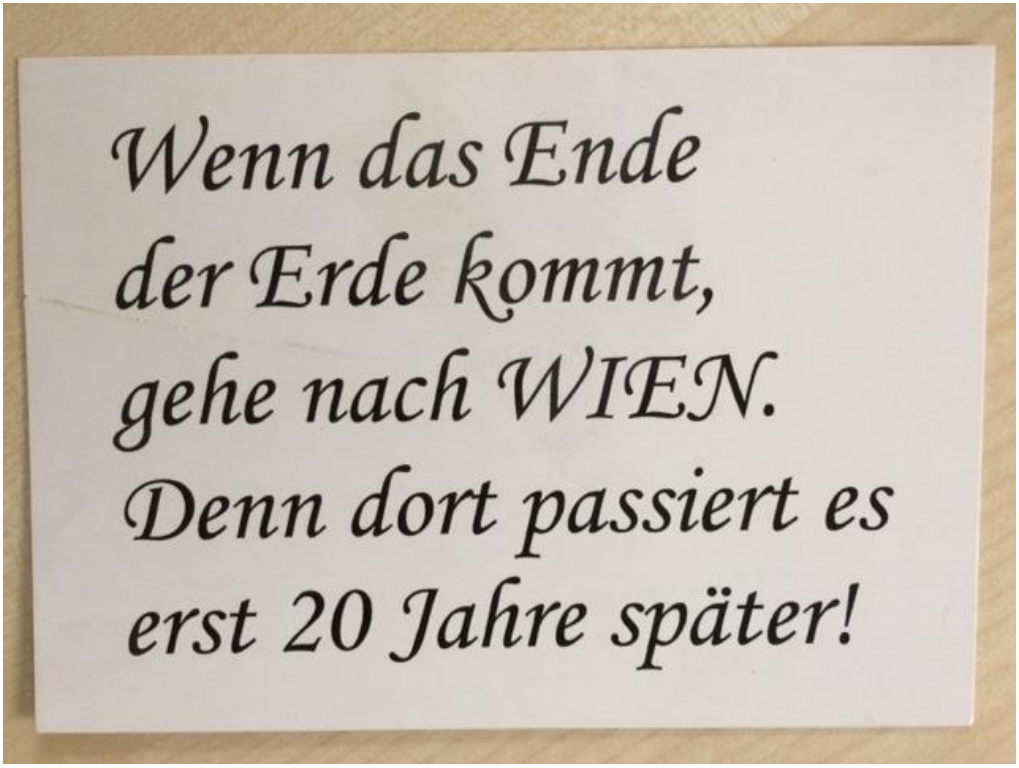
Aus dieser Geschichte meiner 45 Jahren in der deutschsprachigen IT-Welt geht hervor, dass es in diesem Sinne eine Software Engineering nie gegeben hat, denn zu keiner Zeit waren die Projekte wirklich „predictable“ und unter Kontrolle. Ich habe alles versucht, mit der Softorg Software Produktionsumgebung die Kontrolle über die Projekte zu gewinnen und bin damit trotz gelegentlicher Erfolge am Ende gescheitert. Nur mit den Wartungs- und Migrationsprojekten ist es einigermaßen gelungen, die Kontrolle über die Projekte zu gewinnen, in den Entwicklungsprojekten ist es nur in Ausnahmefällen gelungen. Das liegt daran dass in Wartungs- und Migrationsprojekten einen wohldefinierten, messbaren Ausgangszustand gegeben ist – der bestehende

Source-Code. Die Erfahrung aus dem Projekt bei Thyssen-Stahl hat mich gelehrt, dass Entwicklungsprojekte ohne Vorgänger, bzw. Prototyp, weder planbar noch kontrollierbar sind. Sie sind Forschungsexperimente. In diesem Sinne stimme ich mit DeMarco überein, aber er behauptete, es hat mal funktioniert und dann nicht mehr. Ich würde behaupten, es hat nie richtig funktioniert. Ab und zu mal, wo die Umstände besonders günstig waren, sind die Projekte an das Ideal nahe herangekommen. Ein Modell aber, das nur für Ausnahmefälle geeignet ist, kann kein gültiges Modell sein. So war der Begriff Software Engineering vom Anfang an unzutreffend. Ein Software System-Entstehungsprozess ist mit einem industriellen Fertigungsprozess nicht vergleichbar [Sned13].

Anderseits steckt hinter dem Begriff Software Engineering viel mehr als nur das Projektmanagement. Der Disziplin Software Engineering beinhaltet eine Menge einzelner Techniken, die sehr wohl planbar und kontrollierbar sind – wie Anforderungsanalyse, System Design, Codekonstruktion, Softwaretest und Konfigurationsmanagement. Diese Techniken haben eine theoretische Grundlage, sie sind erlernbar und auch in der Praxis umsetzbar. Wenn wir Software Engineering als ein Sammelsurium einzelner handwerkliche Tätigkeiten verstehen, dann ist sie noch lange nicht tot. Da hat DeMarco Unrecht. Software Engineering lebt und wird noch lange weiterleben. Es ist schließlich eine Frage der Definition [Poor04].

12.8 Wien hat auf mich gewartet

Ich habe dieses Buch mit einer Interpretation von Tom DeMarco begonnen. Sein Buch „Peopleware“ wurde von Peter Hrushka aus welchen Gründen auch immer in „Wien wartet auf Dich“ übersetzt. Ich beende dieses Buch mit einer weiteren Interpretation von DeMarco über das Schicksal des Software Engineering. Wenn es zutrifft, was er am Ende behauptet, das Software Engineering keine Zukunft hat – dann kann ich mich zu Ruhe setzen. Ich habe ausgedient. Und wenn es zutrifft, was er am Anfang behauptet – das Wien der Sehnsuchtsort aller ausgestressten amerikanischen Softwerker ist, dann bin ich eh schon im Paradies angekommen.


