Peter Rechenberg – Forscher, Lehrer, Mensch

Günther Blaschek, Hanspeter Mössenböck, Gustav Pomberger
Johannes Kepler Universität Linz

25 Jahre lang – von 1974 bis 1999 – war Peter Rechenberg Professor für Informatik an der Universität Linz. Generationen von Studenten haben ihn geschätzt und sein breites Wissen und tiefes Verständnis für die Informatik mit auf ihren beruflichen Lebensweg genommen. Auch für uns – seine Schüler und späteren Kollegen – ist Peter Rechenberg ein Vorbild, dem wir nacheifern und das wir bestrebt sind, an unsere Studenten weiterzugeben. In diesem Beitrag versuchen wir, ein Bild zu zeichnen von Peter Rechenberg als Forscher, als Lehrer und als Mensch.

Jugend und Entwicklungsjahre

Peter Rechenberg wurde am 18. Juli 1933 in Berlin geboren. Dort besuchte er auch die Volksschule und das Gymnasium. Schon als Abiturient fiel er durch eine Allgemeinbildung auf, die weit über das übliche Schulwissen hinausging. In der Schule, wo er nicht nur der Klassenbeste, sondern auch einer der beliebtesten Mitschüler war, lag seine Stärke in den naturwissenschaftlichen Fächern.

Privat interessierte er sich besonders für Literatur und Musik und hielt bereits in jungen Jahren im Freundeskreis fundierte Vorträge über *Thomas Mann* und *Bergengruen*. Seine Kenntnis und Verehrung der Klassiker machten Staunen. Unzählige Gedichte konnte er auswendig, davon über 100 allein von *Goethe*.

In der Musik, die bis heute seine besondere Passion geblieben ist, wusste er hervorragend Bescheid. Wissensdurst und Begeisterung brachten ihn selbstständig voran und sein Wissensstand auf dem Gebiet der Musik war schon im jugendlichen Alter außergewöhnlich. Er besuchte die Oper und Konzerte, absolvierte Volkshochschulkurse, hörte Radio und las Noten. Erst als Fünfzehnjähriger begann Peter Rechenberg mit dem Klavierunterricht, aber konsequente Zeiteinteilung und Disziplin – zwei seiner besonderen Persönlichkeitsmerkmale – machten es möglich, dass er jeden Tag zum Üben kam, mit kultiviertem Spiel Schulfeiern mitgestaltete und nach dem Abitur die Aufnahmeprüfung an der Hochschule für Musik für das Tonmeisterfach bestand.

So mancher mag sich fragen, woher ein Mensch in seinen jungen Jahren für all das die Zeit hernahm, umso mehr als damit die Interessensvielfalt von Peter Rechenberg noch bei weitem nicht erschöpft war. Schilderungen des Vaters weckten in dem Großstadtkind die Liebe zu den Bergen. Da er alles aus eigener Kraft und eigenem Antrieb tat, gründete er mit drei Schulkameraden eine Gruppe, die zunächst Alpenwanderungen machte. Ohne Geld natürlich, man fuhr per Anhalter, schlief irgendwo auf dem Fußboden, und das Essen war auch knapp. Der Hunger auf die Berge aber wurde immer größer und die Gruppe trat in Berlin dem Alpenverein bei.

Wo übte man aber fernab von Bergen und Felsen das Klettern? In Hausruinen, die es im Berlin der Nachkriegsjahre reichlich gab, und zum Abseilen eigneten sich Brücken und Kirchturmruinen. So fand man den neunzehnjährigen Peter Rechenberg nicht nur in der Schule, am Klavier, in Bibliotheken, nachts auf der Straße in einer langen Menschenschlange nach billigen Opernkarten anstehend und dann in Konzertsälen und in Theatern, sondern auch unter Brücken, in Ruinen oder kletternd in den geliebten Bergen. Und schon damals wurde alles mit wissenschaftlicher Genauigkeit betrieben.

Studium

Nach dem Abitur, das er mit Auszeichnung bestand, kam die Studienwahl. Elektrotechnik ließ sich mit Musik kombinieren, was zum Beruf eines „Tonmeisters" führen sollte. Das hieß volles Studium an der Technischen Universität, dazu ernsthaftes Klavier-, Tonsatz- und Gehörbildungsstudium an der Hochschule für Musik, der heutigen Universität der Künste. Morgens wurde am Uni-Flügel Klavier geübt, dann kamen die Vorlesungen, nachmittags wurde zuhause Klavier gespielt, etc. In den Ferien musste Geld verdient werden. Freunde und all die anderen Interessen – Literatur, Konzerte, Berge, Reisen – wurden nicht vernachlässigt und sogar für Diavorträge über diverse Bergtouren nahm sich der Vielbeschäftigte noch Zeit.

Das Doppelstudium wurde mit straffer Zeiteinteilung vorangetrieben, nach 8 Semestern aber war klar, dass er sich zum Tonmeister nicht berufen fühlte. Er gab das Musikstudium auf und rüstete sich zum Diplom in Elektrotechnik. Dieses Studium schloss er mit einer Diplomarbeit ab, in der er den Einschwingvorgang einer gezupften Geigensaite mathematisch untersuchte.

Bereits in seiner Studienzeit Ende der fünfziger Jahre kam Rechenberg zum ersten Mal mit Computern in Berührung. Die TU Berlin bot eine Vorlesung über die Programmierung des Zuse Z22-Rechners an, der ausschließlich in Maschinensprache programmiert werden konnte, wobei nahezu jedes Bit eines Befehlswortes eine besondere Funktion hatte. Diesen kryptischen Code zu erlernen war Rechenberg aber zu langweilig und so verlief sein erster Kontakt mit der Informatik eher enttäuschend.

Heirat

1960 heiratete Peter Rechenberg Ursula Müller. Es ist wohl kein Zufall, dass er eine Künstlerin, eine Pianistin zur Frau wählte. Seither spielt sich das Leben der beiden, wenn Beruf und Alltag es zulassen, zwischen Schreibtisch und Klavier ab. Er sitzt am Schreibtisch über einem Buch und seine Frau übt Klavier.

Industrietätigkeit bei AEG

1960 nahm Rechenberg seine erste Anstellung am Institut für Automation der AEG in Berlin an und befasste sich dort mit der Automatisierung von Walzwerken mittels mechanischer und elektrotechnischer Elemente wie Fotozellen und Relais. Obwohl noch keine Computer im Spiel waren, sah Rechenberg bereits ihr großes Potential in der Automatisierungstechnik voraus und las sich neben seiner Arbeit in die damals noch dürftige Computer-Literatur ein. Sein Wissen war bald so umfassend, dass er, junger Diplomingenieur bei der AEG, zu Vorträgen eingeladen wurde. Ein Beitrag in

einer Zeitschrift brachte ihm vom Oldenbourg Verlag München den Auftrag zu seinem ersten Buch „Grundzüge digitaler Rechenanlagen“ ein. Das Buch erschien 1964 und fand als eines der ersten deutschsprachigen Bücher zu diesem Thema viel Beachtung [Rech64].

Inzwischen hatte eine Nachbarfabrik der AEG einen Rechner des Typs Zuse Z23 gekauft und Rechenberg bekam die Gelegenheit, diesen Transistorrechner mit einigen hundert Wörtern Arbeitsspeicher, einer Magnettrommel als externes Medium und Lochstreifen für die Ein- und Ausgabe zu benutzen. In zahlreichen Nächten, in denen er die Maschine für sich allein hatte, lernte er sie in Assemblersprache zu programmieren. Obwohl von der Ausbildung her Elektrotechniker, wurde er nun immer mehr von der Softwareentwicklung gepackt.

