

des Zweiten Weltkriegs nach außerhalb der Stadt verlegt. AT&T war ein kriegswichtiges Unternehmen und bot fachliche Beratung für ein breites Spektrum von militärischen Problemen – selbstverständlich für Kommunikationssysteme, aber auch für die Feuerleitreechner von Flugabwehrgeschützen, für Radar und Kryptografie. Ein Teil dieser Arbeit wurde in Vororten und ländlichen Gegenden von New Jersey erledigt. Der größte Standort befand sich in einem Gebiet namens Murray Hill, das zu den Kleinstädten New Providence und Berkeley Heights gehörte und 33 km westlich des alten Standorts lag.

Abbildung 1.1 zeigt die geografische Lage. 463 West Street befindet sich am Hudson River, auf der Karte ein wenig nördlich der Markierung für den Highway 9A. Die Bell Labs in Murray Hill liegen auf der Grenze zwischen New Providence und Berkeley Heights, etwas nördlich der Interstate 78. Beide Standorte sind mit roten Punkten markiert.

Immer mehr Tätigkeiten der Bell Labs wurden nach Murray Hill verlegt. Der Standort in der West Street wurde 1966 endgültig aufgegeben. In den 60er Jahren arbeiteten in Murray Hill mehr als 3.000 Personen, davon mindestens 1.000 mit einem Dokortitel in einer naturwissenschaftlich-technischen Disziplin wie Physik, Chemie, Mathematik oder verschiedenen Ingenieurwissenschaften.

Abbildung 1.2 zeigt eine Luftaufnahme der Einrichtungen in Murray Hill aus dem Jahr 1961. Aufgeteilt ist die Anlage in drei Hauptgebäude. Nr. 1 befindet sich in der Bildmitte und Nr. 2 oben links. Bei dem Viereck mit dem offenen Innenhof rechts handelt es sich um Nr. 3. Von dem einen Ende von Gebäude 1 bis zum anderen Ende von Gebäude 2 verlief damals ein durchgehender Korridor von 400 m Länge, der erst in den 70er Jahren durch zwei neu errichtete Gebäude unterbrochen wurde.

Ich habe mehr als 30 Jahre in Gebäude Nr. 2 gearbeitet, von meinem Praktikum im Jahr 1967 bis zu meiner Pensionierung 2000. Meine Büros befanden sich in den mit Punkten gekennzeichneten Seitenflügeln im vierten (obersten) Stock. Zur besseren Orientierung bei den folgenden Beschreibungen: Treppenhaus 9 befindet sich am hintersten Ende von Gebäude Nr. 2, Treppenhaus 8 einen Seitenflügel näher im Vordergrund. In den ersten Jahren war der Unix-Raum in der Dachkammer im fünften Stock zwischen diesen beiden Treppenhäusern untergebracht.

Abbildung 1.3 zeigt eine Google-Satellitenaufnahme der Bell Labs aus dem Jahr 2019. Die Gebäude Nr. 6 (unterhalb und links der Bildmitte, markiert) und Nr. 7 (oberhalb und rechts der Bildmitte) wurden Anfang der 70er hinzugefügt. Nach 1996 fungierte Gebäude Nr. 6 einige Jahre lang als Hauptsitz von Lucent Technologies. Es ist faszinierend zu sehen, wie viel Firmengeschichte in den von Google hinzugefügten Beschriftungen steckt: Bell Labs als Hauptmarkierung, Lucent Bell Labs auf der Ausfahrt, Alcatel-Lucent Bell Labs an der Einfahrt und Nokia Bell Labs am Anbau der Verwaltungspyramide in Gebäude Nr. 6.

Ich bin nicht wirklich qualifiziert, um eine ausführliche Geschichte der Labs zu schreiben, aber das haben zum Glück schon andere Autoren erledigt. Mir persönlich gefallen besonders das Buch *The Idea Factory* von Jon Gertner, das den Schwerpunkt auf die physikalische Forschung legt, sowie *The Information* von James Gleick, eine hervorragende Quelle für die Tätigkeiten auf dem Gebiet der Informationswissenschaft. Die umfangreiche offizielle Publikation der Bell Labs (fast 5.000 Seiten in sieben Bänden) unter dem Titel *A History of Science and Engineering in the Bell System* ist ausführlich, zuverlässig und für meinen Geschmack äußerst interessant.

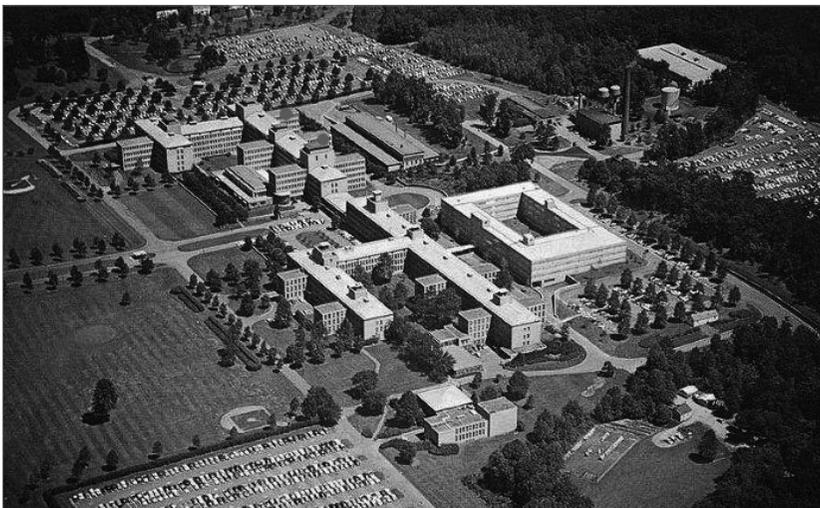


Abbildung 1.2
Die Bell Labs im Jahre 1961 (mit freundlicher Genehmigung der Bell Labs)

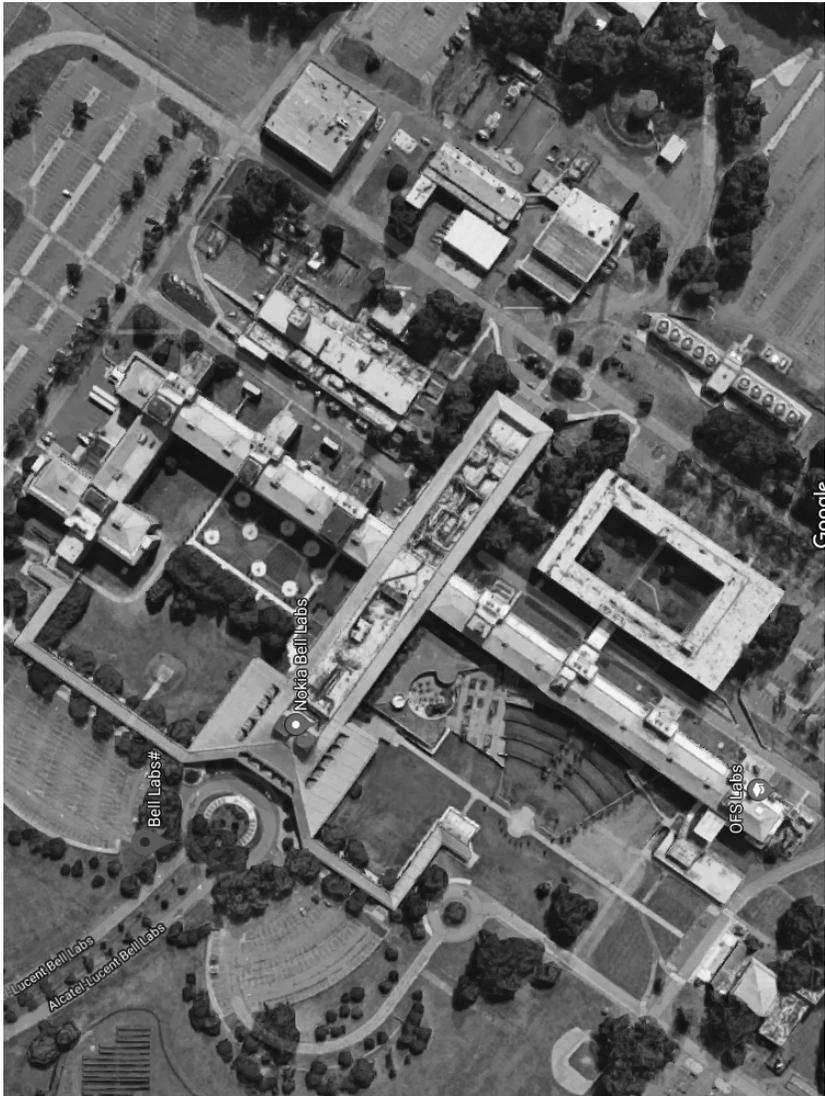


Abbildung 1.3
Die Bell Labs im Jahre 2019. Gebäude Nr. 6 liegt unten links.