Wenn das Ende
der Erde kommt,
gehe nach WIEN.
Denn dort passiert es
erst 20 Jahre später!

Referenzen

Kapitel 1:

[DeMa87] DeMarco, T., Lister, T.: Peopleware – Productive Projects and Teams, Dorset House Pub., New York, 1987.

[DeMa99] DeMarco, T., Lister, T., Hrushka, P.: Wien wartet auf Dich – Der Faktor Mensch im DV-Management, Hanser Verlag, München-Wien, 1999.

[Joel75] Joel, B.: "The Stranger", American Pop Song, aus den 70er Jahren, Los Angelos, 1975

[Mein77] Meinike, U.: "Da kommt ein Tag in Wien", deutsche Übersetzung von "the Stranger", Munich, 1977.

[Zweig82] Zweig, S. : Die Welt von Gestern - Erinnerungen eines Europäers, S. Fischer Verlag, Frankfurt, 1982

[Marx94] Marx, K.: Das Kapital, in Marx-Engels Werke, Dietz Verlag, Berlin (Ost), Buch III, 1894.

Kapitel 2:

[Dijk67] Dijksta, E. "GoTo Statement considered harmful", Communications of the ACM, Band 11, Nr. 3, 1968, S. 147

[Royce70] Royce, W.: "Managing the Development of large Software Systems", Proceedings of IEEE WESCON 26, August 1970, S. 1–9.

[Helm70] Helm, B. "Normierte Programmierung", Computer Praxis, Heft 12, Dez., 1970.

[Sned71] Sneed, H., Jacobs, G.: "Modularprogrammierung - Technik, Methodik und Organisationsform" Online Zeitschrift für Datenverarbeitung, Heft 7, Okt. 1971, S. 487.

[Mert72] Mertens, P.: Industrielle Datenverarbeitung – Operationssysteme, Band 2, „StudentenOperationssystem" , Gabel Verlag, Wiesbaden, 1972

[Stru72] Strünz, H.: "Entscheidungstabellen und ihre Anwendung bei Systemplanung, -implementierung und documentation", Elektronische Datenverarbeitung, Heft 12, 1970, S. 56.

[Sned73] Sneed, H.: “Programorganisation” Online Zeitschrift für Dataenverarbeitung, Heft 10, Okt. 1973, S. 688.

[Mund74] Mundhenke, E.: “Haushaltswesen der Hochschulen”, HIS Brief 17, Hannover, 1971.

[Walt74] Walter, E., Sneed, H.: “ADV-Projekterfahrungen einer kommunalen Datenverarbeitungszentrale”, ÖVD – Öffentliche Verwaltung und Datenverarbeitung”, Heft 10, Okt. 1974, S. 462.

[Sned75] Sneed, H.: “Die Strukturierte Programmierung” in Heilmann, H. ed. 4. Jahrbuch der EDV, Forkel Verlag, Stuttgart, 1975 S. 187.

[Mund75] Mundhenke, E, Sneed, H. Zöllner, U.: Informationssysteme für Hochschulverwaltung, Walter de Gruyter Verlag, Berlin, 1975.

[Sned76] Sneed, H.: „Balanceakt Programmoptimierung“, Online Zeitschrift für Datenverarbeitung, Heft 9, Sept. 1976, S. 545.

[Mills72] Mills, H.: “IBM Chief Programmer Concept”, in IBM Systems Journal, 1972

[Sned77] Sneed, H.: “Software-Qualitätskontrolle – Der Preis der Systemzuverlässigkeit”, Online Journal für Informationsverarbeitung, Heft 10/77, Okt. 1977, S. 820

[Conw68] Conway, M. : „Conway’s Law of System Architecture“ Datamation, Band 14, Nr. 5, April 1968, S. 28

[Kirch77] Kirchhof, K. : „Der Prüfplangesteuerte Programmtest“, Online Zeitschrift für Datenverarbeitung, Heft 4, April 1975, S. 245.

[Sned78] Sneed, H.: „Der Software Prüfstand“ im Ed. Heilmann, H. Handbuch der Datenverarbeitung, Forkel Verlag, Stuttgart, 1978

[Sned83] Sneed, H.: „Prüfstand für den Modultest“, Proc. of First Int. Conference on Software Maintenance, IEEE Computer Society Press, Monterey CA., 1983

[Mill79] Miller, E.: Ed. Infotech State of the Art Report on Software Testing, Infotech International, Maidenhead, G.B., 1979

Kapitel 3:

[Alfo77] Alford, M.: “A Requirements Engineering Methodology for Realtime Systems” IEEE Trans on SE, Vol. SE-2, Nr. 1, Jan. 1977, S. 60

[Mill77] Miller, E.: „Program Testing - Art meets Theory“, IEEE Computer Magazine, Juli, 1977, S. 42

[Rama75] Ramamoorthy, C, : “Testing large Software with automated Software Evaluation Systems”, IEEE Trans on S.E., Vol. SE-1, Nr. 1, March 1975, S. 31

[Siem77] Gewald, Haake, Pfadler: Software Engineering – Grundlagen und technik rationeller Programmentwicklung, Oldenbourg Verlag, München, 1977

[McCa77] McCall, J., Richards, P., Walters, G.: “Factors in Software Quality” in Concepts and Definitions of Software Quality, General Electric NTIS Pub., Vol. 1, Springfield, Va., Nov. 1977

[NCCD77] National Computer Conference, Dallas, 1977

[Spil05] Spillner, A./ Linz, T./ Schaefer, H.: Software Testing Foundations, dpunkt Verlag, Heidelberg, 2005

[NaSh73] Nassi, I., Sheidermann, B.: “Flowchart Techniques for Structured Programming”, ACM Sigplan Notices, Vol.8. Nr. 8, 1973, S. 12

[KrNo12] Kruchten, P./Nord, R.: „Technical Debt – from Metaphor to Theory and Practice“. IEEE Software, Dez. 2012, S. 18