Bald darauf wechselte er ins Forschungsinstitut der AEG und bekam die Gelegenheit, eine moderne Großrechenanlage vom Typ IBM 7040 zu benutzen, diesmal in Fortran und mit Lochkarten anstatt mit Lochstreifen – ein enormer Produktivitätsgewinn. Rechenberg wurde zum Leiter eines Programmierlabors mit drei Mitarbeitern und entwickelte Software zum Leiterplatten-Layout.

Promotion am Heinrich-Hertz-Institut

1966 fasste Rechenberg den Entschluss zu promovieren und bewarb sich bei Prof. Giloi an der TU Berlin, der sein Buch über digitale Rechenanlagen bereits kannte und ihm eine Dissertationsstelle am Heinrich-Hertz-Institut anbot. Sein Forschungsthema war die digitale Simulation, seine Arbeitsumgebung ein französischer Kleinrechner mit einem Fortran-II-Compiler.

Peter Rechenbergs Eifer, seine wissenschaftliche Genauigkeit und nicht zuletzt seine mehrjährige Berufserfahrung brachten ihm schon bald den Ruf eines Fachmanns ein, nicht nur in der Simulation, sondern auch auf dem Gebiet der Programmiersprachen. So erhielt er Lehraufträge für Vorlesungen über Fortran-Programmierung und über Prinzipien von Programmiersprachen. Die Studenten kamen in Scharen, denn es gab damals an der TU Berlin noch kaum Lehrveranstaltungen auf diesem Gebiet.

1969 schloss Rechenberg seine Dissertation ab, in der er ein auf endlichen Automaten basierendes mathematisches Modell zur Simulation stetiger Vorgänge entwickelte sowie eine Simulationssprache, die auf den besten Konzepten der damals bekannten Programmiersprachen beruhte. Dies war sein erster Kontakt mit dem Übersetzerbau und den formalen Sprachen, einem Forschungsthema, das Rechenberg während seiner gesamten wissenschaftlichen Laufbahn begleitete.

Professur an der TU Berlin

Gleich nach seiner Promotion wurde Rechenberg an der TU Berlin die Stelle eines „Akademischen Rats“ angeboten. Im Alter von 36 Jahren wurde er zum Professor ernannt. Gleichzeitig übernahm er die Leitung einer Forschungsgruppe für Übersetzerbau.

Die TU Berlin hatte inzwischen einen Rechner des Typs IBM 360/67 angeschafft, auf dem es einen PL/I-Compiler gab sowie ein Time-Sharing-System, das die Möglichkeit bot, den Rechner von einem Terminal aus statt mit Lochkarten zu

bedienen. Mittels virtueller Maschinen wurde jedem Benutzer der Eindruck vermittelt, den gesamten Rechner für sich allein zu haben.

PL/I war im Vergleich zu Fortran eine wesentlich mächtigere und komfortablere Sprache, aber auch komplexer und schwerer zu erlernen. So fasste Rechenberg den Entschluss, ein zweibändiges Buch über die Programmierung in PL/I zu schreiben und dieses Wissen auch in Lehrveranstaltungen zu vermitteln. Das Buch erschien 1974, erlebte zwei Auflagen und war ein großer Erfolg [Rech74]. Auch die Verfasser dieses Beitrags haben anhand dieses Buchs programmieren gelernt.

Rechenbergs Wirken an der Universität Linz

Politische Unruhen und die damit verbundenen schlechteren Arbeitsbedingungen an der TU Berlin veranlassten Rechenberg 1974, einen Ruf an die damals noch junge Hochschule für Sozial- und Wirtschaftswissenschaften – die heutige Johannes Kepler Universität Linz – anzunehmen. Vielleicht waren es auch die geliebten Berge, die ihm den Ortswechsel von der Großstadt Berlin in die „Alpenrepublik“ erleichterten. Im Jänner 1975 trat er seinen Dienst als ordentlicher Universitätsprofessor für Informatik an und gründete eine Abteilung für Software.

In Linz befasste sich Rechenberg unter anderem mit Softwaretechnik [Rech78, Rech83, Rech89], Algorithmen und Datenstrukturen, vor allem aber mit Übersetzerbau [Rech76a, Rech79a, Rech93] und Formalen Sprachen [Rech97a], die er in den Lehrplan der Linzer Informatik einführte und in der Forschung vertrat.

Die frühen siebziger Jahre standen im Zeichen neuer Programmiersprachen wie Pascal und C sowie neuer Syntaxanalyseverfahren. Rechenberg leistete seinen Beitrag zu diesem Thema mit der Entwicklung eines neuen Verfahrens für die sackgassenfreie Syntaxanalyse [Rech75]. Neben höheren Programmiersprachen standen aber auch die Assemblersprachen noch hoch im Kurs, denn zeitkritische Aufgaben wurden noch immer am besten maschinennahe implementiert. So entwickelte Rechenberg eine Makroassemblersprache für Rechner des Typs IBM 370, die die Kluft zwischen Assemblersprachen und höheren Programmiersprachen verkleinerte [Rech76b, Rech77].

Das Modula-2-Projekt. 1980 veröffentlichte *Niklaus Wirth* an der ETH Zürich die Definition seiner neuen Programmiersprache Modula-2, die auf Pascal basierte und Konzepte wie Modularisierung, Datenabstraktion, Nebenläufigkeit von Prozessen sowie getrennte Übersetzung von Modulen mit Typprüfung über Modulgrenzen hinweg einführte [Wirth80, Wirth84, Rech84b].

Rechenberg erkannte sofort die Bedeutung dieser neuen Konzepte für die Softwaretechnik und beantragte ein Projekt beim Österreichischen Fonds zur Förderung der wissenschaftlichen Forschung (FWF) zur Entwicklung eines Modula-2-Compilers für handelsübliche Mikrocomputer. Das Projekt wurde 1982 genehmigt, und Rechenberg nahm Kontakte zu Wirth auf, aus denen sich eine Kooperation entwickelte, die bis zum heutigen Tag nachwirkt.

Wirth hatte mit seinem Team nicht nur einen Compiler für Modula-2 geschrieben, sondern auch den Rechner Lilith samt Betriebssystem und Programmierwerkzeugen entwickelt. Lilith war für damalige Begriffe revolutionär, denn diese Maschine besaß einen hochauflösenden Bildschirm, eine Maus, eine Wechselplatte sowie einen Ethernet-Anschluss, über den auch ein Laserdrucker angesprochen werden konnte

[Wirth81, Pom85]. Dies war in Zeiten, in denen weder der IBM-PC noch der Macintosh existierte, mehr als beachtlich. Lilith besaß auch einen äußerst kompakten Maschinencode, ähnlich dem heutigen Bytecode von Java. Auf Grund dieses kompakten Maschinencodes kam Wirths Modula-2-Compiler trotz des geringen Hauptspeichers von 128 Kilobyte mit nur vier Pässen aus.

Rechenberg plante, seinen Modula-2-Compiler für damals handelsübliche Mikrocomputer zu entwickeln, die meist mit Intel-Prozessoren ausgestattet waren. Das Institut in Linz besaß einen derartigen Rechner mit einem Intel 8080-Prozessor und 64 Kilobyte Speicher (später dann mit einem 8085-Prozessor und 128 Kilobyte Speicher). Auf Grund des geringen Speichers wurde beschlossen, den Compiler in sieben anstatt in vier Pässe zu zerlegen. Der erste Pass führte die lexikalische Analyse durch und schrieb einen numerischen Symbolstrom auf eine Zwischendatei. Diese wurde im zweiten Pass gelesen, der die syntaktische Analyse durchführte und wieder eine Zwischendatei erzeugte. Weitere Pässe führten dann eine Namenanalyse, eine Typanalyse sowie weitere Umformungen durch, bis schließlich Code erzeugt wurde, der dann von einem Interpreter ausgeführt werden konnte.