Naturwissenschaften in den Bell Labs

In den ersten Jahren drehte sich die Forschung in den Bell Labs vorwiegend um Physik, Chemie, Materialwissenschaft und Kommunikationssysteme. Die Forscher genossen außerordentlich viel Freiheit, um ihren

eigenen Interessen nachzugehen, aber das Umfeld bot so viele relevante Probleme, dass es nicht weiter schwer war, sich auf Gebieten zu betätigen, die sowohl wissenschaftlich interessant waren als auch potenziellen Nutzen für Bell System und für die Welt im Allgemeinen versprachen.

Die Bell Labs steckten hinter einer großen Menge wissenschaftlicher und technischer Fortschritte von umwälzender Bedeutung. An erster Stelle steht dabei der Transistor, der 1947 von John Bardeen, Walter Brattain und William Shockley erfunden wurde, als sie daran arbeiteten, Verstärker für Ferntelefonleitungen zu verbessern. Der Transistor ging aus Grundlagenforschung über Halbleiter hervor, angeregt durch den Bedarf für Geräte, die haltbarer waren als Vakuumröhren und weniger Energie verbrauchten. Letztere bildeten in den 40er Jahren die einzige Möglichkeit, um Kommunikationsgeräte zu bauen – und nebenbei bemerkt auch die ersten Computer.

Die Erfindung des Transistors wurde mit dem Physik-Nobelpreis von 1956 gewürdigt – einem von neun Nobelpreisen für naturwissenschaftliche Arbeiten, die ganz oder teilweise in den Bell Labs erfolgten. Zu den weiteren bedeutenden Erfindungen gehören gegengekoppelte Verstärker, Solarzellen, Laser, Mobiltelefone, Kommunikationssatelliten und CCD-Sensoren (auf deren Grundlage Ihre Handykamera funktioniert).

Von den 60ern bis in die 80er arbeiteten etwa 3.000 Personen im Forschungsbereich der Bell Labs (vor allem in Murray Hill) und 15.000 bis 25.000 in Entwicklungsgruppen an verschiedenen Standorten, an denen Geräte und Systeme für die Bell Labs konstruiert wurden, oft auf der Grundlage von Ergebnissen des Forschungsbereichs. Das ist eine stattliche Anzahl. Wer hat sie alle bezahlt?

Da AT&T Telefondienste für fast die gesamten USA anbot, hatte das Unternehmen praktisch ein Monopol. Allerdings waren seine Möglichkeiten, diese Monopolstellung auszunutzen, beschränkt, da es von Behörden auf Bundes- und Staatenebene reguliert wurde. Die Preise, die AT&T für die verschiedenen Dienstleistungen verlangen durfte, unterlagen einer Kontrolle, und das Unternehmen durfte nur Geschäfte ausüben, die unmittelbar mit der Bereitstellung von Telefondiensten zusammenhingen.

Diese Regulierung funktionierte viele Jahre lang sehr gut. AT&T war verpflichtet, Dienstleistungen für alle anzubieten («universeller Dienst»),

auch für abgelegene, unrentable Orte. Zum Ausgleich erhielt das Unternehmen eine stabile und vorhersehbare Rendite.

Im Rahmen dieser Vereinbarungen zweigte AT&T einen kleinen Teil seiner Einkünfte an die Bell Labs ausdrücklich für den Zweck ab, die Kommunikationsdienstleistungen zu verbessern. De facto wurden die Bell Labs also durch eine bescheidene Gebühr bezahlt, die bei jedem Telefonat im Land anfiel. Nach einer Abhandlung von A. Michael Noll gab AT&T etwa 2,8% der Einnahmen für Forschung und Entwicklung aus, wobei etwa 0,3% auf Grundlagenforschung entfielen. Ich bin mir nicht sicher, ob so etwas auch heute noch funktionieren würde, aber jahrzehntelang sorgte diese Vorgehensweise für kontinuierliche Verbesserungen des Telefonsystems und eine erhebliche Anzahl grundlegender wissenschaftlicher Entdeckungen.

Die sichere Finanzierung war von großer Bedeutung für die Forschung. Dadurch gewann AT&T eine sehr langfristige Sichtweise. Die Forscher bei den Bell Labs hatten die Freiheit, sich auch mit Themen zu beschäftigen, die keinen kurzfristigen Ertrag versprachen und vielleicht sogar niemals gewinnbringend sein mochten. Das unterscheidet sich sehr stark von der heutigen Situation, in der oft nur wenige Monate vorausgeplant und viel Aufwand betrieben wird, um über die finanziellen Ergebnisse im nächsten Quartal zu spekulieren.

Kommunikationstechnik und Informatik

Die Bell Labs leisteten naturgemäß Pionierarbeit bei der Konstruktion, Erstellung und Verbesserung von Kommunikationssystemen. Dieser Sammelbegriff umfasste alles – von der Endbenutzerhardware wie den Telefonen über Schaltsysteme und Mikrowellen-Sendeanlagen bis zu Glasfaserkabeln.

Manchmal führte diese breite Palette von praktischen Anwendungen sogar zu Fortschritten in der Grundlagenforschung. Beispielsweise versuchten Arno Penzias und Robert Wilson 1964 herauszufinden, was das störende Rauschen verursachte, das bei den von Echo-Ballonatelliten zurückgeworfenen Funksignalen auftrat. Dabei erkannten sie schließlich, dass es sich um eine Hintergrundstrahlung handelte, die noch vom Urknall herrührte. Diese Entdeckung brachte ihnen 1978 den Nobelpreis ein. Penzias kommentierte dies mit den Worten: »Die meisten Leute

bekommen den Nobelpreis für Dinge, nach denen sie gesucht haben. Wir haben ihn für etwas bekommen, das wir loswerden wollten.«

Eine weitere Aufgabe der Bell Labs bestand darin, ein genaues mathematisches Verständnis der Funktionsweise von Kommunikationssystemen zu entwickeln. Das wichtigste Ergebnis dabei war die von Claude Shannon begründete Informationstheorie, die teilweise auf seinen Kryptografiestudien während des Zweiten Weltkriegs fußte. In seiner 1948 im *Bell System Technical Journal* veröffentlichten Abhandlung »A Mathematical Theory of Communication« beschrieb er die grundlegenden Eigenschaften von Information und fand heraus, wie man die Informationsmenge bestimmt, die prinzipiell über ein Kommunikationssystem übertragen werden kann. Shannon arbeitete von Anfang der 40er Jahre bis 1956 in Murray Hill. Danach kehrte er ans MIT zurück, wo er studiert hatte. 2001 starb er im Alter von 84 Jahren.

Als Computer immer leistungsfähiger und billiger wurden, erweiterten die Bell Labs ihren Einsatz nicht nur im Bereich der Datenanalyse, sondern dehnten ihn auch auf die umfassende Modellierung und Simulation von physikalischen Systemen und Prozessen aus. Die Bell Labs hatten sich schon seit den 30er Jahren mit Computern und EDV befasst und unterhielten seit Ende der 50er Jahre Rechenzentren mit zentralen Großrechnern.

Anfang der 60er wurden Mitarbeiter des Forschungszweiges Mathematik sowie Mitglieder des Bedienungspersonals für den großen Zentralcomputer in Murray Hill einer neu gegründeten Informatik-Forschungsgruppe zugeteilt, die die Bezeichnung Computing Science Research Center erhielt (nach der Nummer in der Organisationshierarchie oft auch »Center 1127« genannt). Auch wenn sie für kurze Zeit noch Computerdienste für alle Mitarbeiter in Murray Hill leistete, war sie keine Serviceeinrichtung, sondern gehörte zum Unternehmensbereich Forschung. 1970 wurde die Gruppe, die sich um die IT-Dienstleistungen kümmerte, in eine eigenständige Organisation ausgelagert.

Meine Zeit in den Bell Labs

Ein großer Teil dieses Abschnitts erzählt von meiner persönlichen Geschichte. Ich möchte Ihnen damit zeigen, welchen glücklichen Zufällen ich meine Karriere in der Informatik zu verdanken habe und dass die Bell Labs unschlagbar sind, wenn es darum geht, eine solche Karriere zu verfolgen.

Ich wurde in Toronto geboren und ging auch dort zur Universität. Mein Fachgebiet war Technische Physik, praktisch ein Sammelbecken für alle, die nicht genau wussten, auf welchen Schwerpunkt sie sich konzentrieren sollten. Meinen Abschluss machte ich 1964, in der Frühzeit der EDV. Ich war in meinem dritten Studienjahr, als ich zum ersten Mal einen Computer sah, nämlich den Großrechner, der für die gesamte Uni da war. Die IBM 7094 zählte damals zu den Spitzenmodellen und verfügte über einen Magnetkernspeicher für 32K (also 32.768) 36-Bit-Wörter (heute würden wir sagen 128 KByte) sowie Sekundärspeicher in Form großer, mechanischer Plattenlaufwerke. Sie kostete drei Millionen Dollar und war in einem großen Raum mit Klimaanlage untergebracht, behütet von professionellem Bedienpersonal. Gewöhnliche Sterbliche (insbesondere Studenten) wurden nicht einmal in die Nähe gelassen.