[Sned78] Sneed, H. : “A formal Notation for defining Structured programs” Proceedings of Online Conference on Pragmatic Programming, London, 1978

[Sned79] Sneed, H.: “Das Budapester Testlabor” im Ed. Heilmann, H. Handbuch der Datenverarbeitung, Forkel Verlag, Stuttgart, 1979

[Budd80] Budd, T., DeMillo, R., Lipton, R., Sayward, F.: „Design of a Prototype Mutation system for Program Testing“, in Proc. of AFIPS, Vol. 47, 1980, S. 623

[Faga76] Fagan, M.E.: “Design and Code Inspections to reduce errors in program development”, IBM Systems Journal, Vol. 15, No. 3, 1976

[HVG80] Redaktion, “Az en tökem magyar munkatarsaim fejeben van”, Heti Vilaggazdasag, Jan. 12, 1980, S. 30

Kapitel 4:

[Howd87] Howden, W.: Functional Program Testing & Analysis, McGraw-Hill, New York, 1987

[Alfo77] Alford, M.: “Requirements Engineering Methodology for Realtime Systeme”, IEEE Trans on SE, Vol. 2, Nr. 1, Jan. 1977

[Huang75] Huang, J.: “An approach to program testing” Computing Surveys, Vol. 7, No.3, 1975

[Howd80] Howden, W.: „Functional Program Testing“, IEEE Trans on SE, Vol. 6, Nr. 2, 1980, S. 293

[McClu89] McClure, C, Martin, J.: CASE is Software Automation, Prentice-Hall, Englewood Cliffs, N.J., 1989

[Bröh95] Bröhl, A., Dröschel, W.: Das V-Modell – Der Standard für die Softwareentwicklung mit Praxisleidfaden, Oldenbourg Verlag, München, 1995

[Sned87] Sneed,H.; Software-Management, Rudolf Müller Verlag, Köln, 1987

[Sned80] Sneed,H.; Software-Entwicklungsmethodik, Rudolf Müller Verlag, Köln, 1980

[BoMe82] Bons, H., van Megen, R., Schmitz, P.: Software-Qualitätssicherung Testen im Software-Lebenszyklus, Vieweg Verlag, Wiesbaden, 1982, S. 21

[DeMa78] DeMarco, T.: Structured Analysis and System Specification, Yourdan Press, New York, 1978

[Cons74] Constantine, L., Stevens, R., Myer, G.: "Structured Design", IBM Systems Journal, Vol. 13, Nr. 2, 1974

[Jack75] Jackson, M.: Principles of Program Design, Academic Press, London, 1975

[Chen76] Chen, P.: "The Entity/Relationship Model – Towards a unified View of Data", ACM Transactions on Database Systems, Vol. 1, Nr. 1, 1976, S. 9

[Nyar83] Nyari, E., Sneed, H.: "SOFPEC – A pragmatic Approach to automated Specification Verification" Journal of Systems and Software, Nr. 3, North-Holland Press, Amsterdam, 1983

[Your79] Yourdan, E., Constantine, L.: Structured Design – Fundamentals of a Discipline of Computer Program and System Design, Prentice-Hall, Englewood Cliffs, N.J., 1979

[Dave77] Davis, C., Vick, C.: "The Software Development System", IEEE Trans on SE, Vol. SE-3, Nr. 1, Jan., 1977, S. 23

[Mart85] Martin, J.: Fourth-Generation Languages, Prentice-Hall, Englewood Cliffs, N.J., 1985

[Mart85a] Martin, J.: Manifest für die Informationstechnologie von morgen, Econ Verlag, Düsseldorf, 1985

[IBM89] Anonymous IBM Team: "Use of Application Generators" in CASE – The Potential and the Pitfalls, QED Information Sciences, Wellesley, MA., 1989, , John Wiley & Sons, New York, 1981

[Rice81] Rice, J.: Build Program Technique – A practical Approach to the Development of automatic Software Generation Systems

[SnMe85] Sneed, H., Merey, A.: "Automated Software Quality Assurance", IEEE Trans on SE, Vol. SE-11, Nr. 9, Sept.1985, S. 909.

[Majo83] Majoros, M., Sneed, H.: “Testing Programs against a formal Specification”, Proceedings of COMPSAC-83, IEEE Computer Society Press, Chicago, 1983, S. 512

[Erdo85] Erdös, K., Sneed, H.: “SoftInt – Software Integration Test System”, Jorunal of Systems and Software, Nr. 7, North Holland, Amsterdam, 1985

[Sned86] Sneed, H.: “Data Coverage Measurement in System Testing”, Proceedings of 2nd ACM Workshop on Software Testing, Banff, Canada, Juli, 1986

[Boeh96] Boehm, B. „Anchoring the Software Process“, IEEE Software, July 1996, S. 73

[Sned83] Sneed, H., Gawron, R.: “The Use of the Entity-Relationship Model as a Schema for managing Data Processing Activities”, in Ed. C. Davis, The Entity-Relationship Approach to Software Engineering, North-Holland, Amsterdam, 1983, S. 717

[Haus85] Hausen, H.-L., Müllerburg, M., Sneed, H.: Software-Produktionsumgebungen, Verlagsgesellschaft Rudolf Müller, Köln, 1985

Kapitel 5:

[Sned79] Sneed, H.: “Das Quelle Software Testsystem”, CDI Proceedings of STRUCTO-79 Congress on Software Qualitätssicherung, Frankfurt, 1979

[Sned82] Sneed, H.: Software-Qualitätssicherung, Rudolf Müller Verlag, Köln, 1982

[Göll82] Göller, G., Stoffel, C.: “KTDS – ein integrierter Satz von Werkzeugen für Analyse, Test und Wartung embedded Software”, Proceedings of Tagung I des German Chapter of the ACM, Ed. H. Sneed und H. Wiehle, Software-Qualitätssicherung, Bundeswehrhochschule, Neubiberg, März, 1982, S. 203

[Sned82] Sneed, H.: “Ein Integrationstestsystem für die Lagerhaltung”, Online Journal für Informationsverarbeitung, Heft 5/82, Sept. 1982, S. 28.