Der Compiler wurde zwar in dieser Form fertig gestellt [RPR84], aber er war für den praktischen Einsatz zu langsam. Die vielen Pässe und das ständige Schreiben und Lesen von Zwischendateien kosteten zu viel Zeit. Inzwischen hatte sich auch die Technik weiterentwickelt und es standen Rechner mit leistungsfähigeren Prozessoren und größerem Speicher zur Verfügung.

Wirth hatte außerdem in Zürich seinen eigenen Modula-2-Compiler vereinfacht, so dass er statt vier Pässen nur noch einen einzigen Pass benötigte. Das Linzer Compiler-Projekt wurde daher nochmals von vorne begonnen und Wirths Compiler wurde so adaptiert, dass er Maschinencode für die IBM/370 erzeugte, die damals in der Informatik-Lehre an der Universität Linz eingesetzt wurde. Diesmal wurde das Projekt ein voller Erfolg. Der Compiler war pfeilschnell und robust. Er war der erste Modula-2-Compiler auf einem IBM-Großrechner und wurde in Linz und an anderen Universitäten bis zum Ende der 80er-Jahre in der Programmierausbildung eingesetzt.

Obwohl das Ergebnis des Linzer Modula-2-Projekts im ersten Anlauf enttäuschend war, hatte es doch für das Institut einen positiven Effekt. Nicht nur wurden die Compilerbau-Techniken des Sprachpioniers Wirth studiert und in der Lehre übernommen; das Projekt schweißte auch Rechenbergs Team zusammen: Jeder Mitarbeiter hatte – geleitet durch Rechenbergs Erfahrung – seine ganz besondere Aufgabe in diesem Projekt, und die Verfasser dieses Beitrags erinnern sich noch heute gerne an die aufregende „Modula-Zeit“ zurück.

Kontextbedingungen von Modula-2. Das Modula-2-Projekt brachte noch ein weiteres Ergebnis, das aber leider nur zu einem internen Bericht und nicht zu einer öffentlichen Publikation führte: Rechenberg entwickelte ein halbformales Schema zur Beschreibung der Kontextbedingungen einer Programmiersprache, also jener Regeln, die etwas über die semantische Korrektheit eines Programms dieser Sprache aussagen [Rech85].

Während die Syntax von Programmiersprachen seit 1960 mit Hilfe der Backus-Naur-Form (BNF) formal und einfach beschrieben werden kann, gibt es zur Beschreibung der Semantik bis heute noch kein einfaches formales Verfahren. Methoden wie die denotationale Semantik [Tenn76] oder die VDM (Vienna Development Method) [Bjør78] sind so kompliziert und fehleranfällig, dass man es in

der Praxis immer noch vorzieht, die Semantik einer Sprache verbal zu beschreiben, was oft zu Ungenauigkeiten oder gar zu Mehrdeutigkeiten führt.

Rechenbergs Idee war es, die Kontextbedingungen an die Syntaxregeln zu knüpfen. Für jede Syntaxregel eines Sprachkonstrukts wird eine nummerierte Liste von Bedingungen angegeben, die erfüllt sein müssen, damit das Sprachkonstrukt semantisch korrekt ist. Die einzelnen Bedingungen werden zwar in Prosa formuliert, aber der Zwang, sie für jede Syntaxregel explizit aufzuschreiben, führt dazu, dass nichts vergessen wird, und dass Programmierer und Compilerbauer sofort wissen, wo sie nachschlagen müssen, wenn sie die Semantik eines Sprachelements verstehen wollen. Die in den Kontextbedingungen verwendeten Begriffe wie „imported“ oder „declared“ werden in einem gesonderten Abschnitt definiert und zwar wieder durch eine nummerierte Aufzählung von Bedingungen. Hier ist ein Beispiel der Kontext-bedingungen für Prozedurdeklarationen:

ProcedureDeclaration = PROCEDURE $ident_1$ [FormalParams] ";" Block $ident_2$.

1. $ident_1$ and $ident_2$ must be the same.
2. If the procedure declaration is directly contained in an implementation module, $ident_1$ may already have been defined as procedure identifier in the corresponding definition module. In this case the correspondence rules for definition and implementation modules apply (see ...).
3. In all other cases $ident_1$ must not be *imported* and not be *declared* before in the current block.
4. If the procedure is a function then the block must contain at least one return statement.

Rechenbergs System der halbformalen Beschreibung von Kontextbedingungen ist ein schönes Beispiel für sein Bestreben, Formalität und Praxistauglichkeit unter einen Hut zu bringen.

Tabellengesteuerte Topdown-Syntaxanalyse. In den siebziger Jahren waren die Hauptspeicher klein und man versuchte daher, möglichst platzsparende Programme zu schreiben, was im Übersetzerbau für die tabellengesteuerte Syntaxanalyse sprach. Rechenberg entwickelte den sogenannten *Topdown-Graphen*, eine Datenstruktur zur Speicherung von Grammatiken für die tabellengesteuerte Topdown-Syntaxanalyse [ReMö85]. Ein universelles Analyseprogramm von wenigen Zeilen Länge kann an Hand eines solchen Graphen die syntaktische Korrektheit eines Quellprogramms feststellen.

Im Topdown-Graphen wird jede Regel einer Grammatik als Teilgraph modelliert, wobei die Syntaxsymbole die Knoten bilden und durch zwei Arten von Kanten miteinander verbunden sind: eine Kante nach rechts verbindet aufeinander folgende Symbole, während eine Kante nach unten zur nächsten Alternative führt. Abb. 1 zeigt ein Beispiel eines Topdown-Graphen für eine einfache Grammatikregel.

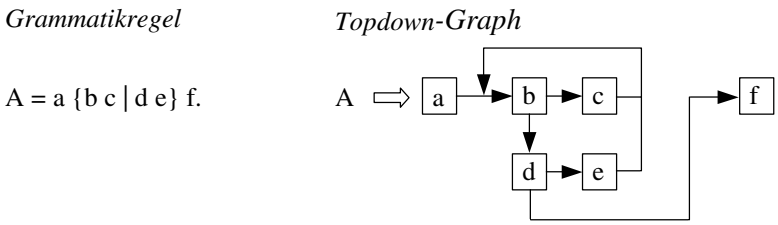

Abb. 1 *Topdown-Graph für eine Grammatikregel*

Der Graph wird als Tabelle abgelegt und von einem universellen Syntaxanalysator interpretiert. Tritt bei der Analyse ein Fehler auf, so wird aus den Topdown-Graphen aller Regeln, an denen der Syntaxanalysator gerade arbeitet, die Menge jener Symbole berechnet, mit denen die Analyse nach der Fehlerstelle wieder fortgesetzt werden kann. All das geschieht automatisch, also ohne Zutun des Übersetzerbauers.

Attributierte Grammatiken. Attributierte Grammatiken wurden 1968 von *Donald Knuth* zur Beschreibung von Übersetzungsprozessen entwickelt [Knuth68]. Jedes Syntaxsymbol kann dabei Attribute besitzen, die semantische Eigenschaften des Symbols festhalten (z.B. die Adresse einer Variablen oder den Typ eines Ausdrucks). Semantische Aktionen berechnen die Attribute aus den Werten anderer Attribute und lösen Übersetzungsaktionen aus.