Daher habe ich mich während meines Diplomstudiums nur wenig mit Informatik beschäftigt. Allerdings lernte ich die Programmiersprache Fortran. Deshalb kann ich mich nur zu gut in Menschen hineinversetzen, die Schwierigkeiten haben, ihr erstes Programm zu schreiben. Ich hatte zwar Daniel McCrackens hervorragendes Buch über Fortran II gelesen und alle Regeln parat, aber mir war nicht klar, wie ich überhaupt beginnen sollte, um ein Programm zu schreiben. Dieser erste Schritt scheint vielen Einsteigern Schwierigkeiten zu bereiten.

Im Sommer vor meinem letzten Collegejahr hatte ich einen Job bei Imperial Oil in Toronto als Mitglied einer Gruppe, die Optimierungssoftware für Raffinerien entwickelte. (Imperial Oil gehörte zum Teil der Firma Standard Oil of New Jersey, aus der 1972 Exxon wurde.)

Im Nachhinein muss ich zugeben, dass ich wohl ein ziemlich unterdurchschnittlicher Praktikant war. Ich verwendete den ganzen Sommer darauf, ein riesiges Cobol-Programm zur Analyse von Raffineriedaten zu schreiben. An den genauen Zweck kann ich mich nicht mehr erinnern, aber ich weiß noch genau, dass es niemals funktionierte. Mir war nicht klar, wie

man überhaupt programmiert, Cobol bot wenig Unterstützung für eine gute Programmgliederung, und strukturierte Programmierung war noch nicht erfunden. Mein Code bestand aus einer endlosen Folge von IF-Anweisungen mit Verzweigungen, bei denen der Code irgendetwas tun sollte, sobald ich herausgefunden hatte, was dieses Etwas war.

Ich versuchte auch, einige Fortran-Programme auf der IBM 7010 von Imperial ans Laufen zu bekommen. Schließlich hatte ich gewisse Kenntnisse in Fortran – auf jeden Fall bessere als in Cobol –, und Fortran hätte sich auch besser für die Datenanalyse geeignet. Erst nachdem ich mich wochenlang mit JCL herumgeschlagen hatte, der »Job Control Language« von IBM, stellte ich fest, dass der 7010 keinen Fortran-Compiler hatte. Die Fehlermeldungen von JCL waren so undurchschaubar, dass es bis dahin auch niemand sonst herausgefunden hatte.

Als ich nach diesem etwas frustrierenden Sommer für mein Abschlussjahr an die Universität zurückkehrte, war ich immer noch sehr stark an Computern interessiert. Es gab keine formalen Informatikkurse, aber dennoch schrieb ich meine Diplomarbeit über künstliche Intelligenz, was damals ein gefragtes Thema war. Sogenannte »Theorembeweiser« oder Programme, die Schach und Dame spielen können, sowie die maschinelle Übersetzung menschlicher Sprachen schienen damals alle in Reichweite zu sein und nur noch ein bisschen Programmierung zu erfordern.

Nachdem ich 1964 meinen Abschluss gemacht hatte, wusste ich nicht recht, was ich als Nächstes tun sollte. Wie viele andere Studenten verschob ich die Entscheidung, indem ich mich zu einem Aufbaustudium entschloss. Ich bewarb mich bei einem Dutzend Universitäten in den USA (was damals unter Kanadiern nicht üblich war – und ich bin aus Kanada) und erhielt von mehreren eine Zusage, darunter vom MIT und von Princeton. Princeton teilte mir mit, dass normalerweise drei Jahre bis zur Promotion vergingen, während es laut Aussage des MIT wahrscheinlich sieben Jahre sein würden. Außerdem bot mir Princeton ein komplettes Stipendium an, während ich beim MIT 30 Stunden die Woche als Forschungsassistent zu arbeiten hätte. Das machte die Entscheidung ziemlich einfach. Außerdem studierte Al Aho, ein guter Freund, der in Toronto ein Jahr über mir gewesen war, bereits in Princeton. Also machte ich mich auf den Weg, was sich später als eine äußerst vorteilhafte Entscheidung herausstellte.

1966 hatte ich erneut Glück, da ich ein Sommerpraktikum beim MIT ergattern konnte. Teilweise lag das daran, dass Lee Varian, ebenfalls Princeton-Student, dort im Vorjahr hervorragende Arbeit geleistet hatte. Den Sommer über nutzte ich das CTSS (Compatible Time-Sharing System) und schrieb Programme in MAD (Michigan Algorithm Decoder, ein Dialekt von Algol 58), um Tools für das neue Betriebssystem Multics zu schreiben, auf das ich in Kapitel 2 noch zurückkommen werde. (Multics wurde ursprünglich MULTICS geschrieben, aber die Variante mit Kleinbuchstaben wirkt optisch angenehmer. Ich verwende hier die besser lesbare Version, auch wenn sie historisch nicht korrekt ist. Das gilt auch für andere komplett in Großbuchstaben geschriebene Wörter, also z.B. UNIX/Unix.)

Mein offizieller Vorgesetzter beim MIT war Prof. Fernando Corbató, von allen nur »Corby« genannt, ein wunderbarer Gentleman, der nicht nur CTSS entworfen hatte, sondern auch für Multics verantwortlich war. Für seine grundlegenden Arbeiten an Timesharing-Systemen erhielt er den Turing Award 1990. Corby starb im Juli 2019 im Alter von 93 Jahren.

Corby leitete nicht nur das Design und die Implementierung von CTSS und Multics, sondern war auch der Erfinder von Passwörtern für den Computerzugriff. Im Stapelverarbeitungsbetrieb war so etwas nicht nötig gewesen, aber bei einem Mehrbenutzercomputer mit einem gemeinsamen Dateisystem war ein Sicherheitsmechanismus zum Schutz der privaten Dateien erforderlich.

Ich kann mich noch an die beiden kurzen, aussprechbaren Passwörter erinnern, die für mich im Sommer 1966 am MIT generiert wurden. Es waren zwei, nicht nur eines, da eines Tages jemand die Datei mit den Passwörtern und die Datei mit der Login-Meldung des Tages verwechselte. Da die Passwörter unverschlüsselt gespeichert waren, konnte dadurch jeder, der sich anmeldete, sämtliche Passwörter einsehen.

Mein Büro gehörte eigentlich Prof. Joseph Weizenbaum, der den Sommer über aber nicht da war, sodass ich es nicht mit ihm teilen musste. Anfang 1966 hatte Weizenbaum eine Beschreibung von Eliza veröffentlicht, einem Programm, das ein Gespräch mit einem Psychologen der Rogers-Schule simulierte. Eliza stellte einen der ersten Versuche dar, zur Interaktion mit einem Computer auf natürliche Sprache zurückzugreifen. Die Simulation wirkte erstaunlich überzeugend, zumindest wenn man

ihr zum ersten Mal ausgesetzt wurde, und ich hatte großen Spaß dabei, sie Besuchern stolz vorzuführen. Heute sind im Web viele Eliza-Simulatoren zu finden. Es ist immer noch recht vergnüglich, ein paar Minuten damit herumzuspielen, allerdings werden ihre Schwächen sehr bald deutlich.

ELIZA: Please tell me what's been bothering you.

SIE: I'm trying to write a book about Unix.

ELIZA: Is it because you are trying to write a book about unix that you came to me?

SIE: Yes.

ELIZA: You are sure.

SIE: Yes.

ELIZA: You seem to be quite positive.

SIE: Yes.

ELIZA: I see.

Im Sommer 1967 zog ich dann das große Los: Ich bekam eine Praktikumsstelle bei den Bell Labs in Murray Hill, wo ich im Computing Science Research Center für Doug McIlroy (Abbildung 1.4) arbeitete. Doug schlug mir vor, ein Problem bei der Beurteilung von Speicherzuweisungsverfahren zu untersuchen, was eines seiner langfristigen Forschungsgebiete war. In typischer Praktikantenmanier wurstelte ich also so herum und machte letzten Endes etwas völlig anderes – nämlich eine Bibliothek von Funktionen zur Vereinfachung der Listenverarbeitung in Fortran-Programmen aufzustellen. Den Sommer über schrieb ich Programme in einer strengen Assemblersprache für den damals aktuellen Großrechner in Murray Hill, einen GE 635. Dabei handelte es sich im Grunde genommen um eine verbesserte IBM 7094, aber auch um eine vereinfachte Version der GE 645, die eigens für Multics entworfen worden war. Das war so ziemlich die letzte Gelegenheit, bei der ich in Assemblersprache schrieb. Die ganze Sache war zwar völlig verfehlt, machte aber einen Riesenspaß, und damit war ich der Programmierung ein für alle Mal verfallen.