[Sned08] Sneed, H.: “Bridging the Concept to Implementation Gap in Software Testing”, IEEE Proc. of QSIC2009, Oxford, G.B., 2008, S. 92

[Wieh82] Sneed, H., Wiehle, M.: Software-Qualitätssicherung, Berichte des German Chapter of ACM, Band 9, Neubiberg, 1982

[Jand83] Jandrasics, G., Sneed, H.: „SoftDoc – A Static Analyzer for Commercial Programs“, in Proceedings of ACM SigSoft Workshop on Software Quality Assurance, Gaithersburg, MD., 1983

[Sned82a] Sneed, H.; “Die Datenverarbeitung zerfällt in zwei Welten”, Online Journal für Informationsverarbeitung, Heft 2/82, März 1982, S. 64.

[Bend83] Bender, H. et. al.: Software Engineering in der Praxis - Das Bertelsmann Modell, CW-Publikationen, München, 1983

[Majo82] Majoros, M., Sneed, H.: "SoftTest for Program Testing" Journal of Systems and Software, Nr. 5, North Holland, 1982, S. 39

[Sned84] Sneed, H.: „Software Renewal – A case study", IEEE Software Magazin, Vol. 1, No. 3, 1984, S. 56

[Nyar83] Nyary, E., Sneed, H.: „SoftSpec – A pragmatic Approach to automated Specification" Journal of Systems and Software, Nr. 3, North Holland, 1983

[Sned84] Sneed, H.: „Requirements Analysis of Business Systems", Proceedings of international Computing Symposium, German Chapter of ACM, Bericht Nr. 13, Teubner Verlag, Stuttgart, 1984

[Gutz88] Gutzwiller, T., Österle, H.: Entwicklungssysteme und 4. Generationssprachen, AIT – Angewandte Informationstechnik Verlag, Halbermoos, 1988

[Geri84] Gerisch, M., Schumann, J.: Softwareentwurf – Prinzipien, Methoden, Arbeitsschritte, Rechnerunterstützung, VEB Verlag Technik, Berlin, 1984

[Bend83] Bender, H.: „Das Management der Software-Wirtschaftlichkeit", Proceedings zum Software Forum 83 – Effizientes Software Management, Computerwoche Verlag, München 1983, S. 67

[Oest03] Österle, H., Gronover, S.: „Multikanalmanagement bei Finanzdienstleistern", Handbuch moderner Datenverarbeitung – HMD Heft 233, DPunkt Verlag, Heidelberg, 2003, S. 82

[Schnei85] Schneider, M. : „SoftOrg – ein deutsch-ungarisches Verbundprojekt", Computer Magazin, Nr. 6, June 1985, S. 57

[Sned89] Sneed,H.: "The Myth of Top-Down Development and its Consequences for Software Maintenance", Proc. Of 5th ICSM, IEEE Computer Society Press, Miami, Nov. 1989, S. 22

[Bröhl93] Bertram, H., Blönningen, P., Bröhl, A.-P.: CASE in der Praxis, Springer Verlag, Berlin, 1993

[Noth84] Noth, T., Kretzschmar, M.: Aufwandsschätzung von DV-Projekten, Springer-Verlag, Berlin, 1984

[TRUST89] Esprit Project 1258: Testing and consequent reliability estimation for real-time embedded software, EU Project Report, University of Liverpool, August, 1989

[Sned87] Sneed, H., Jandrasics, G.: Software Recycling, IEEE Proc. Of 3rd ICSM, Computer Society Press, Austin, Sept. 1987, S. 82.

[DOCK91] Esprit Project 5111: Reverse Engineering Approaches, Tools and Intermediate Languages, EU Project Report, CRIAI, Portici, Italy, April 1991

[Fowl01] Fowler, M.: "The Agile Manifesto" www.martinfowler.com/articles/newMethodology.html),

[Sned90] Sneed,H.: "Top-Down Software Development considered harmful", Proc of 6th ICSM, IEEE Computer Society Press, San Diego, Nov. 1990, S. 102

[Brok87] Brooks, F.: "The Silver Bullet – Essence and Accidents of Software Engineering", IEEE Computer, April 1987, S. 10

[Spil94] Spillner, A.: "Kann eine Krise 25 Jahre dauern", Informatikspektrum, Band 17, 1994, S. 48-52

[Jack98] Jackson, M.: "Will there ever be Software Engineering?", IEEE Software, January, 1998, S. 36

[Shaw90] Shaw, M.: "Prospects for an Engineering Discipline of Software", IEEE Software, November, 1990, S. 15

[Dene93] Denert, E.: "Software-Engineering in Wissenschaft und Wirtschaft – wie breit ist die Kluft?", Informatikspektrum, Band 16, 1993, S. 295

[Sned88] Sneed, H.. "Software-Testen Stand der Technik", Informatikspektrum, Band 11, Heft 6, Dez. 1988, S. 303

[Sned89] Sneed, H.: "Software-Engineering – ein Überblick", in PIK – Praxis der Informationsverarbeitung und Kommunikation, Nr. 1/89, Januar 1989, S. 11

Kapitel 6:

[McCra82] McCracken, D., Jackson, M.: "Life Cycle Concept considered harmful", ACM: Software Eng Notes, Vol. 7, No. 2, April 1982

[Jack81] Jackson, M.: "Critique of the Life Cycle Concept", Proc of IEEE Conference on Systems Analysis and Design, Atlanta Ga., Elsevier North Holland, 1981

[Glad82] Gladden, G.: "Stop the Life Cycle, I want to get off", ACM: Software Eng Notes, Vol. 7, No. 2, April 1982