Während Knuth attributierte Grammatiken als deklarative Beschreibungen sah, die nur die Abhängigkeiten zwischen Attributen, aber nicht die Reihenfolge ihrer Berechnungen festlegte, fasste Rechenberg attributierte Grammatiken als prozedurale Beschreibungen auf [Rech79b, Rech84a]. Eine Grammatikregel wird als Erkennungsprozedur betrachtet, die von links nach rechts abgearbeitet wird und dabei die Symbole dieser Regel im Quelltext erkennt. Attribute werden als Parameter der Symbole aufgefasst, die bei der Erkennung eines Symbols entstehen (Ausgangsattribute) oder der Erkennung eines Symbols von außen mitgegeben werden (Eingangsattribute). Semantische Aktionen sind beliebige Codestücke, die während der Syntaxanalyse an der Stelle ausgeführt werden, an der sie in der Grammatik stehen.

Folgendes Beispiel zeigt eine attributierte Grammatikregel zur Verarbeitung von Deklarationen einer Programmiersprache. Attribute werden dabei zwischen spitze Klammern geschrieben und semantische Aktionen zwischen "(." und ".)". Zur besseren Lesbarkeit sind die Syntaxteile hier fett hervorgehoben.

```
VarDeclaration          (. TypeDescriptor typ; String name; .)
= Type<out typ>
  ident<out name>       (. SymbolTable.Enter(name, typ); .)
  {ident<out name>      (. SymbolTable.Enter(name, typ); .)
  } ";".
```

Während Rechenberg attributierte Grammatiken anfangs nur zur Dokumentation von Übersetzungsprozessen verwendete, entstand bald die Idee, sie in einen Syntaxanalysator zu übersetzen. So entstand der Compilergenerator Coco, der aus einer attributierten Grammatik einen Syntaxanalysator und einen lexikalischen Analysator erzeugt. Der Übersetzerbauer muss nur noch semantische Routinen (z.B. *SymbolTable.Enter*) hinzufügen, um einen vollständigen Compiler zu erhalten.

Coco wurde 1985 in einem Buch beschrieben [ReMö85] und in den Folgejahren in Linz [DoPi90] sowie an der ETH Zürich [Möss90] erweitert. Heute wird Coco weltweit von Firmen und Universitäten eingesetzt, und es gibt sogar Bücher anderer Autoren, die auf Coco basieren [Terry97].

Analysatoren und Übersetzungswerkzeuge. Der Compilergenerator Coco zusammen mit einem Generator für lexikalische Analysatoren (Alex) bot die Möglichkeit, Werkzeuge für die Quelltext-Analyse und die Übersetzung einer höheren Programmiersprache in eine andere zu realisieren.

So entstanden Analysatoren für Ada-Quelltexte, Werkzeuge zur statischen und dynamischen Programmanalyse sowie Übersetzer zwischen höheren Programmiersprachen (z.B. von Modula-2 nach PL/M, C und Ada).

Rechenberg leitete von 1991 bis 1993 ein vom Fonds zur Förderung der wissenschaftlichen Forschung (FWF) gefördertes Projekt, bei dem es um die Übersetzung des ganzen Sprachumfangs von Fortran 77 (inklusive Ein/Ausgabe) nach Ada ging. Hier kamen nicht nur die oben erwähnten Werkzeuge und Techniken (attributierte Grammatiken) zum Einsatz, sondern auch zwei rechnerinterne Zwischensprachen (eine für Fortran und eine für Ada), wobei der eigentliche Übersetzungsprozess auf diesen Zwischensprachen definiert war [Dob94].

Diagrammgeneratoren. Zur Dokumentation und zum Entwurf von Programmen wurden schon früh grafische Notationen entwickelt, die den Steuerfluss von Programmen klarer hervortreten lassen. Im Zusammenhang mit der strukturierten Programmierung entstanden zum Beispiel in den siebziger Jahren die nach ihren Erfindern benannten *Nassi-Shneiderman-Diagramme* (oder Struktogramme) [NaSh73], in welchen Anweisungen durch grafische Bausteine dargestellt werden, die modular zusammengesetzt werden können.

Diagramme haben den Nachteil, dass sie aufwendig zu zeichnen und noch aufwendiger zu ändern sind. Aus diesem Grund wurde unter Rechenbergs Leitung und in Kooperation mit der Firma Siemens der Diagrammgenerator *Struct* entwickelt, der aus einem Quellprogramm automatisch Struktogramme erzeugte. Da Struktogramme keine unstrukturierten Anweisungsarten (z.B. Sprünge) gestatten, konnte mit Struct auch überprüft werden, ob ein Programm den Regeln der strukturierten Programmierung gehorchte. Die Eingabe eines Algorithmus sollte interaktiv erfolgen. Fehlerhafte Eingaben sollten sofort gemeldet und vom Benutzer korrigiert werden. Das erforderte ein Zurücksetzen des Analyseprogramms zur letzten fehlerfreien Stelle und eine Fortsetzung mit der neuen Eingabe. Mit herkömmlichen Techniken war das nicht möglich, aber Rechenbergs Topdown-Graphen führten zu einer einfachen und eleganten Lösung.

Als Fortsetzung dieser Arbeit entstand an Rechenbergs Abteilung der Generator *Moses* [Bla87], der Programme als Ablaufdiagramme darstellte und auch die interaktive Bearbeitung in dieser Form erlaubte. Im Gegensatz zu Struct benutzte Moses eine Ablaufdiagramm-Darstellung, die Rechenberg Jahre zuvor für die interne Verwendung und für den Einsatz in der Lehre erfunden hatte.

Komplexitätsanalyse. Zur Qualitätssicherung von Software möchte man oft die Komplexität von Programmen messen, um zum Beispiel sicherzustellen, dass die Komplexität des entwickelten Codes unterhalb einer bestimmten Schranke bleibt.

Ein einfaches Maß für die Komplexität eines Programms ist die Anzahl seiner Quellzeilen oder Anweisungen. Dieses Maß gewichtet jedoch einfache Zuweisungen genauso hoch wie komplexe Schleifen und kann daher nur als grober Schätzwert dienen. Aufwändigere Maße wie das von *McCabe* [McCa76] oder *Halstead* [Hals77] liefern zwar genauere Aussagen, sind aber immer noch zu grob. Rechenberg entwickelte daher ein neues und genaueres Maß, das zwischen Anweisungskomplexität, Ausdruckskomplexität und Datenkomplexität unterscheidet [Rech86].

Zur Berechnung der *Anweisungskomplexität* werden die Anweisungen eines Programms gezählt, wobei jede Anweisung ein vom Benutzer definierbares Gewicht bekommt. Verzweigungen können zum Beispiel das dreifache Gewicht einer Zuweisung tragen. Geschachtelte Anweisungen werden zusätzlich mit dem Faktor 1.5 gewichtet, da die Schachtelung zusätzliche Komplexität bewirkt.

Zur Berechnung der *Ausdruckskomplexität* werden alle Operatoren in Ausdrücken gezählt, wobei wiederum benutzerdefinierbare Gewichte benutzt werden und jede Klammerung die Komplexität des geklammerten Ausdrucks um den Faktor 1.5 erhöht.

Bei der *Datenkomplexität* wird schließlich die Art der verwendeten Variablen berücksichtigt. Lokale Variablen werden als weniger komplex erachtet als globale Variablen oder Variablen, die in statisch umgebenden Prozeduren deklariert sind.

Die Gewichte können vom Benutzer nach Erfahrungswerten vergeben werden, was es auch erlaubt, firmenspezifische Richtlinien zu berücksichtigen, in denen gewisse Anweisungsarten weniger gerne gesehen werden als andere. Neben der absoluten Komplexität, bei der die gewichteten Einzelkomplexitäten addiert werden, kann auch die relative Komplexität ermittelt werden, indem man die absolute Komplexität durch die Anzahl der Anweisungen des Programms dividiert.