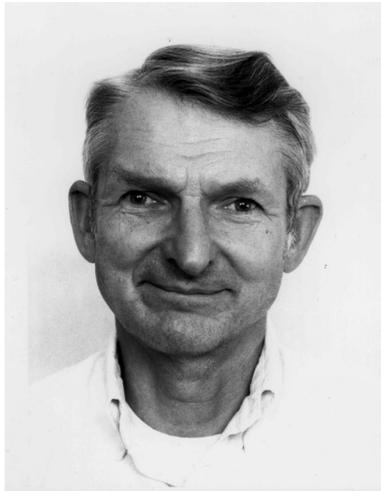


Abbildung 1.4
Doug McIlroy, ca. 1984 (freundlicherweise zur Verfügung gestellt von Gerard Holzmann)

Die Büros

Die räumliche Anordnung kann manchmal schicksalhaft sein.

Das Büro, das ich 1967 als Praktikant bezog, lag im vierten Stock von Gebäude Nr. 2 an einem Flur, der von Treppenhaus 8 abzweigte. An meinem ersten Tag saß ich in diesem Büro (das waren noch die guten alten Zeiten, in denen man selbst als Praktikant mit Glück ein Büro für sich allein ergattern konnte) und fragte mich, was ich hier eigentlich tun sollte, als ein älterer Herr um 11 Uhr seinen Kopf zur Tür hereinstreckte und sagte: »Hallo, ich bin Dick [unverständlich]. Sollen wir Mittagessen gehen?«

Also gut, dachte ich, warum nicht? Von der Mittagspause selbst weiß ich nichts mehr, aber ich kann mich noch gut daran erinnern, dass Dick [unverständlich] nachher noch anderswo hinging und ich den Flur entlangschlich, um das Namensschild an seiner Bürotür zu lesen. Richard Hamming! Mein freundlicher Zimmernachbar war der berühmte Erfinder von Fehlerkorrekturcodes und der Autor des Lehrbuchs für einen Kurs in numerischer Analyse, den ich gerade erst belegt hatte.

Dick (Abbildung 1.5) und ich wurden gute Freunde. Er hatte feste Überzeugungen und scheute sich auch nicht, sie auszudrücken, auch wenn er

damit einige Leute vor den Kopf stieß. Ich dagegen genoss seine Gesellschaft, und sein Rat war für mich äußerst nutzbringend.

Zwar war er Abteilungsleiter, allerdings gab es in seiner Abteilung keinen einzigen Mitarbeiter, was ein bisschen merkwürdig erschien. Er erzählte mir, dass er hart gearbeitet hatte, um diese Kombination aus angemessenem Titel und fehlender Verantwortung zu erreichen. Nachvollziehen konnte ich das erst, als ich selbst Leiter einer Abteilung mit einem Dutzend Mitarbeitern wurde.



Abbildung 1.5
Dick Hamming, ca. 1975, mit seinem typischen Plaidjacket (Wikipedia)

Ich war dabei, als er im Sommer 1968 erfuhr, dass er den ACM Turing Award gewonnen hatte, der heute als das Gegenstück zum Nobelpreis in der Informatik angesehen wird. Damals war der Nobelpreis mit 100.000 Dollar dotiert, der Turing Award dagegen mit nur 2.000 Dollar, was Dick zu der sarkastischen Bemerkung veranlasste, er hätte 2% des Nobelpreises gewonnen. (Dies war übrigens erst die dritte Verleihung des Turing Award. Die beiden vorhergehenden Preise gingen an Alan Perlis und Maurice Wilkes, ebenfalls Pioniere der Informatik.) Dick wurde für seine Arbeiten über numerische Methoden, automatische Codiersysteme sowie Codes zur Fehlererkennung und -behebung ausgezeichnet.

Es war auch Dick, der mich dazu anregte, Bücher zu schreiben, was sich als eine gute Sache herausstellte. Er hielt nicht viel von den meisten Programmierern, da sie seiner Meinung nach wenn überhaupt, dann nur

eine schlechte Ausbildung erhalten hatten. Ich kann ihn immer noch sagen hören:

»Wir geben ihnen ein Wörterbuch und die Grammatikregeln an die Hand und sagen ihnen: ›Junge, jetzt bist du ein großartiger Programmier.««

Seiner Ansicht nach sollte Programmieren ebenso gelehrt werden wie Schreiben. Was guter und was schlechter Code war, sollte auch anhand des Stils unterschieden werden. Programmierer sollten lernen, gut zu schreiben und guten Stil zu würdigen.

Unsere Meinungen gingen zwar darüber auseinander, wie das zu erreichen war, aber sein Grundgedanke war sinnvoll und führte unmittelbar zu meinem ersten Buch, *The Elements of Programming Style*, das ich 1974 zusammen mit P. J. »Bill« Plauger veröffentlichte, der zu diesem Zeitpunkt im Büro nebenan saß. Dabei folgten Bill und ich dem Vorbild des englischen Stilhandbuchs *The Elements of Style* von Strunk und White, indem wir ebenfalls Beispiele von schlecht geschriebenem Code vorstellten und zeigten, wie sie sich jeweils verbessern ließen.

Unser erstes Beispiel stammte aus einem Buch über numerische Analyse, das Dick mir gezeigt hatte. Eines Tages kam er damit in mein Büro und ließ sich völlig aufgebracht darüber aus, wie schlecht die numerischen Programmteile waren. Mir sprang gleich ein Abschnitt mit furchtbarem Fortran-Code ins Auge:

```
DO 14 I=1, N
DO 14 J=1, N
14 V(I, J) = (I/J) * (J/I)
```

Lassen Sie mich das für Nicht-Fortran-Programmierer erklären. Dieser Code besteht aus zwei verschachtelten DO-Schleifen, die beide in der Zeile mit der Nummer 14 enden. Jede der Schleifen durchläuft ihre Indexvariable von der unteren zur oberen Schranke. In der äußeren Schleife durchläuft also I die Werte von 1 bis N , in der inneren Schleife J . Die Variable V ist ein Array aus N Zeilen und N Spalten. I durchläuft alle Zeilen und J jeweils alle Spalten einer Zeile.

Dieses Schleifenpaar erstellt also eine $N \times N$ -Matrix mit dem Wert 1 in der Diagonalen und 0 an allen anderen Positionen. Für $N = 5$ ergibt sich beispielsweise:

[2]

Das Ur-Unix (1969)

»Irgendwann bemerkte ich, dass ich drei Wochen von einem Betriebssystem entfernt war.«

– Ken Thompson, Vintage Computer Festival East, 4. Mai 2019

Das Betriebssystem Unix erblickte 1969 das Licht der Welt, aber es entstand nicht einfach aus dem Nichts heraus, sondern fußte auf den Erfahrungen verschiedener Personen bei den Bell Labs, die an anderen Betriebssystemen und Sprachen gearbeitet hatten.

Technischer Hintergrund

Dieser Abschnitt gibt eine kurze Einführung in die technischen Aspekte, um die es in diesem Buch geht, also Computer, Hardware, Software, Betriebssysteme, Programmierung und Programmiersprachen. Wenn Sie mit diesen Begriffen bereits vertraut sind, können Sie ruhig vorblättern, und wenn nicht, hoffe ich, dass Ihnen die folgenden Beschreibungen helfen, den Rest besser zu verstehen. Als ausführlichere Erklärung für technisch nicht versierte Leser empfehle ich mein Buch *Understanding the Digital World*, wobei ich natürlich nicht ganz unvoreingenommen bin.

Ein Computer ist im Grunde genommen nicht viel mehr als einer der Taschenrechner, die es früher als eigenständige Geräte gab und die jetzt als App auf Smartphones zu finden sind. Sie können arithmetische Berechnungen ungeheuer schnell ausführen. Heutzutage erledigen sie Milliarden Rechenvorgänge pro Sekunde, während es in den 70er Jahren noch deutlich weniger als Millionen pro Sekunde waren.

Ein typischer Computer der 60er und 70er verfügte über einen Vorrat von wenigen Dutzend Instruktionen, die er ausführen konnte, etwa um arithmetische Operationen vorzunehmen (Addition, Subtraktion, Multiplikation und Division), um Informationen aus dem primären Speicher zu lesen und darin zu speichern und um mit Festplatten oder anderen angeschlossenen Geräten zu kommunizieren. Von entscheidender Bedeutung aber sind Anweisungen, die aufgrund der Ergebnisse vorheriger Berechnungen bestimmen, welche Instruktionen als Nächstes ausgeführt werden sollen. Der Computer entscheidet also aufgrund dessen, was er getan hatte, was er als Nächstes tut. In diesem Sinne lenkt er sein eigenes Schicksal.

Instruktionen und Daten werden im selben primären Speicher untergebracht, der gewöhnlich als Arbeitsspeicher oder RAM (Random Access Memory, also »Speicher mit wahlfreiem Zugriff«) bezeichnet wird. Je nachdem, was für eine Folge von Instruktionen Sie in das RAM laden, erledigt der Computer bei der Ausführung eine andere Aufgabe. Das geschieht etwa, wenn Sie auf das Symbol für ein Programm wie Word oder Chrome klicken: Dem Betriebssystem wird dadurch befohlen, die Instruktionen für das betreffende Programm in den Arbeitsspeicher zu laden und sie auszuführen.