[Balz82] Balzert, G.: "On the inevitable intertwining of specification and implementation", Communications of ACM, Vol. 25, Nr. 7, Juli, 1982

[Perl76] Perlis, A.: "Proofs and Refutations – The logic of mathematical Discovery", Cambridge University Press, Cambridge G.B., 1976

[Fetz88] Fetzer, J.: "Program Verification – The very idea", Communications of the ACM, Vol. 31, Nr. 9, Sept. 1988

[Rich88] Rich, D., Waters, F.: “The Programmer’s Apprentice”, IEEE Computer, Nov., 1988

[Mont91] Montgomery, S.: AD/Cycle – IBM’s Framework for Application Development and CASE, van Nostrand Reinhold, New York, 1991

[Knut74] Knuth, D.: “Computer Programming as an Art”, Communications of the ACM, Vol. 17, Nr. 12, May, 1974

[Beck01] Beck, K. et al.: “Manifest für Agile Softwareentwicklung”, agilemanifesto.org/ iso.de, 2001

[Bach01] Bach, J.: „Rethinking the Role of Testing for the e-Business Era“, Cutter IT Journal, Vol. 13, No. 4, April 2000, S. 15

[Shne80] Shneidermann, B.: Software Psychology – Human Factors in Computer and Information Systems, Winthrop Publishers, Cambridge, MA., 1980

[Sned91] Sneed, H. “Software muss meßbar werden”, Information Management, Nr. 4/91, Nove. 1991, S. 56

[Bröh90] Bröhl, A.: “SE-Umgebung für Informationssysteme der Bundeswehr”, ÖVD-Online, Nr. 3/90, S. 96

[Tock05] Tockey, S.: Return on Software – maximizing the Return on your Software Investment, Addison-Wesley, Boston, 2005

[Your92] Yourdan, E.: Decline and Fall of the American Programmer, Prentice-Hall, Englewood Cliffs, N.J., 1992

[Rock92] Rockwell, B.: “The Coming of CASE”, American Programmer, Vol. 5, Nr. 9, S.2

[Mart91] Martin, J.: Systems Application Architecture – SAA, Prentice-Hall, Englewood Cliffs, N.J., 1991

[Sned94] Sneed, H.M.: Validating Functional Equivalence of Reengineered Programs, in Software Testing, Verifications and Realibility, Band 4, Nr. 1, März 1994, S. 33.

[Mars89] Marshall, T., Veevers, A.: “Code Coverage and Remaining Error Probability”, Trust Report,Esprit Project 1258, Liverpool, 1989

[SnKa90] Sneed, H., Kaposi, A.: “A Study on the Effect of Reengineering upon Software Maintainability”, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, San Diego, Nov. 1990, S. 91

[Sned91] Sneed, H.: “Software muss messbar werden”, Information Management, Nr. 4/91, November, 1991, S. 56

[SnRo96] Sneed, H./Rothhardt,G.: “Softwaremessung”, Wirtschaftsinformatik, Band 38, Heft 2, April 1996, S. 172

[Sned95] Sneed, H.: “Understanding Software through Numbers”, Journal of Software Maintenance, Vol. 7, Nr. 6, Dez.1995, S. 405

[Sned93] Sneed, H., Ritsch, H.: “Reverse Engineering via dynamic Analysis”, Proceedings of 1st WCRE Workshop, Baltimore, MD, May 1993, S. 192

[Sned87] Sneed, H. “Software ist vielfach sanierungsreif”, Online, Zeitschrift für Datenverarbeitung, Feb., 1987

Kapitel 7:

[Oest85] Oestreich, H.: “Der Toolmarkt – Strukturen und Trends”, Computer Magazin, Vol. 14, Oktober, 1985, S. 32

[Sned91] Sneed, H.M.: Economics of Software Reengineering, Journal of Software Maintenance, Vol. 3, No. 3, Sept. 1991, s. 163

[Sned91a] Sneed, H,: “Reengineering in a Swiss Bank”, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, Sorrento, 1991

[Sned90] Sneed, H.: “Die Data-Point Methode” in Online - Zeitschrift für Datenverarbeitung, Nr. 5, Mai 1990, S. 48.

[Sned88] Sneed, H./Jandrasics, G : “Inverse Transformation of Software from Code to Specification”, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, Phoenix, 1988, S. 102.

[Sned95] Sneed, H,: “Reverse Engineering with a CASE Tool”, Proc. of Int. Workshop on Reverse Engineering, IEEE Computer Society Press, Toronto, 1995

[Nyar92] Nyary, E.: “Procedural to Object-oriented Transformation”, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, Orlando, 1992

[Sned90] Sneed, H.: Softwaresanierung, Rudolf Müller Verlag, Köln, 1990

[Sned95] Sneed, H.M.: Planning the Reengineering of Legacy Systems. IEEE Software, Vol. 12, No. 1, S. 24-34, Jan. 1995, p. 24.

[Sned91] Sneed, H.M.: Migration of procedurally oriented COBOL programs in an object-oriented Architecture, in Proc. Of ICSM-91, Computer Society Press, Montreal, Nov. 1991, S. 105.

[Arno90] Arnold, R.: Impact Analysis of Software Systems, McGraw-Hill, New York, 1990, S. 11

[Sned94] Sneed, H.: “Validating Functional Equivalence of Reengineered Programs, in Software Testing, Verifications and Realibility, Band 4, Nr. 1, März 1994, S. 33.