Wenn ein Programmstück zu hohe Komplexität aufweist, kann man das umgehen, indem man es in kleine Module zerlegt. Dadurch sinkt zwar die innere Komplexität der Module, dafür werden die Zusammenhänge zwischen ihnen immer komplexer. In einer weiteren Studie wurde daher eine Reihe von Maßen entwickelt, mit denen man über die algorithmische Komplexität hinaus auch die Komplexität der Grobstruktur in Zahlen fassen und damit beurteilen und vergleichen kann [Bla85].

Prototyp-basierte Sprache Omega. In den 80er-Jahren wurde die objektorientierte Programmierung populär, bei der Programme in Objekte zerlegt werden, die als Sammlung von Daten und den darauf anwendbaren Operationen aufgefasst werden.

Während die meisten objektorientierten Sprachen gleichartige Objekte zu Klassen zusammenfassen, die als Typen zur Deklaration neuer Objekte dienen, gibt es auch den Ansatz der prototyp-basierten Sprachen, bei denen Objekte zur Laufzeit aus neuen Daten und Operationen zusammengebaut werden können. Hat man ein passendes Objekt konstruiert, kann man es als Prototyp verwenden, um daraus beliebig viele gleichartige Objekte als Kopien zu erzeugen.

An Rechenbergs Abteilung wurde in Linz die prototyp-basierte Sprache *Omega* entwickelt [Bla94], die anschließend auch in der Lehre eingesetzt wurde. Omega vereint die Vorteile zweier Familien objektorientierter Sprachen: Klassenbasierte Sprachen sind in der Regel typisiert und erlauben daher die Erkennung vieler Programmierfehler zur Übersetzungszeit. Prototypbasierte Sprachen sind sehr einfach, aber meist untypisiert. Omega war die erste prototypbasierte Sprache mit Typisierung.

Omega ist ein hübsches Beispiel für die Fortführung der Rechenbergschen Denkschule durch seine Schüler: Im Vordergrund standen die Sprachkonzepte, deren Wahl durch das Streben nach Einfachheit und die Anforderungen der Praxis bestimmt wurde.

Als Ergebnis der Arbeit entstand neben der Sprache ein Compiler, der in eine Entwicklungsumgebung eingebettet ist und mit dem Laufzeitsystem eng zusammenarbeitet. Neue Prototypen, Methoden und Variablen werden interaktiv erzeugt, und während des Eintippens einer Methode werden die vom Laufzeitsystem verwalteten Objekte aktualisiert. Nach einer Änderung einer Methode kann das Programm sofort gestartet werden. Wartezeiten durch Neuübersetzungen entfallen dadurch.

Software-Versionsverwaltung. Anfang der 90er-Jahre widmeten sich Rechenberg und sein Team der Versionsverwaltung von Software. Fast jedes größere Programm durchläuft im Laufe seines Lebens mehrere Versionen, die oft gleichzeitig existieren und mittels Werkzeugen auseinandergehalten werden müssen. Dabei unterscheidet man zwischen Revisionen eines Programms, bei denen es sich um zeitlich aufeinander folgende Versionen handelt, die durch Weiterentwicklung entstehen, und Varianten, die gleichzeitig existierende Versionen desselben Programms (z.B. für verschiedene Betriebssysteme oder Kunden) darstellen.

Im Rahmen einer Dissertation wurden neue Verfahren für die Software-Versionsverwaltung entwickelt [Rei93]. Es wurde eine Alternative zur bisher üblichen baumartigen Organisation von Varianten und Revisionen gefunden, wobei Varianten und Revisionen als zueinander orthogonal angesehen wurden. Dadurch konnte ein besonders hohes Maß an Einfachheit und Übersichtlichkeit erzielt werden.

Ferner wurde ein Algorithmus entwickelt, der die Deltaspeicherung beliebiger Dateien (d.h. die platzsparende Speicherung verschiedener Versionen) ermöglichte [Rei91]. Der neue Algorithmus war nicht nur allgemeiner und schneller als die bekannten Algorithmen (die nur Textdateien verarbeiten konnten), sondern lieferte auch kleinere Deltas. Das Verfahren konnte in Industrie-Projekten erfolgreich angewandt werden und wurde schließlich von der Firma *Atria* zur Integration in ihr Produkt *Clearcase* lizenziert.

Zum Beweis der praktischen Anwendbarkeit der neu entwickelten Verfahren wurde das Versionsverwaltungswerkzeug *VOODOO* (Versions of Outdated Documents Organized Orthogonally) entwickelt. Das Werkzeug weist eine grafische Benutzerschnittstelle auf und zeigt somit auch, wie man moderne Konzepte der Mensch-Maschine-Kommunikation nutzbringend bei der Versionsverwaltung einsetzen kann. VOODOO war so erfolgreich, dass es später zum Produkt weiterentwickelt wurde und zur Gründung einer Startup-Firma führte. Die Arbeiten und das Produkt VOODOO wurden bereits mehrmals national und international ausgezeichnet: Innovationspreis des Landes OÖ, MacWorld Editors' Choice Award (USA), Software-Engineering-Preis der Ernst-Denert-Stiftung (Deutschland).

Kategorisierung der Informatik. Peter Rechenberg hat die Entwicklung der Informatik seit ihren Anfängen miterlebt. In den vierzig Jahren seiner beruflichen Laufbahn hat sich diese Disziplin von bescheidenen Anfängen zu einer Technologie entwickelt, die heute in alle Lebensbereiche vordringt. Dabei hat sie sich in so viele

Zweige aufgespaltet, dass es selbst Fachleuten kaum noch möglich ist, alles zu überblicken und zu beherrschen.

Rechenbergs wissenschaftliche Arbeit war dadurch gekennzeichnet, dass er Wissen sammelte, ordnete und in leicht verständliche Form brachte. So ist es nicht verwunderlich, dass er 1991 ein Buch mit dem Titel *Was ist Informatik?* schrieb [Rech91a], in dem er für interessierte Laien darlegt, worum es in der Informatik geht und was diese Disziplin in der Lage ist zu leisten. Das Buch ist eine der besten Einführungen in dieses inzwischen äußerst komplexe Gebiet.

Viel ambitionierter noch ist aber das *Informatik-Handbuch* [RePo99], das er als Hauptherausgeber bis heute betreut, und in dem er sich zum Ziel gesetzt hat, das gesamte Wissen der Informatik in knapper und verständlicher Form für Fachleute zusammenzufassen. In der jungen Disziplin der Informatik gibt es zumindest im deutschsprachigen Raum kein anderes Standardwerk, das ähnliche Breite und Tiefe aufweist.

Neben den segensreichen Erfindungen der Informatik gab es aber auch zahlreiche Modeströmungen und Schlagwörter, die Wichtigkeit vortäuschten, aber wenig Substanz aufwiesen. Rechenberg war diesen Trends gegenüber stets kritisch eingestellt und hat dies auch in Veröffentlichungen niedergelegt [Rech91b, Rech97b, Rech99].

Wirken als akademischer Lehrer in Linz

Die persönlichen Erfahrungen Rechenbergs in der Informatik reichen zurück bis in die Gründerzeit. Sein eigener Werdegang als Informatiker spiegelt sich daher auch in seinen Lehrveranstaltungen wider. In den Anfangsjahren an der Johannes Kepler Universität lehrte Rechenberg noch die Programmierung in der Assemblersprache des Großrechners IBM/360. Er verfolgte die Entwicklung neuer Programmiersprachen und Programmiermethoden wachsam und kritisch und baute neue Entwicklungen in seine Vorlesungen ein. Dabei standen nicht die syntaktischen Feinheiten der Sprachen im Vordergrund, sondern die in ihnen steckenden Konzepte. Im Laufe der Jahre behandelte Rechenberg in mehreren Seminaren Sprachen für verschiedene Spezialaufgaben, wie z.B. funktionale und logische Sprachen, sowie Sprachen zur Mustererkennung und Skriptsprachen. Er vertrat die Ansicht, dass eine Programmiersprache das Denken eines Softwareentwicklers prägt. Das führte schließlich zu einer eigenen Vorlesung *Programmiersprachen als Denkmodelle* [Rech90], in der er seine gesammelten Erfahrungen an die Studenten weitergeben konnte.