Unter Programmierung versteht man die Tätigkeit, Folgen von Operationen zusammenzustellen, um eine gegebene Aufgabe zu erledigen. Dazu wird eine Programmiersprache verwendet. Es ist zwar auch möglich, die erforderlichen Instruktionen direkt zu erstellen, allerdings ist das selbst für kleine Programme eine mühselige Aufgabe, bei der viele Einzelheiten zu beachten sind. Bei den meisten Fortschritten, die auf dem Gebiet der Programmierung erzielt wurden, ging es daher darum, Programmiersprachen zu erschaffen, die stärker daran angelehnt sind, wie Menschen eine Berechnung ausdrücken würden. Als *Compiler* bezeichnete Programme (die natürlich ebenfalls geschrieben werden müssen) übersetzen von Sprachen höherer Ordnung (die sich stärker an menschlichen Sprachen orientieren) in die einzelnen Instruktionen für die jeweilige Art von Computer.

Ein Betriebssystem schließlich ist ein umfangreiches und kompliziertes Programm, das aus den gleichen Instruktionen aufgebaut ist wie gewöhnliche Programme, etwa Word oder ein Browser. Seine Aufgabe be-

steht darin, alle anderen Programme zu steuern, die ausgeführt werden wollen, und sich um die Wechselwirkung mit dem Rest des Computers zu kümmern.

Das klingt alles ziemlich abstrakt. Das folgende kleine Beispiel soll Ihnen daher konkret zeigen, worum es bei der Programmierung geht. Nehmen wir an, Sie möchten die Fläche eines Rechtecks aus seiner Länge und Breite berechnen. Auf Deutsch können wir sagen: »Die Fläche ist das Produkt aus Länge und Breite.« Als Formel ausgedrückt, ergibt sich Folgendes:

$$\text{Fläche} = \text{Länge} \times \text{Breite}$$

In einer höheren Programmiersprache (deren Elemente meistens aus dem Englischen entlehnt sind) können wir das wie folgt schreiben:

```
area = length * width
```

In den meisten Programmiersprachen von heute würde diese Anweisung tatsächlich genau so aussehen. Ein Compiler übersetzt dies in eine immer noch verständliche, rechner-spezifische Folge von Maschineninstruktionen für einen Computer. Für einen einfachen hypothetischen Computer kann das Ergebnis wie folgt aussehen:

```
load      length
multiply  width
store     area
```

Ein als *Assembler* bezeichnetes Programm überführt diese Folge von mehr oder weniger verständlichen Instruktionen schließlich in eine Folge von Maschineninstruktionen, die in den Arbeitsspeicher des Computers geladen werden. Bei der Ausführung dieser Instruktionen wird die Fläche aus der gegebenen Länge und Breite berechnet. Diese vereinfachte Darstellung schweigt sich zwar über viele Einzelheiten aus – etwa wie wir den Kompilier- und Ladevorgang festlegen, wie die Werte für die Länge und die Breite in den Computer gelangen, wie die Fläche ausgegeben wird usw. –, vermittelt aber die Grundzüge des Vorgangs.

Als konkretes, funktionierendes Beispiel finden Sie hier ein vollständiges Programm in der Sprache C, das eine Länge und eine Breite liest und die daraus berechnete Fläche ausgibt:

```
void main() {
    float length, width, area;
    scanf("%f %f", &length, &width);
    area = length * width;
    printf("area = %f\n", area);
}
```

Dieses Programm kann auf jedem beliebigen Computer kompiliert und ausgeführt werden.

Moderne Betriebssysteme wie Windows und macOS oder auf Smartphones auch Android und iOS sind zumindest dem Namen nach allgemein bekannt.

Ein Betriebssystem ist ein Programm, das den Computer steuert. Es verteilt die Ressourcen auf die laufenden Programme, verwaltet den Hauptspeicher und weist ihn den Programmen nach deren Bedarf zu. Auf einem Desktop- oder Laptopcomputer sorgt das Betriebssystem dafür, dass Sie einen Browser, eine Textverarbeitung, ein Musikwiedergabeprogramm und vielleicht auch unser kleines Programm zur Flächenberechnung ausführen können, und zwar alle zugleich, wobei es seine Aufmerksamkeit nach Bedarf einmal dem einen und einmal dem anderen zuwendet.

Es steuert auch die Anzeige, um die einzelnen Programme auf dem Bildschirm darzustellen, wenn es gefordert wird, und verwaltet Speichergeräte wie die Festplatte, sodass beispielsweise ein Word-Dokument beim Speichern so abgelegt wird, dass Sie es später wieder aufrufen und weiterbearbeiten können.

Das Betriebssystem koordiniert auch die Kommunikation mit Netzwerken wie dem Internet, sodass Sie Ihren Browser zur Recherche, zur Kommunikation mit Freunden, zum Einkaufen und zum Veröffentlichen von Katzenvideos nutzen können, und all das zur gleichen Zeit.

Auch wenn es für Nichtprogrammierer nicht so offensichtlich ist, dient das Betriebssystem auch dazu, die einzelnen Programme und sich selbst vor fehlerhaften oder schädlichen Programmen und Benutzern zu schützen.

Betriebssysteme auf Telefonen funktionieren ähnlich. Hinter den Kulissen geschieht sehr viel, um die Kommunikation über ein Mobilfunknetz oder ein WLAN aufrechtzuerhalten. Apps entsprechen dabei Program-

men wie etwa Word, auch wenn es Unterschiede in den Einzelheiten gibt. Sie sind auch in den gleichen Programmiersprachen geschrieben.

Heutzutage handelt es sich bei Betriebssystemen um große und komplizierte Programme. In den 60er Jahren war das Leben einfacher, aber nach den Maßstäben jener Zeit waren die damaligen Betriebssysteme ebenfalls groß und kompliziert. Gewöhnlich stellte ein Computerhersteller wie IBM oder DEC (Digital Equipment Corporation) ein oder mehrere Betriebssysteme für seine verschiedenen Hardwareprodukte bereit. Es gab keine Gemeinsamkeiten zwischen Hardware von verschiedenen Herstellern und manchmal nicht einmal zwischen den Geräten ein und desselben Herstellers. Daher wiesen auch die verschiedenen Betriebssysteme keinerlei Gemeinsamkeiten auf.

Noch komplizierter wurde die Sache dadurch, dass die Betriebssysteme in Assemblersprache geschrieben waren, also einer Form von Maschineninstruktionen, die zwar für Menschen lesbar war, aber stark in die Einzelheiten ging und auf den Befehlsumfang einer bestimmten Hardware zugeschnitten war. Jede Art von Computer brachte ihre eigene Assemblersprache mit. Bei den Betriebssystemen handelte es sich daher um umfangreiche und komplizierte Assemblerprogramme, die in der spezifischen Sprache für die jeweilige Hardware geschrieben waren.

Dieser Mangel an Gemeinsamkeiten und die Verwendung inkompatibler maschinennaher Sprachen behinderten erheblich den Fortschritt, da sie es erforderlich machten, jedes Programm in mehreren Versionen zu schreiben. Um ein Programm, das für ein bestimmtes Betriebssystem erstellt worden war, auf ein anderes Betriebssystem oder eine andere Architektur zu übertragen, musste es komplett neu geschrieben werden. Wie Sie noch sehen werden, wurde es mit Unix möglich, das gleiche Betriebssystem auf allen Arten von Hardware auszuführen. Als Unix schließlich nicht mehr in einer Assembler-, sondern einer Hochsprache geschrieben war, ließ es sich mit relativ wenig Aufwand von einem Computer auf einen anderen übertragen.

CTSS und Multics

Das innovativste Betriebssystem seiner Zeit war das 1964 beim MIT entwickelte CTSS (Compatible Time-Sharing System). Die meisten Betriebssysteme waren für das da, was man damals unter »Stapelverarbeitung« verstand: Programmierer stanzen ihre Programme in Lochkarten (das war vor langer Zeit!), übergaben diese dann einem Operator und warteten darauf, dass ihnen die Ergebnisse ausgehändigt wurden, was Stunden und manchmal sogar Tage dauern konnte.

Lochkarten bestanden aus hochwertigem, steifem Karton und konnten bis zu 80 Zeichen speichern. Gewöhnlich enthielt jede Karte eine Programmzeile. Unser sechszeiliges C-Programm hätte also sechs Karten beansprucht. Bei einem Programmwechsel mussten die Karten ausgetauscht werden. Abbildung 2.1 zeigt eine Standardkarte mit 80 Spalten.