[Guen92] Guengerich, S.: Downsizing Information Systems, SAMS Publishing, Prentice-Hall Publishing, Carmel Indiana, 1992

[Sned95] Sneed, H.: “Implementation of a Software Maintenance Workbench at the Union Bank of Switzerland” in Proc. of Oracle Conference on System Downsizing, Oracle Institute, Munich, April 1995

[Eber01] Ebert, C./de Neve, P.: “Surviving Global Software Development”, IEEE Software, March 2001, S. 62

Kapitel 8:

[Kobi01] Kobitsch,W./Rombach, D./Feldmann, R.; “Outsourcing to India”, IEEE Software, March 2001, S.78

[Mert12] Mertens, P.: “Schwierigkeiten mit IT-Projekten der Öffentlichen Verwaltung”, InformatikSpektrum, Band 35, Heft 6, 2012, S.433

[FAZ12] Frankfurter Allgemeine Zeitung, “Einvernehmen über E-Bilanz für den Fiskus”, FAZ vom 31.5.2012, S. 12

[Sned96] [Sned01] Sneed, H.M.: "Recycling Software Components extracted from Legacy Programs", ACM Proc. of IWPSE-2001, ACM SigSoft, Vienna, 2001, S. 43.

[Roß15] Roßbach, P.: Docker – Container-Infrastruktur für Microservices, dpunkt Verlag, Heidelberg, 2015

[SnNy99] Sneed, H./ Nyary, E: “Salvaging an ancient Legacy System at the German Foreign Office”, Proc. of Int. Conference on Software Maintenance, IEEE Computer Society Press, Oxford G.B., Sept. 1999, S. 434

[Diet88] Dietrich, T.: Saving a Legacy system with Objects, Proc. of OOPSLA-88, ACM Press, New York, 1989, S. 54.

[Sned96] Sneed, H.M.: Encapsulating Legacy Software for Reuse in Client/Server Systems, IEEE Proc. Of 3rd WCRE, Computer Society Press, Monterey, Nov. 1996, S. 104.

[Sned96a] Sneed, H. “Die Einbindung alter Host-Software in eine Client/Server Architektur”, ObjektSpektrum, Nr. 4, Juli 1996, S. 36

[Ward99] Ward, M. “Assembler to C Migration using FermaT Transformation System”, Proceedings of ICSM1999, IEEE Computer Society Press, Oxford G.B., Sept. 1999, S.67

[Borc97] Borchers, J.: “Erfahrungen mit dem Einsatz einer Reengineering Factory in einem großen Umstellungsprojekt”, Handbuch moderner Datenverarbeitung, Band 34, Heft 194, 1997, S. 77-94.

[SnMa98] Sneed, H./Majnar, R.: “A Case Study in Software Wrapping”, Proceedings of ICSM1998, IEEE Computer Society Press, Washington D.C., 1998, S. 86

Kapitel 9:

[KaBa03] Kaner, C., Bach, J., Pettichord, B.: Lessons learned in Software Testing, John Wiley & Sons, New York, 2003

[Budd98] Budd, T.: Object-oriented Programming with Java, Addison-Wesley-Longman, Reading Mass., 1998

[Qual99] Qualine, S: C Elements of StyleC Programming, M&T Publishing, San Mateo, CA, 1992

[Simo97] Simoni, K: “Hungarian Notation” https://msdn.microsoft.com/en-us/library/aa260976(v=vs.60).aspx

[Sned99] Sneed, H. “Static Analysis of object-oriented Programs with the CodeAnal Toolset” GMD-TAV Tagung, Herbst, 1999, S. ???

[Sned99a] Sneed, H.: „Applying Size, Complexity and Quality Metrics to object-oriented Applications”, Proc. Of 10th European Software Control and Metrics, Herstmonceux, G.B., April, 1999, S. 377

[Sned00] Sneed, H.: Aufwandsschätzung in der Softwarewartung, Proc. of GI Software Management 2000, Österreichische Computer Gesellschaft, Marburg, Nov. 2000, S. 167

[Fowl99] Fowler, M.: Refactoring – Improving the Design of existing Code, Addison-Wesley, Reading MA, 1999

[Sned00] Sneed H.: „Aufbau einer Software Repository für komplexe Client/Server Systeme“ in Proc. of 2nd GI Workshop on Software Reengineering, Univ. Koblenz, Bad Honnef, Mai 2000

[SnDo99] Sneed, H. / Dombovari, T.: „Comprehending a complex distributed object-oriented Software system“ Proc. Of 7th Workshop on Program Comprehension, IEEE Computer Society Press, Pittsburgh, May 1999, S. 218

[Sned04] Sneed, H.: A Cost Model for Software Maintenance and Evolution, Proceedings of ICSM 2004, IEEE Computer Society Press, Chicago, September 2004, S. 264.

[Sned01] Sneed, H.: „Impact Analysis of Maintenance Tasks for a distributed Banking System“, Proceedings of ICSM 2001, IEEE Computer Society Press, Florenz, November 2001, S. 180.

[Sned04] Sneed, H.: „Reverse Engineering of Test Cases for Selective Regression Testing”, Proc. of European Conference on Software Maintenance and Reengineering, CSMR-2004, IEEE Computer Society Press, Tampere, Finnland, March 2004, S. 69

[SHT04] Sneed, H./ Hasitschka, M./ Teichmann, M.-T.: Software Produktmanagement , dpunkt Verlag, Heidelberg, 2004

[SnBr03] Sneed, H., Broessler, P.: “Critical Success Factors in Software Maintenance” Proceedings of ICSM 2003, IEEE Computer Society Press, Amsterdam, September 2003, S. 190.