Wenn eine Sprache oder Sprachfamilie in der Lehre behandelt werden sollte, gab sich Rechenberg nicht mit der Vermittlung von Lehrbuchwissen zufrieden, sondern versuchte, die Sprachkonzepte und ihre Funktionsweise selbst zu ergründen. Bei Bedarf legte er selbst Hand an und entwickelte beispielsweise kleine Interpreter für logische und funktionale Programmiersprachen, mit denen er die Funktionsweise veranschaulichen und Experimente mit Beispielprogrammen durchführen konnte. Als Ergebnis dieser Studien entstanden institutsinterne „Technische Notizen“, in denen er komplizierte Sachverhalte einfach und verständlich darstellte. In verdichteter und didaktisch aufbereiteter Form wurden diese Notizen dann in die Lehrinhalte eingearbeitet.

Die Entwicklung der Informatik zu immer größeren Programmen hin fand ebenfalls ihren Niederschlag in den Lehrinhalten. In der Vorlesung *Algorithmen* vermittelte Rechenberg das „Programmieren im Kleinen“, und später kam eine neue Vorlesung *Softwaretechnik* hinzu, in der er die Probleme behandelte, die bei der Entwicklung großer Programme auftreten.

Neben der Anwendung der Programmiersprachen behandelte Rechenberg in der Lehre vor allem die Übersetzung von Programmen. Er führte an der Universität Linz die Vorlesungen *Formale Sprachen* und *Übersetzerbau* ein, in denen die Grundzüge des Compilerbaus vermittelt wurden. Später kam als weiterführende Vorlesung *Übersetzerbau 2* hinzu, in der unter anderem Methoden zur Optimierung der Codeerzeugung behandelt wurden.

Da Rechenberg im Verlauf seiner beruflichen Tätigkeit viele Trends in der Informatik kommen und manche davon wieder in der Versenkung verschwinden sah, stand er neueren Entwicklungen stets kritisch gegenüber. Er trennte klar zwischen Modeerscheinungen und den „ewigen Werten“, deren Kenntnis den Studenten auch in zehn Jahren noch gute Dienste leisten würde. Das zeigt sich beispielsweise in der Algorithmen-Vorlesung, für die Rechenberg eine eigene Algorithmenbeschreibungssprache namens „Adele“ (für Algorithm Description Language) erfand. Adele orientierte sich an bestehenden Sprachen wie Pascal, Modula-2 und Ada, beschränkte sich jedoch auf das Wesentliche und bot Freiheiten, die für die schrittweise Entwicklung von Algorithmen (an der Tafel ebenso wie im wissenschaftlichen Alltag) hilfreich waren. Die Verwendung von Adele erlaubte die Konzentration auf die Kernelemente der Algorithmen und zwang die Studenten zu abstraktem Denken.

Die Gründlichkeit und systematische Arbeitsweise Rechenbergs prägten auch seine Lehrveranstaltungen. Da wurde nichts dem Zufall überlassen: Jede einzelne Vorlesung wurde minutiös geplant. Die Reihenfolge der Themen wurde sorgsam gewählt, für jeden Abschnitt wurden Zeitabschätzungen vorgenommen, die im Laufe der Jahre immer mehr verfeinert wurden. Diese Zeitplanung war unter anderem die Grundlage für begleitende Übungen, deren Inhalte mit jenen der Vorlesung synchronisiert werden mussten. Besonders wichtig war das für die Vorlesungen Formale Sprachen und Übersetzerbau, die im selben Semester parallel abgehalten wurden. Rechenberg stellte auf diese Weise sicher, dass die Inhalte dieser beiden Lehrveranstaltungen wochenweise aufeinander abgestimmt waren.

Für sein Vorlesungsmanuskript fand Rechenberg eine Gliederung, die – wie so viele seiner Ideen – bestechend einfach und gerade deswegen leicht und universell anwendbar war: Die Vorlesungsunterlagen bestanden aus zwei Spalten: die linke Spalte enthielt das Tafelbild, in der rechten Spalte standen stichwortartige Anmerkungen zum gesprochenen Text und didaktische Hinweise. Dieses Schema hat sich so bewährt, dass es heute auch von seinen ehemaligen Mitarbeitern noch eingesetzt wird.

Rechenberg verzichtete bewusst darauf, ein vollständiges Skriptum seiner Vorlesungen herauszugeben. Die Studenten sollten sich die Inhalte während der Vorlesungen selbst erarbeiten und sich ihre eigenen Notizen und Gedanken dazu machen. Statt eines Skriptums gab es so genannte „Umdrucke“; das waren Ergänzungen und Vertiefungen zum Vorlesungsstoff, die den Studenten weiterführende Einsichten vermitteln sollten.

Es gehörte zu Rechenbergs Arbeitsstil, sich vor jeder Vorlesungsstunde etwa eine halbe Stunde Zeit für die Vorbereitung zu nehmen. In dieser Zeit durfte er nicht gestört werden, denn die Weitergabe von Wissen war etwas, was Rechenberg nicht

auf die leichte Schulter nahm. Auch nach den Vorlesungen hielt Rechenberg seine Gedanken und Erfahrungen fest: Was hat gut geklappt, wo hatten die Studenten Verständnisschwierigkeiten, was hat sie gelangweilt, was sollte im nächsten Jahr anders gemacht werden? Auf diese Weise wurden die Vorlesungen von Jahr zu Jahr überarbeitet, ergänzt und gelegentlich entrümpelt.

Zu Rechenbergs Verständnis von Qualitätssicherung gehörten neben seinen persönlichen Notizen auch Fragebögen, die am Ende jedes Semesters ausgeteilt und von den Studenten beantwortet wurden. Die Studenten konnten darin anonym und daher hemmungslos Kritik an den Lehrveranstaltungen üben und Verbesserungsvorschläge anbringen. Die Ergebnisse dieser Befragungen wurden jedes Mal gewissenhaft ausgewertet und fanden wiederum ihren Niederschlag in der Vorlesung für das nächste Studienjahr. Meist waren jedoch nur wenige Änderungen erforderlich, denn ein Großteil der Studenten bescheinigte ihm einen hervorragenden Lehrstil. Bemerkenswert an dieser Fragebogenaktion ist, dass sie an Rechenbergs Abteilung schon lange üblich war, bevor an der Universität Linz die Evaluation von Lehrveranstaltungen allgemein eingeführt wurde. Dass diese ständige kritische Beurteilung der eigenen Lehrveranstaltungen letztlich von den Studenten honoriert wird, zeigt sich am Beispiel einer Absolventenbefragung, die im Jahr 2002 durchgeführt wurde. Rechenberg wird darin von einem Informatik-Absolventen in einem persönlichen Kommentar erwähnt: *„Besonders hervorheben möchte ich die didaktische/fachliche Leistung von Prof. Rechenberg. Dieser sollte der Prototyp eines Hochschullehrers sein!“*

Rechenberg versuchte stets, seine eigenen didaktischen Erfahrungen auch an Studenten weiterzugeben. Vor allem in Seminaren – insbesondere im Diplomandenseminar – nahm er daher nicht nur die technischen Inhalte, sondern auch deren Darstellung kritisch unter die Lupe. Er verabscheute die übermäßige Verwendung von Fachjargon und durchschaute jeden Versuch, Eindruck zu schinden. So wie er selbst jeden Begriff hinterfragte und mit einfachen Worten verständlich zu erklären versuchte, machte er den Studenten klar, worauf es wirklich ankommt: einfache und angemessene Darstellung in Wort und Bild, ein erkennbarer „roter Faden“ und die Konzentration auf das Wesentliche.

