Fred P. Brooks:
A Software Pioneer

Armin Grodon
Hochschule für Angewandte Wissenschaften München
Fakultät für Informatik und Mathematik

10.12.2015

Inhaltsverzeichnis

1 Fred Brooks

Frederick Phillips Brooks, Jr. wurde am 19. April 1931 in Durham, North Carolina, geboren[1]. Zunächst studierte er Physik an der Duke University in seiner Heimatstadt. Nachdem er sein Bachelorstudium 1953 mit Auszeichnung abschloss, wechselte er zum Informatikstudium (damals noch Angewandte Mathematik) an die Harvard University. Dort erlangte er 1956, mit der Arbeit "The Analytic Design of Automatic Data Processing Systems", seinen Doktortitel. Sein Doktorvater war Howard Aiken, der Entwickler des Harvard Mark I[2].
Bekannt wurde er hauptsächlich durch seine Arbeit bei IBM, als Projektleiter für die Entwicklung der Computerfamilie System/360 und darauffolgend als Projektleiter für das Design des dafür entwickelten Betriebssystems OS/360, sowie die Veröffentlichung seiner Management-Erfahrungen im Buch "The Mythical Man Month"[1].

Abbildung 1: Fred Brooks in Berlin[(2)]

2 IBM

Von 1956 bis 1965 arbeitete Brooks bei IBM in New York[1]. Dort war er zunächst als Architekt für die Supercomputer Stretch und Harvest beschäftigt. Später auch als Manager für die Architektur der IBM 8000 Serie, welche allerdings nie gebaut wurde.
IBM Stretch, beziehungsweise IBM 7030 Data Processing System, war der damals schnellste Supercomputer der Welt[3]. Der Computer wurde für das Los Alamos National Laboratory entwickelt und nahm fast 190m^2 Fläche ein. Der Name Stretch wurde gewählt, da das System als Basis für mehrere Modelle geplant war und sich über das ganze Spektrum der nächsten

[(2)] https://commons.wikimedia.org/wiki/File:Fred_Brooks.jpg

IBM-Rechner erstrecken sollte[4]. Allerdings wurde der IBM Stretch ein finanzieller Misserfolg. Aus Angst, das Los Alamos National Laboratory an den Konkurrenten UNIVAC zu verlieren, wurden sehr hoch gesteckte Leistungen versprochen, welche am Ende nicht erreicht werden konnten. Anstatt der angekündigten 4 MIPS schaffte der Stretch im fertigen Zustand nur noch 1.2 MIPS. Das System wurde direkt vom Markt genommen und der Preis für die bereits verkauften Modelle von den geplanten 13,5 Millionen Dollar auf 7,8 Millionen Dollar reduziert. Trotz des finanziellen Schadens und der nicht erreichten, technischen Ziele, brachte der IBM Stretch einige revolutionäre Konzepte. Zum Beispiel wurde das von Brooks mitentwickelte Interrupt-System auch für nachfolgende Architekturen eingesetzt und bildet heute die Grundlage für die meisten verwendeten Interrupt-Systeme.

IBM Harvest, beziehungsweise IBM 7950, wurde als Zusatz zum Stretch entwickelt, war in etwa doppelt so groß und wurde zur Kryptoanalyse für die NSA gebaut[4]. Während der Entwicklung einigte man sich auf eine Zeichenlänge von 8 statt den damals üblichen 6 Bit und dafür den Namen "Byte" zu verwenden[5]. Dieses 8-Bit-Byte wurde später von Fred Brooks für OS/360 übernommen und sorgte für die Verbreitung als Standard, sowie die verstärkte Entwicklung von 8-Bit Ein- und Ausgabegeräten.

3 OS/360

Anfang der 60er Jahre hatte IBM einen Marktanteil von etwa 70%, weshalb der Computermarkt zu dieser Zeit, mit IBM und seinen sieben Konkurrenten, auch als "Schneewittchen und die sieben Zwerge" bezeichnet wurde[6]. Allerdings hatte man bei IBM Angst, durch Projekte wie Stretch, seine Marktposition zu verlieren. Die bestehenden IBM Computer konnten architekturbedingt nicht so viel Speicher adressieren, wie benötigt und bezahlbar wurde[1]. Zusätzlich kam das IBM 2311 Festplattenlaufwerk auf den Markt. Dieses wurde zwar im Stretch testweise eingesetzt, allerdings zeigte sich dabei, dass für die effektive Nutzung ein komplexeres Betriebssystem notwendig war. Aus diesen Gründen entschied sich IBM 1961, in einem zweiten Versuch, nach Stretch, die bestehenden Computer durch eine neue Computerserie abzulösen[7]. Geplant war eine einheitliche, modulare und vollständig auf- und abwärtskompatible Familie aus 7 Computern, welche sowohl leistungsschwache Geräte als auch neue Supercomputer umfasste. Dadurch erhoffte