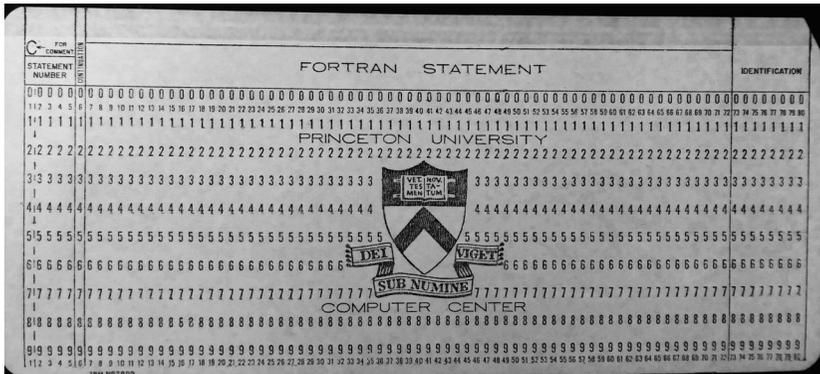


Abbildung 2.1
Lochkarte, 18,7 cm × 8,3 cm

Dagegen verwendeten CTSS-Programme schreibmaschinenähnliche Geräte (»Terminals« wie der Teletype 33 in Abbildung 3.1 im nächsten Kapitel), die direkt oder über Telefonleitungen an einen Großrechner angeschlossen waren – eine IBM 7094 mit doppelt so viel Speicher wie den üblichen 32K Wörtern. Das Betriebssystem verteilte seine Aufmerksamkeit auf die angemeldeten Benutzer, wobei es schnell von einem aktiven Benutzer zum anderen umschaltete und so die Illusion hervorrief, dass jeder den kompletten Computer zu seiner Verfügung hatte. Dieses Timesharing-System (»Zeitscheibensystem«) war meiner persönlichen Erfahrung nach unendlich viel angenehmer und produktiver als die Sta-

pelverarbeitung. Meistens hatte man das Gefühl, als ob es gar keine anderen Benutzer gäbe.

CTSS erwies sich als eine so produktive Programmierumgebung, dass sich die Forscher am MIT entschlossen, eine noch bessere Version zu erstellen, die als »Informationsversorger« Computerdienste für umfangreiche und weit verteilte Benutzergruppen bereitstellen sollte. 1965 begannen sie, ein System namens Multics zu konstruieren, was für »Multiplexed Information and Computing Service« stand.

Die Entwicklung von Multics erforderte einen erheblichen Aufwand, da sie sowohl anspruchsvolle neue Software als auch neue Hardware mit mehr Möglichkeiten erforderte, als sie die IBM 7094 bot. Daher nahm das MIT zur Unterstützung noch zwei weitere Organisationen ins Boot. Das Unternehmen General Electric, das damals Computer baute, sollte einen neuen Rechner mit Hardwaremerkmalen konstruieren, die Timesharing-Systeme und mehrere Benutzer besser unterstützten. Die Bell Labs, die schon seit den frühen 50er Jahren viel Erfahrung durch die Entwicklung ihrer eigenen Systeme gewonnen hatten, sollten am Betriebssystem mitarbeiten.

Die Aufgabenstellung des Multics-Projekts an sich stellte bereits eine Herausforderung dar, und schon bald traten Probleme auf. Rückblickend betrachtet lag das wohl zum Teil am *Zweitsystemeffekt*: Wenn man ein erfolgreiches System hat (wie CTSS), ist die Versuchung groß, ein neues System erstellen zu wollen, das alle restlichen Probleme des ursprünglichen beseitigt und zusätzlich die Wunschmerkmale aller Beteiligten erfüllt. Das resultiert oft in einem zu komplizierten System, da versucht wird, zu vieles auf einmal zu erreichen. Das war auch bei Multics der Fall. Es wurde mehrfach als »überkonstruiert« beschrieben, und Sam Morgan nannte es den »Versuch, auf zu viele Bäume auf einmal zu klettern«. Außerdem muss man nicht einmal Organisationswesen studiert haben, um zu ahnen, dass bei einem Projekt, an dem eine Universität und zwei Unternehmen an drei verschiedenen Standorten beteiligt sind, Schwierigkeiten auftreten werden.

Von 1966 bis 1969 arbeiteten gut ein halbes Dutzend Forscher der Bell Labs an Multics, darunter Doug McIlroy, Dennis Ritchie, Ken Thompson und Peter Neuman, der Vic Vyssotskys Posten übernommen hatte, nachdem Vic zu einem anderen Standort der Bell Labs umgezogen war.

Doug war intensiv mit PL/I beschäftigt, der Programmiersprache, die zum Schreiben von Multics-Software vorgesehen war. Dennis hatte als Student in Harvard an der Multics-Dokumentation gearbeitet und kümmernte sich in den Bell Labs um die Ein- und Ausgabesysteme. Ken konzentrierte sich ebenfalls auf diese Teilsysteme, was sich bei der Arbeit an Unix noch als hilfreich erweisen sollte. Nach seinen Äußerungen in einem Interview aus dem Jahre 2019 war die Arbeit an Multics für ihn jedoch »ein Zahn in einem riesigen Zahnrad« und »führte zu etwas, das ich selbst nicht verwenden wollte«.

1968 wurde deutlich, dass Multics zwar eine gute Umgebung für die Handvoll Benutzer darstellte, die es unterstützte, aber niemals zu dem vorgesehenen System werden konnte, das in der Lage war, für die Bell Labs Computerdienste zu vertretbaren Kosten bereitzustellen. Es war einfach zu teuer. Daher verabschiedeten sich die Bell Labs im April 1969 von dem Projekt und ließen das MIT und GE allein weitermachen.

Multics wurde schließlich fertiggestellt und zumindest zu einem Erfolg erklärt. Es wurde bis zum Jahr 2000 unterstützt und verwendet, war allerdings nicht weit verbreitet. Aus Multics gingen viele gute Ideen hervor, aber sein bedeutendster Beitrag war ganz und gar ungeplant, nämlich sein Einfluss auf das kleine Betriebssystem Unix, das zumindest zum Teil als Reaktion auf die Komplexität von Multics geschaffen wurde.

Die Ursprünge von Unix

Als sich die Bell Labs aus dem Multics-Projekt zurückzogen, mussten sich die damit beschäftigten Mitarbeiter eine andere Aufgabe suchen. Ken Thompson (Abbildung 2.2) wollte weiterhin an Betriebssystemen arbeiten, aber nach den Erfahrungen mit Multics war das obere Management ein gebranntes Kind und scheute den Kauf neuer Hardware für ein weiteres Betriebssystemprojekt. Daher nutzten Ken und andere ihre Zeit dazu, Ideen zu entwickeln und theoretische Designs für verschiedene Komponenten von Betriebssystemen zu entwickeln, ohne sie konkret zu implementieren.

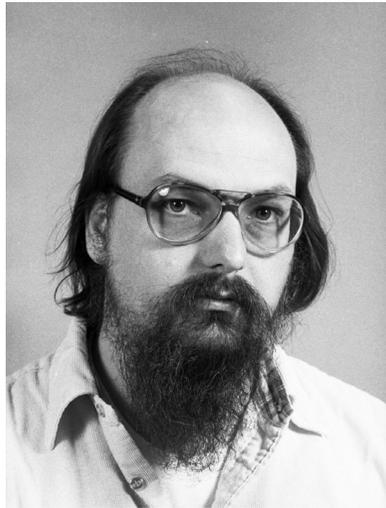


Abbildung 2.2

Ken Thompson, ca. 1984 (freundlicherweise zur Verfügung gestellt von Gerard Holzmann)

Etwa um diese Zeit stieß Ken auf einen wenig genutzten Computer vom Typ DEC PDP-7, der hauptsächlich als Eingabegerät zum Erstellen von Entwürfen für elektronische Schaltungen genutzt wurde. Der PDP-7 war 1964 auf den Markt gekommen, und da Computer einer schnellen Entwicklung unterliegen, galt er 1969 schon als veraltet. Mit einem Arbeitsspeicher von lediglich 8K 18-Bit-Wörtern (16 KByte) war er nicht sehr leistungsfähig, aber da er eine hübsche grafische Anzeige hatte, schrieb Ken eine Version des Spiels Space Travel dafür. Darin konnte ein Spieler die verschiedenen Planeten des Sonnensystems besuchen und dort landen. Es hatte einen gewissen Suchtfaktor, und ich spielte es viele Stunden lang.

Die PDP-7 hatte aber noch ein anderes bemerkenswertes Peripheriegerät, nämlich ein sehr großes Plattenlaufwerk mit einer einzigen, senkrechten Platte. Es ging das glaubhafte Gerücht, dass es gefährlich sein konnte, davorzustehen, wenn irgendetwas kaputtging. Die Festplatte war außerdem zu schnell für den Computer, was ein bemerkenswertes Problem darstellte. Ken schrieb einen Festplatten-Zeitplanungsalgorithmus, um den Durchsatz beliebiger Festplatten zu maximieren, insbesondere aber dieser Platte.