Rechenbergs feines Sprachgefühl und sein gezielter Einsatz von treffenden Begriffen macht jede seiner Vorlesungen und jede schriftliche Ausarbeitung zu einem sprachlichen Meisterwerk. Da sitzt jedes Wort, und komplizierte Sachverhalte werden dem Zuhörer oder Leser einfach und schlüssig dargestellt. Die Regeln dafür sind einfach, aber für Studenten oft schwierig zu befolgen. Rechenberg bot daher ein eigenes Seminar zum Thema *Technisches Schreiben* an, dessen Inhalte er später auch in seinem gleichnamigen Buch [Rech02] behandelte.

Rechenbergs Funktion als Lehrer beschränkte sich allerdings nicht auf seine Lehrveranstaltungen; auch seine Mitarbeiter profitierten von Rechenbergs bescheidener Art, Wissen weiterzugeben. Es gehörte zur Arbeitskultur an seiner Abteilung, dass jedes Werk allen Kollegen mit der Bitte um Kritik vorgelegt wurde. Diese ungeschriebene Regel galt für den „Chef“ ebenso wie für den Studienassistenten. Die korrigierten Exemplare strotzten oft geradezu vor Verbesserungsvorschlägen mit Rotstift; Rechenbergs Anmerkungen wurden zudem meist durch ein getrenntes Dokument ergänzt, in dem er einzelne Punkte ausführlicher kommentierte und Verbesserungen anregte. Oft führte dann die gemeinsame Durchsicht der Werke zu angeregten fachlichen und philosophischen Diskussionen, aus denen alle Beteiligten mit dem Gefühl davongingen, etwas dazugelernt zu haben.

Insgesamt vermittelte Rechenberg sein Wissen weniger durch Belehrungen als durch seine Vorbildfunktion: In seinem Bestreben nach Einfachheit und Gründlichkeit zeigte er täglich, wie wissenschaftliche Arbeit und die didaktische Aufbereitung von Wissen seiner Meinung nach aussehen sollten. Kein Wunder, dass dieses Vorbild auf alle Mitarbeiter in vielfacher Weise abfärbte. Rechenbergs ehemalige Mitarbeiter fühlen sich auch nach Jahren noch durch ihren gemeinsamen Lehrer verbunden und setzen sein Werk in der universitären Lehre ebenso wie in der Praxis fort.

Arbeitsstil

Hinsichtlich seines Arbeitsstils war und ist Peter Rechenberg für viele ein unerreichtes Vorbild. Disziplin, Bescheidenheit, Wissensdurst, Liebe zum Beruf und Ehrfurcht vor der Kunst blieben sein ganzes Leben hindurch stabil. Sein Arbeitsstil ist gekennzeichnet durch konsequente Zeiteinteilung und eiserne Selbstdisziplin, durch eine bewundernswerte Intensität und Systematik in der Auseinandersetzung mit dem jeweiligen Forschungs- oder Arbeitsgegenstand und durch die standhafte Weigerung, sich dem Zug der Zeit zu unterwerfen, dass alles immer schneller, dafür notgedrungen schlampiger zu geschehen hat. Seine Kunst der Zeiteinteilung bewirkt, dass er niemals Leerlauf hat, nicht einmal im Ruhestand.

Bewundernswert sind auch seine Bescheidenheit und sein Humor. Er drängt sich nie in den Vordergrund. Er macht auch nie, wie im Wissenschaftsbetrieb nicht selten der Fall, aus Mücken Elefanten, sondern geht konsequent mit vornehmer Zurückhaltung seinen Weg.

Wirken als Vorgesetzter

Der „Institutsbetrieb“ war geprägt durch den persönlichen Lebens- und Arbeitsstil des Professors und das bedeutete, dass der „Chef“ auch an seine Mitarbeiter hohe Anforderungen im Hinblick auf Zuverlässigkeit, Pünktlichkeit, Selbstdisziplin, konsequente Zeiteinteilung und Ergebnisqualität stellte. Wenn es auch nicht immer leicht war, diese Anforderungen einigermaßen zu erfüllen, empfanden seine Mitarbeiter dies meist schon nach einer kurzen Eingewöhnungsphase nicht als eine Last, sondern als besonders günstiges Umfeld zur Persönlichkeitsentwicklung.

Jeden Montag Nachmittag lud Rechenberg seine Mitarbeiter zur Teestunde ein – heute würde man fantasielos „Jour Fixe“ dazu sagen. Bei Tee und Kuchen wurde aber nicht nur Dienstliches besprochen, im Gegenteil, die Teestunde bot gleichermaßen Raum für wissenschaftlichen Disput und für einen kultivierten Gedankenaustausch, der weit über das Fachliche hinausging. Und auch das Spiel kam nicht zu kurz. So mancher Mitarbeiter verdankt seine Leidenschaft für „Go“ der Begeisterung des Chefs für das japanische Brettspiel.

Professor Rechenberg war ein Vorgesetzter, zu dem man aufblicken konnte, kompetent, gebildet, gerecht und integer, ein Vorbild und Mentor, der es stets verstand, eigenverantwortliches und angeordnetes Handeln seiner Mitarbeiter im Gleichgewicht zu halten, ein Balanceakt, der nicht vielen Vorgesetzten gelingt.

Akademische Funktionen

Rechenberg sah sich selbst vor allem als Wissenschaftler und akademischer Lehrer und nicht als Funktionär und Manager. Trotzdem leistete er, als man ihn rief, seinen Beitrag zur akademischen Verwaltung der Universität Linz. Von 1983 bis 1985 war er Dekan der Technisch-Naturwissenschaftlichen Fakultät und von 1995 bis 1997 Fakultätsvorsitzender. Von 1981 bis 1983 stand er der Studienkommission Informatik vor. Als Leiter und Mitglied diverser Fakultätskommissionen, als Vorsitzender zweier Berufungskommissionen und mehrerer Habilitationskommissionen hat er die Linzer Informatik und die Technisch-Naturwissenschaftliche Fakultät der Universität Linz mit geprägt.

Besondere Höhepunkte waren seine Reden als Dekan bei den Sponsionen. Er wollte den jungen Menschen mehr mitgeben als die üblichen Floskeln. Die Titel waren: *Die Idee der Universität* (1984), *Gehe Deinen Weg* (1984), *Universitäre Ausbildung zwischen Theorie und Praxis* (1985), *Gefahren des Technisch-Naturwissenschaftlichen Denkens* (1985), *Vom Gleichgewicht* (1986).

Emeritierung

Am 30. 9. 1999 wurde Peter Rechenberg nach 25 Dienstjahren an der Universität Linz emeritiert. Seitdem widmet er sich vor allem der Erweiterung des Informatik-Handbuchs, das inzwischen seine dritte Auflage erlebt hat. Sein neuestes Buch trägt den Titel "Technisches Schreiben" [Rech02]. In ihm behandelt Rechenberg eines der Gebiete, auf denen er Meister ist: er zeigt, wie man komplizierte Inhalte klar und einfach beschreibt.

Neben der Informatik beschäftigt sich Rechenberg nun wieder vermehrt mit Musik und Literatur. Er hält Vorträge über immer neue Themen, über Komponisten und Musikstile, über „musikalische Kostbarkeiten“ und hat nun wieder zum Ausgangspunkt seines beruflichen Lebens zurück gefunden. Damals schlug er den Weg zur Elektrotechnik und in der Folge dann zur Informatik ein; nun geht er ein Stück des anderen Weges - zur Musik und zu den Künsten.