man sich, ein einzelnes Betriebssystem für die ganze Familie entwickeln zu können, welches im Gegenzug wesentlich umfangreicher und qualitativ hochwertiger ausgearbeitet werden könnte. Zusätzlich wollte man für alle Modelle das Festplattenlaufwerk IBM 2311 verwenden. Mit Ausnahme des leistungsschwächsten Modells ließen sich diese Ideen auch komplett umsetzen.

3.1 Grundlegende Neuerungen

Abbildung 2: IBM 2311 Festplattenlaufwerk(4)

Eine wichtige Neuerung war das bereits angesprochene Festplattenlaufwerk. Durch dieses war OS/360 das erste Betriebssystem, welches für alle Systemkomponenten direkten Zugriff, im Vergleich zu sequentiellem Zugriff bei Bändern oder Karten, annehmen konnte. Nachdem dieses Laufwerk im IBM Stretch getestet wurde, wurde es verpflichtend für das System/360 übernommen. Neben dem Geschwindigkeitsvorteil durch direkten Zugriff und generell kürzere Zugriffszeiten, ermöglichte das Festplattenlaufwerk die Entwicklung eines größeren und umfangreicheren Betriebssystems[7].
Die zweite große Besonderheit war die Unterstützung für echtes Multitasking unabhängiger Jobs. Vorangehende Betriebssysteme unterstützten entweder nur rein sequentielles Abarbeiten von Aufgaben oder das gleichzeitige Bearbeiten mehrerer Aufgaben mit dem gleichen Programm. Da ein Programm dadurch das komplette System für sich allein hatte, musste sich auch weder der Programmierer, noch das Betriebssystem über Ressourcenkonflikte Gedanken machen. IBM Stretch verwendete bereits die Technik Simultaneous Peripheral Operation On Line (SPOOL), welche es ermöglicht, Daten zwischen Datenträgern zu kopieren, während eine andere Aufgabe abgearbeitet wird. Dies diente Hauptsächlich dem Zweck, Daten während der Verarbeitung von einem langsamen Medium, wie Bänder oder Lochkarten, in einem schnelleren Pufferspei-

(4) https://commons.wikimedia.org/wiki/File:IBM_2311_memory_unit.JPG

cher für die Bearbeitung abzulegen. Durch die Einführung eines Supervisors, welcher alle Systemressourcen verwaltete, war es in OS/360 auch möglich, Programme gleichzeitig abzuarbeiten, selbst wenn diese nicht speziell dafür programmiert waren. In OS/360 war es dagegen aber auch möglich, dass sich mehrere Tasks eine CPU teilten (**multiprogramming**) oder ein Task mehrere CPUs nutzen konnte (**multiprocessing**).
Auch wurde ein Kontrollprogramm entwickelt, welches dem "Operator" Arbeiten wie das Starten, Stoppen und Priorisieren von Arbeitsaufträgen abnahm. Er konnte, wenn er wollte, noch Eingreifen um beispielsweise Aufträge abzubrechen oder anders zu priorisieren, musste dies allerdings nicht mehr. Seine neue Hauptaufgabe bestand dadurch darin, Peripheriegeräte und Datenträger zu mounten, wenn ein Auftrag diese benötigte und ihm dies über das Betriebssystem mitteilte. Auch konnten Arbeitsaufträge direkt von Benutzern an entfernten Terminals, welche mit dem Hauptrechner verbunden waren, angelegt werden, anstatt diese vom "Operator" eintragen zu lassen.
Durch die anfangs erwähnte Modularität war es auch möglich, ein System/360 bei Bedarf durch schnellere und bessere Bausteine zu erweitern, anstatt sich einen komplett neuen Computer kaufen zu müssen. Mit einem speziellen Prozess namens **system generation** konnte dann das Betriebssystem an die neue Hardware angepasst werden. Beispielsweise um die bestehenden Programme durch schnellere zu ersetzen und um Features zu aktivieren, die durch Hardwarebeschränkungen vorher nicht möglich waren. Zu diesem Zweck wurden viele der Programme in mehreren Versionen mitgeliefert; einer langsamen mit minimalem Funktionsumfang und einer schnellen, dafür aber großen und rechenintensiven.