Nun stellte sich aber die Frage, wie er diesen Algorithmus testen sollte. Dazu war es erforderlich, Daten auf die Platte zu laden. Ken kam zu dem Schluss, dass er ein Programm brauchte, das Mengen von Daten auf die Platte packte.

»Irgendwann bemerkte ich, dass ich drei Wochen von einem Betriebssystem entfernt war«, erinnert er sich. Er musste drei Programme schreiben, eines pro Woche: einen Editor, um Code schreiben zu können, einen Assembler, um den Code in Maschinensprache für den PDP-7 zu übersetzen, und ein »Kernel-Overlay – was man auch ein Betriebssystem nennen kann«.

Genau zu diesem Zeitpunkt fuhr Kens Frau mit ihrem ein Jahr alten Sohn zu einem dreiwöchigen Besuch zu Kens Eltern in Kalifornien, weshalb Ken drei Wochen lang ungestört arbeiten konnte. In einem Interview im Jahr 2019 sagte er: »Eine Woche, noch eine Woche und noch eine Woche, und wir hatten Unix.« Das ist nun wirklich wahre Softwareproduktivität!

Einige Jahre, nachdem Ken und ich beide unseren Abschied von den Bell Labs genommen hatten, erkundigte ich mich bei ihm noch einmal über die Geschichte, nach der er die erste Version von Unix in drei Wochen geschrieben hatte. Seine Antwort-E-Mail entspricht genau dem, was er in dem viel späteren Interview gesagt hat:

Datum: Donnerstag, 9. Jan. 2003 13:51:-:56 -0800

unix war eine dateisystem-implementierung, um durchsatz usw zu testen. einmal implementiert, war es schwer, daten zum hochladen dort hineinzubekommen. ich konnte lese/schreib-aufrufe in schleifen stellen, aber anspruchsvollere aufgaben waren so gut wie unmöglich. so sah es aus, als bonnie meine eltern in san diego besuchen ging.

ich erkannte, dass ich kurz vor einem timesharing-system stand, dem nur ein exec-aufruf, eine shell, ein editor und ein assembler fehlten. (kein compiler.) der exec-aufruf war trivial und die anderen 3 erledigte ich 1 pro woche - genauso lange, wie bonnie weg war.

der computer hatte 8k x 18 bits. 4k war der kernel und 4k war die auslagerungsdatei für den benutzer.

ken

Diese erste Version eines schon als Unix erkennbaren Systems lief Mitte bis Ende 1969. Es ist durchaus angebracht, dies als die Geburt von Unix zu bezeichnen.

Das System hatte in diesem Frühstadium nur eine kleine Gruppe von Benutzern. Dazu gehörten natürlich Ken und Dennis sowie Doug McIlroy, Bob Morris, Joe Ossanna und zufällig auch ich. Jeder Benutzer hatte eine numerische Benutzer-ID. Einige der IDs gehörten auch zu Systemfunktionen statt zu menschlichen Benutzern, etwa 0 für Root oder den Super-User. Dazu kamen noch eine Reihe weiterer IDs für Sonderfälle. Die IDs für richtige Benutzer begannen bei 4. Wenn ich mich recht entsinne, hatte Dennis die 5, Ken die 6 und ich die 9. Eine einstellige Benutzer-ID auf dem ursprünglichen Unix-System gehabt zu haben, stellt schon eine gewisse Auszeichnung dar.

Namen sind Schall und Rauch

Noch in seiner Frühzeit erhielt das neue Betriebssystem für den PDP-7 seinen Namen. Die Einzelheiten sind allerdings etwas unklar.

Ich erinnere mich, dass ich in der Tür zu meinem Büro stand und mit mehreren Leuten sprach, unter denen sich zumindest Ken, Dennis und Peter Neumann befanden. Zu diesem Zeitpunkt hatte das System noch keinen Namen. Da Multics »viele von allem« bot, schlug ich daher vor, das neue System, das höchstens eines von allem hatte, als Wortspiel mit den lateinischen Wurzeln »uni« und »multi« auf den Namen UNICS zu taufen.

Allerdings erfand nach einer anderen Erinnerung Peter Neumann den Namen Unix als Abkürzung für »UNiplexed Information and Computing Service«. Er selbst sagt dazu:

»Ich kann mich noch lebhaft daran erinnern, dass Ken eines Tages zum Mittagessen vorbeikam und mir sagte, dass er über Nacht einen tausendzeiligen Einzelbenutzer-Betriebssystemkernel für den PDP-7 geschrieben hätte, den Max Matthews ihm ausgeliehen hatte. Ich schlug ihm vor, daraus ein Mehrbenutzersystem zu machen, und als er am nächsten Tag wieder zum Essen kam, hatte er weitere tausend Zeilen mit einem Mehrbenutzerkernel geschrie-

ben. Es war der Einbenutzerkernel, der die Vorstellung eines ›kastrierten Multics‹, also UNICS, aufkommen ließ.«

Großzügigerweise hat Peter gesagt, dass er sich an keine weiteren Einzelheiten mehr erinnert, weshalb mir die Prägung der Bezeichnung zugeschrieben wird, ob verdient oder nicht.

Wie dem auch immer sei, mutierte UNICS schließlich zu Unix, was deutlich besser aussieht. (Angeblich gefiel der Rechtsabteilung von AT&T das Wort »Unics« nicht, da es ähnlich wie »eunuchs« klang.) Dennis Ritchie hat den Namen später als »etwas hinterhältige Anspielung auf Multics« bezeichnet, was er auch tatsächlich war.

Biografie: Ken Thompson

Im Mai 2019 führten Ken und ich ein informelles »Kamingespräch« beim Vintage Computer Festival East in Wall, New Jersey. Ich stellte dabei einige Fragen, um die Sache ins Rollen zu bringen, lehnte mich zurück und hörte zu. Einige Teile dieses Abschnitts gehen auf diese Unterhaltung zurück. Aufzeichnungen der Veranstaltungen sind auf YouTube zu finden.

Ken wurde 1943 geboren. Da sein Vater bei der US Navy war, verbrachte Ken große Teile seiner Kindheit an verschiedenen Orten rund um die Welt, darunter in Kalifornien, Louisiana und Neapel.

Als Heranwachsender begann er, sich für Elektronik zu interessieren, weshalb er schließlich Elektrotechnik an der University of California in Berkeley studierte. Die elektronischen Anteile des Studiums waren für ihn sehr einfach, da er sich schon zehn Jahre lang als Hobby damit beschäftigt hatte. In Berkeley aber entdeckte er auch das Gebiet der Informatik für sich:

»Ich verschlang Computer, ich liebte sie. Zu der Zeit gab es noch keinen Lehrplan für Informatik in Berkeley; das wurde gerade erst erfunden.

In dem Sommer nach meinem Abschluss ließ ich mich treiben. [Der Abschluss] kam für mich überraschend, da mir nicht klar gewesen war, dass ich alle Voraussetzungen erfüllt hatte.

Ich wollte einfach an der Uni bleiben, weil ... sie mein war. Ich hatte meine Finger in absolut allem. Der Hauptmonstercomputer der Uni wurde um Mitternacht eingeschlossen, aber ich kam mit meinem Schlüssel, machte auf, und dann war er bis 8 Uhr morgens mein persönlicher Rechner.

Ich fühlte mich glücklich. Keinerlei Ehrgeiz. Ich war ein Workaholic, aber ohne Ziel.«

In seinem letzten Jahr nahm Ken an einem Kurs teil, der von Elwyn Berlekamp gehalten wurde, einem Berkeley-Professor, der kurz danach zu den Bell Labs ging. Im Sommer nach dem Abschluss meldete sich Ken nicht für ein Masterstudium an, da er nicht glaubte, gut genug dafür zu sein.

»Gegen Ende des Sommers sagte [Berlekamp] zu mir: ›Das hier sind deine Kurse für das Masterstudium.« Er hatte mich angemeldet, und ich wurde angenommen.«

Als Ken 1966 sein Masterstudium in Berkeley abschloss, versuchten mehrere Unternehmen, ihn für sich zu gewinnen, darunter die Bell Labs. Er aber sagte klipp und klar, dass er nicht für ein Unternehmen arbeiten wollte.

Der Headhunter blieb jedoch hartnäckig. Ken berichtet: »Ich ließ etwa sechs oder acht Angebote der Bell Labs links liegen. Wie gesagt, kein Ehrgeiz. Irgendwann klopfte der Personalmensch der Bell Labs bei mir zu Hause an die Tür. Ich ließ ihn herein, und er behauptet, ich hätte ihm Ingwerkekse und Bier angeboten.« (Das muss wohl Bestandteil der merkwürdigen kalifornischen Ernährungsweise sein.)

Schließlich willigte Ken ein, auf Kosten der Bell Labs nach New Jersey zu kommen – aber nur für einen Tag und hauptsächlich, um Freunde aus Highschool-Zeiten zu besuchen. Als er aber die Bell Labs betrat, war er von den Namen an den Türen beeindruckt:

»Als Erstes ging ich durch den Korridor der Abteilung Informatikforschung. Ich kannte alle Namen an den Türen. Es war einfach erschütternd. Dann führten zwei ganz erstaunliche Menschen das Vorstellungsgespräch mit mir – einer davon war Shen Lin.