Professor Dr. Peter Rechenberg war und ist für uns als Mensch, als Forscher und als akademischer Lehrer ein großes Vorbild, das unseren beruflichen Werdegang aber auch unser persönliches Handeln in vielerlei Hinsicht positiv beeinflusst hat. Dafür bedanken wir uns von ganzem Herzen.

Literatur

[Bjør78] Bjørner D. et al (eds,): The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science 61, Springer 1978

[Bla85] Blaschek G: Statische Programmanalyse. Elektronische Rechenanlagen 2/1985

[Bla87] Blaschek G., Pomberger, G: Moses – A Graphic Oriented Software Development Environment, Proceedings ACM Fifteenth Annual Computer Science Conference (1987): 58-66

[Bla94] Blaschek G.: Object-Oriented Programming with Prototypes. Springer-Verlag 1994

[Dob94] Dobler H.: Crosscompiler am Beispiel der Übersetzung von Fortran nach Ada. Dissertation. Technisch Naturwissenschaftliche Fakultät, Universität Linz. Universitätsverlag Trauner 1994
[DoPi90] Dobler H., Pirklbauer K.: Coco-2 - A New Compiler-Compiler. ACM SIGPLAN Notices 25 (1990)
[Hals77] Halstead M.A.: Elements of Software Science. Amsterdam: Elsevier 1977
[Knuth68] Knuth D.E.: Semantics of Context-free Languages. Mathematical Systems Theory 2(1968)2: 127-145
[McCa76] McCabe T.: A Complexity Measure. IEEE Transactions on Software Engineering (1976)2: 308-320
[Möss90] Mössenböck H.: A Generator for Production Quality Compilers. 3rd Intl. Workshop on Compiler Compilers (CC'90), Schwerin, Lecture Notes in Computer Science 477, Springer-Verlag 1990
[NaSh73] Nassi I., B. Shneiderman B.: Flowchart Techniques for Structured Programming, ACM SIGPLAN Notices, v. 8, n. 8, 12-26 (August 1973).
[Pom85] Pomberger G. (Hrsg.): Lilith und Modula-2 – Werkzeuge der Softwaretechnik. Hanser-Verlag 1985
[Rech64] Rechenberg P.: Grundzüge digitaler Rechenautomaten. Oldenbourg-Verlag 1964.
[Rech74] Rechenberg P.: Programmieren für Informatiker in PL/I. Band 1 und 2. Oldenbourg-Verlag 1974.
[Rech75] Rechenberg P.: Sackgassenfreie Syntaxanalyse. Elektronische Rechenanlagen 15(1975)3&4
[Rech76a] Rechenberg P.: Ein Kursus Übersetzerbau. In: Kerner, Melezinek: Lehrmethoden der Informatik. Verlag J. Heyn, Klagenfurt 1976.
[Rech76b] Rechenberg P.: Zur Didaktik der Assemblerprogrammierung. In: Kerner, Melezinek: Lehrmethoden der Informatik. Verlag J. Heyn, Klagenfurt 1976.
[Rech77] Rechenberg P.: MUMS – A Machine-Independent Programming Language Consisting of 360 Assembler Macro Calls. ACM Sigplan Notices, Sept. 1977
[Rech78] Rechenberg P.: Daten- und Programmkontrollstrukturen. In: Rechenberg, Schauer, Schoitsch, Schulz: Software-Entwurfsmethoden. Österreichische Computergesellschaft 1978
[Rech79a] Rechenberg P.: Prinzipien des Übersetzerbaus. In: Brockhaus, Pomberger, Rechenberg, Schauer: Prinzipien des Übersetzerbaus. Österreichische Computergesellschaft 1979
[Rech79b] Rechenberg P.: Übersetzung mit attributierten Grammatiken In: Brockhaus, Pomberger, Rechenberg, Schauer: Prinzipien des Übersetzerbaus. Österreichische Computergesellschaft 1979
[Rech83] Rechenberg P.: Konventionelle Programmstruktur und Datenkapselung: Eine Fallstudie. Angewandte Informatik 12, 1983
[Rech84a] Rechenberg P.: Attributierte Grammatiken als Methode der Softwaretechnik. Elektronische Rechenanlagen 26(1984)3: 111-119
[Rech84b] Rechenberg P.: Ada und Modula-2 - Programmiersprachen der achtziger und neunziger Jahre? Elektroniker (1984)3: 70-71
[Rech85] Rechenberg P.: Context Conditions for Modula-2. Technical Report. Institut für Informatik, Universität Linz, 1985
[Rech86] Rechenberg P.: Ein neues Maß für die softwaretechnische Komplexität von Programmen. Informatik Forschung und Entwicklung (1986)1: 26-37
[Rech89] Rechenberg P.: Programmierung. In Hütte: Das Grundwissen des Ingenieurs. 31. Auflage, Springer-Verlag 1989, S.99-133
[Rech90] Rechenberg P.: Programming Languages as Thought Models. Structured Programming 11 (1990) 3: 105-115
[Rech91a] Rechenberg P.: Was ist Informatik? Eine allgemeinverständliche Einführung. Hanser-Verlag 1991
[Rech91b] Rechenberg P.: Übersetzungen von Informatik-Literatur – bekümmert betrachtet. Informatik-Spektrum 14 (1991) 1: 28-33

[Rech93] Rechenberg P.: Compilers. In: Morris, Tamm (eds.): Concise Encyclopedia of Software Engineering. Pergamon Press 1993, pp. 59-64
[Rech97a] Rechenberg P.: Formale Sprachen und Automaten. In: Rechenberg, Pomberger (Hrsg.): Informatik-Handbuch, Hanser-Verlag 1997
[Rech97b] Rechenberg P.: Quo vadis, Informatik? LogIn 17 (1997) 1: 25-32
[Rech99] Rechenberg P.: Mythen und Fetische des Computerzeitalters. LogIn 19 (1999) 2: 28-33
[Rech02] Rechenberg P.: Technisches Schreiben (nicht nur) für Informatiker. Hanser-Verlag 2002
[Rei91] Reichenberger Ch.: Delta Storage for Arbitrary Non-Text Files. Proc. 3rd Intl Workshop on Software Configuration Management. Trondheim, June 1991, pp.144-152
[Rei93] Reichenberger Ch.: Konzepte und Verfahren für die Software-Versionsverwaltung. Dissertation, Technisch Naturwissenschaftliche Fakultät, Universität Linz. Universitätsverlag Trauner 1993
[ReMö85] Rechenberg P., Mössenböck H.: Ein Compiler-Generator für Mikrocomputer. Hanser-Verlag 1985
[RePo99] Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. Hanser-Verlag 1999
[RPR84] Rechenberg P., Pomberger G., Ritzinger F.: Compiler Writing Techniques and their Use in a Modula-2 Compiler. Tagungsband Implementierung von Programmiersprachen, 4. Fachgespräch, Zürich, März 1984
[Tenn76] Tennent R.D.: The denotational semantics of programming languages. CACM (1976) 437-453
[Terry97] Terry P.D.: Compilers and Compiler Generators. International Thomson Computer Press, 1997
[Wirth80] Wirth N.: Modula-2. Report 36, ETH Zürich, Institut für Informatik, März 1980.
[Wirth81] Wirth N.: Lilith: A Personal Computer for the Software Engineer. Proc. 5th Intern. Conf. on Software Engineering 1981, 2-15
[Wirth84] Wirth N.: Programming in Modula-2. Springer-Verlag 1982