[SnJu06] Sneed, H., Jungmayr, S.: “Produkt und Prozessmetrik für den Softwaretest”, Informatikspektrum, Band 29, Nr. 1, 2006, S. 23

[Sned03] Sneed, H.: Software Testmetriken für die Kalkulation der Testkosten und die Bewertung der Testleistung, GI Softwaretechnik Trends, Band 23, Heft 4, Nov. 2003, S.11

[Sned05] Sneed, H.: “Measuring the Effectiveness of Software Testing“, Proc. Of Metrikon2005, Shaker Verlag, Kaiserslautern, Nov. 2005, S 145

[LeSn04] Lehner, F./Sneed,H.: “Braucht die Wirtschaftsinformatik ein eigenständiges Cuuriculum für Software-Engineering”, Wirtschaftsinformatik, Band 46, Heft 6, 2004, S. 26

[SnWi02] Sneed, H./ Winter, M.: Testen objektorientierter Software, Hanser Verlag, München-Wien, 2002

Kapitel 10:

[Sned05] Sneed, H.: Software-Projektkalkulation, Hanser Verlag, München-Wien, 2005

[Brow73] Brown, A., Sampson, W.: Program Debugging, American Elsevier, New York, 1973

[Dunn84] Dunn, R.: Software Defect Removal, McGraw-Hill, New york, 1984

[Groc91] Grochow, J.: SAA- A Guide to implementing IBM#s Systems Application Architecture, Yourdan Press, Englewood Cliffs, N.J., 1991

[BrSt95] Brodie, M. / Stonebraker, M.: Migrating Legacy Systems, Morgan-Kaufmann Pub., San Francisco, 1995

[SnGö00] Sneed, H./ Göschl, S.: „Testing Software for Internet Applications", Software Focus, Vol. 1, Nr. 1, Sept. 2000, S. 15

[Sned04] Sneed, H. „Testing reengineered Reports", Proceedings of 11th WCRE Conference, IEEE Computer Society Press, Delft, NL., Nov. 2004, S. 17

[Sned05] Sneed, H.: " Selektiver Regressionstest einer DOTNET Anwendung", Objektspektrum, Nr. 5, Sept 2005, S. 72.

[Sned05a] Sneed, H.: „System Testing in Old Europe", Professional Tester, Feb. 2005, S. 21

[Sned06] Sneed, H.: „Reverse Engineering deutschspracher Fachkonzepte", GI – Softwaretechnik Trends, Band 25, Heft 2, WSR Proceedings, Bad-Honnef, Mai 2005., S. 49

[Sned07] Sneed, H.: "Testing against natural language Requirements", 7th Int. Conference on Software Quality (QSIC2007), Portland, Oct. 2007

[Sned05] Sneed, H.: Testing an eGovernment Website. Paper presented at 7th IEEE International Symposium on Web Site Evolution (WSE2005), Budapest, S. 3

[Sned04b] Sneed, H.: "Measuring the Effectiveness of a Test", Paper for EuroStar Test Conference in Edinburgh, Nov. 2004

[SnSB06] Der Systemtest, Hanser Verlag, München-Wien, 2006,

[BrRa06] Broy, M.; Rausch A.: Das neue V-Modell XT, Ein anpassbares Modell für Software und System Engineering, Informatik-Spektrum 3(28), Juni 2005, S. 220-229.

[Sned14] Sneed, H.: "Automated Quality Assurance of Software Requirements", Requirements Engineering , Nr. ??, 2014

[Cock02] Cockburn, A.: Agile Software Development, Addison-Wesley, Reading, Ma., 2002

[WiSn12] Winter, M. et al. : Der Integrationstest, Hanser Verlag, München-Wien, 2012

[Sned06] Sneed, H.: "Testing a Datawarehouse – an industrial challenge", Proceedings of TAICPART Conference, IEEE Computer Society Press, Windsor, G.B. August 2006, S. 203

[CrGr09] Crispin, L. / Gregory, J.: Agile Testing – A practical Guide for Testers and agile Teams, Addison-Wesley-Longman, Amsterdam, 2009

[Sned02] Sneed, H.: “Using XML to integrate existing Software Systems”, Proceedings of 26th CompSac Conference, IEEE Computer Society Press, Oxford, G.B. August 2002, S. 167

[Sned08] Sneed, H.: “Bridging the Concept to Implementation Gap in Software System testing”, Proceedings of 8th International Conference on Quality Software, Oxford, G.B., August, 2008, S. 67

[Broy07] Broy, M./ Geisberger, E./ Kazmeier, J./ Rudorfer, A.: „Ein Requirements Engineering Referenzmodell“, Informatikspektrum, Band 30, Nr. 3, 2007, S. 127

[Sned09] Sneed, H.: “Anforderungsmetrik – Zur Messung der Größe, Komplexität und Qualität von Anforderungsspezifikationen“, Tagungsband der Metrikon-09, Shaker Verlag, Kaiserslautern, Nov. 2009, S. 29

[Sned15] Sneed, H.: “Measuring the Degree of Requirements Fulfillment”, Tagungsband der Metrikon-15, Shaker Verlag, Kaiserslautern, Nov. 2015, S. 5

[SnJu10] Sneed, H/ Jungmayr, S.: “Mehr Testwirtschaftlichkeit durch Value-driven Testing“, Informatikspektrum, Band 34, No. 2, 2010, S. 192

Kapitel 11:

[Sned97] Sneed, H.: „Measuring the Performance of Software Maintenance Departments

in Proc. Of CSMR 1997, Berlin, March 1997, S. 119

[Sned05] Sneed, H.M.: Software-Projektkalkulation - Praxiserprobte Methoden der Aufwandsschätzung für verschiedene Projektarten. Hanser, München/Wien, 2005, S. 159.

[Albr83] Albrecht, A./Gaffney J.: „Software Function, Lines of Code and Development Effort Prediction – A Software Science Validation“, IEEE Trans. on S.E., Vol. 9, Nr, 6, Nov. 1983

[Noth84] Noth,T./ Kretzschmar, M.: Aufwandsschätzung von DV-Projekten, Springer-Verlag, Berlin, 1984

[Sned90] Sneed, H.: “Die Data-Point Methode”, Online, Zeitschrift für DV, Nr. 5, Mai 1990, S. 48

[Sned96] Sneed, H.: „Schätzung der Entwicklungskosten von objektorientierter Software“, Informatikspektrum, Band 19, Nr. 13, Jun. 1996

[OMG14] OMG –Object Management Group: „Automated Function Points (AFP), OMG Document Nr. 2014-01-03, http://www.omg.org/spec/AFP

[Sned08] Sneed, H.: „Measuring 70 Million Lines of Code“, Proc. of International Workshop on Software Measurement, Springer Verlag, München, 2008, S. 271

[Sned07c] Sneed, H.: “Das neue SoftCalc – ein Tool für die differenzierte Kalkulation unterschiedlicher Projektarten”, Proc. of Metrikon 2007, Shaker Verlag, Stuttgart,Nov. 2007, S. ???