3.2 Design

Die Grundidee bei dem Entwurf von OS/360 war die verstärkte Abstraktion, welche von George Mealy mit den Schichten einer Zwiebel verglichen wird[7]. Durch diese Abstraktion müsste ein Programmierer beim Zugriff auf Hardware oder Daten nicht mehr spezielle, auf die verwendete Hardware angepasste Maschinensprache verwenden, sondern könnte dies mit Hochsprach-Makros machen, welche vom Betriebssystem übersetzt oder interpretiert werden. Der Programmierer arbeitete also nicht mehr direkt auf der innersten Schicht, der Hardware, sondern auf einer darüber liegenden Schicht, welche unterschiedliche Hardware in einem allgemeinen Interface zusammenfasst.
Außerdem wurden die möglichen Systemzustände auf zwei beschränkt. Das

Betriebssystem ist die meiste Zeit im unprivilegierten **problem state**, was dem **user mode** späterer Unix-Systeme entspricht. In diesem Zustand werden alle Benutzerprogramme, beziehungsweise **problem programs**, ausgeführt. Lediglich der Supervisor läuft im **supervisor state**, was dem heutigen **kernel mode** entspricht. Möchte der Benutzer auf privilegierte Funktionen, wie Dateizugriff, zugreifen, muss er dies über den Supervisor veranlassen. Dies geschieht, indem ein Benutzerprogramm eine **system macroinstruction**, heute der **system call**, ausführt.
Neben der oben genannten Multitasking-Fähigkeit wurde auch als Grundkonzept festgelegt, dass jegliche Kombination aus Programm und zu verarbeitenden Daten als **task** bezeichnet und gleich behandelt wird. Dadurch wurden die in vorangehenden Betriebssystemen sehr unterschiedlich Verarbeitungsweisen für **multiple instruction, single data** und **single instruction, multiple data** vereinheitlicht.
Ein weiteres Konzept war, dass alle Daten, egal ob Text, Quellcode oder Programm, in gleicher Weise als **data set** behandelt wurden[8]. Dieses Prinzip findet sich auch in der Form von "Everything is a file" im später entstehenden Unix wieder.
Auch wurde festgelegt, dass alle Ressourcen des Rechners, auch die CPU, vom System verwaltet und bei Bedarf von Programmen allokiert werden mussten[9].

3.3 Aufbau

Fred Brooks bezeichnet den Aufbau von OS/360 als "One big peach and a bowl full of independent cherries"[10]. Der Pfirsich entspricht dabei dem Kontrollprogramm, welches sich wiederum aus **master scheduler**, **job scheduler** und **supervisor** zusammensetzt, den Großteil des Betriebssystems ausmacht, und dem heutigen **kernel** entspricht[7]. Das Kontrollprogramm wurde durch den Task-Manager des Stretch beeinflusst, allerdings im Funktionsumfang stark erweitert. Die Kirschen entsprechen der Sammlung an Compilern und sonstigen Programmen, welche im Betriebssystem enthalten sind und während der **system generation** in das verwendete System eingebunden oder weggelassen werden können. Neben den Compilern enthält das Betriebssystem auch mehrere Makrogeneratoren, um **system macroinstructions** in Hochsprachen in Assembleranweisungen zu übersetzen.

3.3.1 Supervisor

Der Supervisor ist, wie der Name "Aufseher" oder "Leiter" schon vermuten lässt, allen anderen Programmen übergeordnet und damit der zentrale Teil des Kontrollprogramms[7].
Seine Aufgaben sind:

- Verwaltung des Systemspeichers
- Laden von Programmen und Programmmodulen in den Speicher
- Steuerung der Nebenläufigkeit von Tasks
- Bereitstellen und Abarbeiten von Exceptions
- Bereitstellen von System-Diensten wie Clock, Logging und Monitoring

Kontrolle erlangt der Supervisor über Interrupts, welche von Programmen oder dem System selbst ausgelöst werden können.

3.3.2 Job-Scheduler

Der Job-Scheduler übernimmt einen Großteil der Aufgaben, welche zuvor vom Operator erledigt werden mussten[7]. Nämlich das Analysieren und Optimieren der eingehenden **Jobs** und das Initialisieren und Terminieren der einzelnen **Job Steps**, also das Erzeugen von **Tasks** aus **Job Steps** und die Abschließende Rückgabe deren Ergebnis. Zusätzlich verwaltet er die Ein- und Ausgabegeräte, um die Nebenläufigkeit der **Jobs** sicher zu stellen.