Abends fuhr ich in meinem Leihwagen wieder weg. Aber sie blieben auf meiner Fährte, und etwa bei meinem dritten Zwischen-

stopp entlang der Ostküste wartete bereits ein Angebot auf mich. Ich nahm es mit, und während der etwa zwei Stunden zum nächsten Zwischenstopp dachte ich darüber nach. Als ich bei einem Freund zu Hause ankam, rief ich sie an und sagte zu.»

1966 trat Ken den Bell Labs bei, wo er wie bereits beschrieben zuerst an Multics und dann an Unix arbeitete, weshalb ich es hier nicht wiederholen möchte.

Ken hatte sich schon lange für Spiele interessiert und als Kind für Schach begeistert. Er mochte nicht verlieren, aber wenn er gewann, tat ihm sein Gegner leid, weshalb er Schachspiele schließlich nur noch als Zuschauer verfolgte. 1971 schrieb er ein Schachprogramm für die PDP-11. Dieses Projekt war so vielversprechend, dass er eigens für diesen Zweck eine besondere Hardware konstruierte, um die erforderlichen Berechnungen zu beschleunigen, etwa diejenigen, um die von einer gegebenen Position aus gültigen Züge zu bestimmen. Das führte schließlich zu dem Schachcomputer Belle, den er und Joe Condon von 1976 bis 1980 entwickelten (Abbildung 2.3).



Abbildung 2.3
Ken Thompson und Joe Condon (Computer History Museum)

Belle (Abbildung 2.4) war eine erfolgreiche Karriere vergönnt. Als erster Schachcomputer wurde sie Schachmeister mit einer Wertungszahl von 2200 im regulären Turnierspiel gegen menschliche Gegner. Überdies gewann sie die Computerschach-Weltmeisterschaft 1980 sowie mehrere ACM-Computerschachturniere, bevor sie in der Smithsonian Institution in den Ruhestand ging.

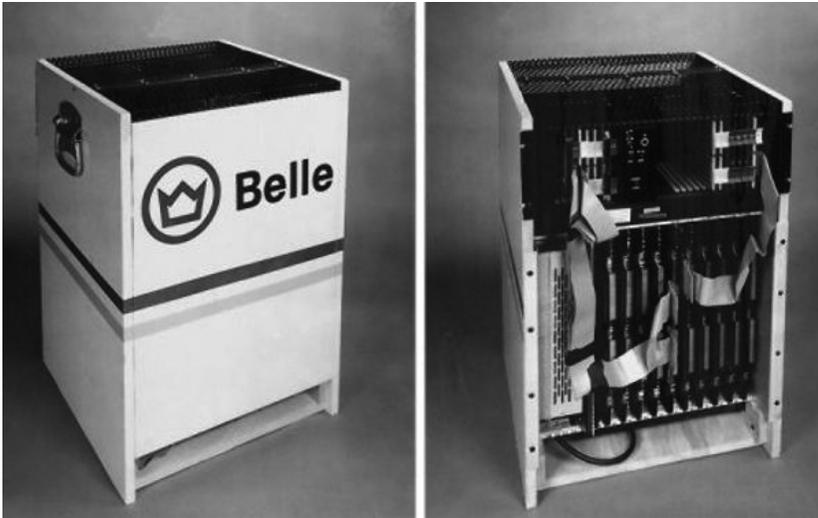


Abbildung 2.4

Der Schachcomputer Belle (mit freundlicher Genehmigung des Computer History Museum)

Dennis Ritchie schrieb einen kurzen Artikel für die International Computer Chess Association, in dem er über Ken Thompsons Arbeiten an mehreren Spielen berichtete (zu finden auf www.bell-labs.com/usr/dmr/www/ken-games.html). Darin wird deutlich, wie breit gestreut Kens Interesse an Spielen außer Schach noch war. Der Artikel enthält auch das Protokoll von Belles Sieg über Blitz 6.5 bei der ACM-Computerschachmeisterschaft am 5. Dezember 1978 mit Kommentaren des Computerschachpioniers Monty Newborn und des internationalen Meisters David Levy:

1. e4 e5 2. Nf3 Nc6 3. Nc3 Nf6 4. Bb5 Nd4 5. Bc4 Bc5 6. Nxe5
Qe7 7. Bxf7+ Kf8 8. Ng6+ hxc6 9. Bc4 Nxe4 10. O-O Rxb2!! 11.
Kxh2 {beschleunigt die Niederlage} Qh4+ 12. Kg1 Ng3 13. Qh5
{wirkunglose Verzögerung} gxh5 14. fxg3+ Nf3# {blockiert auf
wohl einzigartige Weise ein Schach und gibt gleichzeitig doppeltes
Schachmatt; »die schönste Kombination, die bis heute von einem
Computerprogramm geschaffen wurde ... der Beginn einer neuen
Ära im Computerschach.«}

Ein Schachspiel endet mit einem Sieg, einer Niederlage oder einem Remis. Laut der 50-Züge-Regel kann ein Spieler ein Remis beantragen, wenn in den letzten 50 Zügen keine Figur geschlagen und kein Bauer bewegt wurde. Das soll verhindern, dass ein Spieler einfach weitermacht, obwohl er keine Möglichkeit mehr hat, einen Sieg zu erzielen.

Ken wollte herausfinden, ob 50 Züge tatsächlich einen sinnvollen Grenzwert darstellen. Mithilfe von Belle und einigen anspruchsvollen Datenbankstrukturen untersuchte er alle Endspiele mit vier oder fünf Figuren. Dabei erkannte er, dass bei optimalem Spiel einige davon in mehr als 50 Zügen zu gewinnen waren. Zu diesem Zeitpunkt war Ken in der Schachwelt bereits gut bekannt. Manchmal besuchten sogar einige Großmeister die Bell Labs, um sich an einem Spiel gegen Belle zu versuchen, insbesondere einem Endspiel. So begegnete ich den Weltmeistern Anatoli Karpow und Vishy Anand, weil ich gerade an den richtigen Wochenenden anwesend war.

Außerdem war Ken ein begeisterter Pilot. Vom Flugplatz in Morristown aus unternahm er regelmäßig Flüge in und um New Jersey und nahm auch Gäste mit. Oft gelang es ihm, andere Mitarbeiter von 1127 für das Fliegen zu begeistern. Zu Spitzenzeiten umfasste die »1127 Air Force« ein halbes Dutzend Privatpiloten. Diese Gruppe pflegte Ausflüge zu unternehmen, um den Indian Summer zu bewundern, und zum Mittagessen an sehenswerte Orte zu fliegen. Doug McIlroy erinnert sich:

»Neben dem Herbstlaub in Neuengland konnte die Air Force dank Kens Pilotenkünsten und den von Rob Pike bereitgestellten Teleskopen auch eine Sonnenfinsternis in den Adirondacks beobachten. Es gab auch einen Flug, um einen Merkurdurchgang zu verfolgen. Das Interesse für Astronomie unter der Unix-Mannschaft hatte mit Joe Ossannas Programme azel seinen Anfang genommen. Es steuerte die Telstar-Bodenstation und wir teilten uns immer mit, wo gerade Satelliten zu finden waren. Danach kam Bob Morris' Programm sky, dann Kens Vorhersageprogramm für astronomische Ereignisse, die mit meinem Programm map erstellten Sternkarten von Lee McMahon und schließlich Robs Sternkatalog scat.«

Im Dezember 1992 reisten Ken und Fred Grampp nach Moskau, um eine MiG-29 zu fliegen, was eine erhebliche Steigerung gegenüber ihren üblichen Cessnas darstellte. Abbildung 2.5 zeigt Ken vor dem Start, Abbildung 2.6 beim Ausrollen nach der Landung.



Abbildung 2.5
Ken Thompson bei der Vorbereitung zum Start (mit freundlicher Genehmigung von cat-v.org)



Abbildung 2.6

Ken beim Ausrollen nach dem Flug (mit freundlicher Genehmigung von cat-v.org)

Ende 2000 verabschiedeten sich Ken und ich von den Bell Labs. Während ich nach Princeton ging, fing er bei Entrisphere an, einem von Bell-Labs-Kollegen gegründeten neuen Unternehmen. 2006 wechselte er zu Google, wo er zusammen mit Rob Pike und Robert Griesemer die Programmiersprache Go entwickelte. Da ich über seinen Wechsel von Entrisphere zu Google von jemand anderem gehört hatte, bat ich ihn um Bestätigung. Seine Antwort lautete:

Datum: Mittwoch, 1. Nov. 2006 16:08:31 -0800

Betreff: Antwort: stimmen aus der Vergangenheit.

es ist wahr. ich habe den altersmedian von google nicht so sehr verschoben, aber ich glaube, ich habe den durchschnitt erheblich gedrückt.

ken