[Sned07] Sneed, H., Huang, S.: “Sizing Maintenance Tasks for Web Applications”, Proceedings of CSMR Conference, Amsterdam, 2007, S. ???

[Albr79] Albrecht, A.: „Measuring Application Development Productivity“, Proc. Of Joint SHARE, GUIDE and IBM Symposium, Philadelphia, Oct. 1979, S. 83

[Abra04] Abran, A., Khelifi, A., Buglione, L.: “A System of Reference for Software Measurement with ISO 19761 (COSMIC FFP)”, in Software Measurement – Research and Application, Shaker pub., Aachen, 2004, S. 89

[SnHa09] Sneed, H., Huang, S.: “Sizing Maintenance Tasks for Web Applications”, Journal of Software Maintenance, Vol ??, Nr. ??, 2009, S.???

[Sned10] Sneed, H.: “Vergleich zweier Aufwandsschätzungen nach Function-Point und UseCase-Point“ DASMA-Metrikon Metrik Kongress, Shaker Verlag, Nov. 2010

[Sned13] Sneed, H.: “Estimating the Costs of Change“ DASMA-Metrikon Metrik Kongress, Shaker Verlag, Nov. 2013

[Maxw02] Maxwell, K./Forseluus, P.: Survey of Software Productivity in Europe, IEEE Software, Jan. 2000, S. 80

[SnEr12] Sneed, H./ Erdös, K.: “Measuring and Evaluating a DotNet Application to better predict Maintenance Effort”, Proc. of International Workshop on Software Measurement, Assisi, Italy, Nov. 2012, p. 42

[Sned12] Sneed, H.: “Nachhaltige Software Evolution”, Proc. of GI Software Management -SWM2012, Springer Verlag, Bielefeld, Nov. 2012, S. 50

[SnSn03] Sneed, H./ Sneed, S.: Web-basierte Systemintegration, Vieweg Verlag, Wiesbaden, 2003

[Sned15] Sneed, H. „Aufwandsschätzung der Softwarewartung und –Evolution“, GI-SW Management Tagung , Springer Verlag, Dresden, März, 2015, S. ???

[KKSM08] Kozlov, D./Koskinen,J./Sakkinen,M./Markkula,J.: „Assessing maintainability change over multiple software releases“, Journal of Software Maintenance and evolution, Vol. 20, no. 1, Jan. 2008, p. 31

Kapitel 12:

[Sned09] Sneed, H.M.: “Messung und Nachdokumentation eines uralten COBOL Systems zwecks der Migration in Java“, GI Software Engineering Notes, Band 29, Heft 2, S. 17

[Sned10a] Sneed, H. “Migrating from COBOL to Java”, IEEE Proc. of International Conference on Software Maintenance and Evolution, Temesvar, Romania, Sept. 2010, S.

[Sned10b] Sneed, H. “Automatisierte Migration alter COBOL Programme in Java”, GI Softwaretechnik-Trends, Band 30, Heft 2, Mai 2010, S. 62

[Sned11] Sneed, H.: “Migrating PL/I Code to Java” IEEE Proc. Of European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, Oldenbourg, March, 2011, S. 287.

[Sned05] Sneed, H.: An incremental Approach to System Replacement and Integration, Paper presented at 9th European Conference on Software Maintenance and Reengineering (CSMR05), Manchester, G.B., 2005, S. 196.

[Sned09] Sneed, H.: A Pilot Project for migrating COBOL Code to Web Services. International Journal of Software Tools Technology Transfer, 1(2), 2009, p. 103-129.

[SnSn12] Sneed, H./ Sneed, S.: “From Source Code to Business Process”, in Proc. Of MESOCA-2012 Workshop, RivadelGarda, Italy, Sept. 2012

[SnVe13] Sneed,H. /Verhoef,C.: „Migrating to Service-oriented Systems“ , Proc. of WSE-2013 -Web Systems Evolution2013, IEEE Computer Society, Eindhoven, Sept. 2013, S. 5.

[SnEr13] Sneed, H./ Erdös, K.: “Migrating AS400- COBOL to Java“; in Proc. of European Conference on Software Maintenance and Reengineering, CSRM-2013, IEEE Computer Society Press, Genova, March 2013, S. 231.

[Sned14] Sneed, H.: “Migration alter Assembler Programme in COBOL”, GI Softwaretechnik-Trends, Band 34, Heft 2, Mai 2014, S. 74

[Sned15] Sneed,H.: „Namensänderung in einem Reverse Engineering Projekt“, GI Softwaretechnik-Trends, Band 35, Heft 2, Mai 2015, S. 23

[Broy06] Broy, M.: “The Grand Challenge in Informatics – Engineering Software-intensive Systems”, IEEE Computer, Oktober, 2006, S. 72

[Glas02] Glass, R.: “Searching for the Holy Grail of Software Engineering”, Comm. of the ACM, Vol. 45, No. 5, May 2002, S. 15

[DeMa09] DeMarco, T. Software Engineering – An Idea whose time has come and gone”, IEEE Software, July, 2009, S. 94

[Sned13] Sneed, H.: Softwareevolution, dPunkt Verlag, Heidelberg, S. 109.

[Poor04] Poore, J.: “A Tale of Three Disciplines and a Revolution”, IEEE Computer Magazine, Jan. 2004, S. 30