Ein **Job** definiert dabei eine unabhängige Arbeitseinheit. Dadurch ist es nicht möglich, einen **Job** auf einen vorangehenden oder dessen Ergebnisse warten zu lassen[9]. Dieses Verhalten lässt sich aber durch sogenannte **Job Steps** umsetzen. Die **Job Steps** sind alle sequentiellen Arbeitsschritte eines **Jobs**, können sich untereinander beeinflussen und ihre Ergebnisse an nachfolgende Schritte durchreichen. Die Arbeit, welche ein Programm während eines **Job Steps** ausführt, wird als **Task** bezeichnet. Diese **Tasks** können bei Bedarf selbst weitere Tasks, sogenannte **Subtasks**, in einer hierarchischen Struktur erzeugen. Die Anforderungen eines **Jobs** werden in einem **Control Statement** festgehalten, welches mit eventuellen, neuen Eingabedaten im **Job Stream** gruppiert wird. Die **Tasks** unterschiedlicher **Jobs** können dann, solange sich aus dem **Control Statement** keine Überschneidungen in den Anforderungen ergeben, parallel abgearbeitet werden.

3.3.3 Master-Scheduler

Der Master-Scheduler bildet die Schnittstelle zum Operator[7]. Über diese kann der Operator einerseits bei Bedarf Tasks selbst starten oder stoppen und wird andererseits darüber benachrichtigt, wenn Datenträger gemountet werden müssen, weil Programme die darauf enthaltenen Daten benötigen.

3.4 Programmstrukturen und Wiederverwendbarkeit

Zur Strukturierung von Programmen wurden mehrere Verfahren definiert[9]. Für überwiegend kleine und einfache Programme gab es die **simple structure**, in der das Programm aus nur einem **load module** bestand, welches auch komplett im Speicher gehalten werden musste. Um auch weiterhin die Entwicklung in Overlays zu ermöglichen, gab es die **planed overlay structure**. In dieser legte der Programmierer selbst fest, welche Teile seines Programms von anderen Teilen "überlagert" werden dürfen. Die zwei neuartigen Strukturen waren die **dynamic serial structure** und die **dynamic parallel structure**. Hier musste sich der Entwickler nicht mehr selbst um die optimale Ausnutzung des Speichers kümmern, sondern sein Programm lediglich in einzelne Unterprogramme, sogenannte **load modules**, unterteilen und diese dann im Programm referenzieren. Der Unterschied der beiden Strukturen untereinander bestand dabei darin, dass durch die Verwendung unterschiedlicher Makrobefehle Unterprogramme in einen seriellen Programmfluss geordnet (LINK, XCTL, LOAD) oder in eigenständigen, parallelen Tasks gestartet (ATTACH) wurden.
Bei den Unterprogrammen konnte der Programmierer auch bestimmen, ob diese von nachkommenden Tasks generell wiederverwendet oder eventuell sogar von mehreren Tasks gleichzeitig durchlaufen werden können.

3.5 Job und Task Management

Das **Job Management** definiert Arbeit und delegiert sie anschließend an das **Task Management**[9]. Es durchsucht alle **Job Steps** nach benötigten Datenträgern und stellt sicher, dass alle Ressourcen für den Job Step bereitgestellt werden. Diese Arbeiten erledigt es über die oben genannten Scheduler.

Das **Task Management** wiederum steuert den eigentlichen Arbeitsablauf[9].

Es verwaltet alle **Tasks**, mitsamt Kontext (Register, allokierte Adressen und weiteren Informationen) in einer priorisierten Schlange und lädt nach und nach die am höchsten priorisierten Tasks, welche nicht durch das Warten auf Ressourcen oder andere Tasks blockiert werden. Die Priorität setzt sich dabei aus mehreren Faktoren zusammen, kann aber zusätzlich noch vom Benutzer und Operator beeinflusst werden. Das **Task Management** des OS/360 hatte bereits sehr einfache Implementierungen für einen Speicherschutz, gemeinsam nutzbare Speicher-Pools für Tasks eines Jobs und einen Swapping-Mechanismus, welcher als **roll-in** und **roll-out** bezeichnet wurde, um bei Speicherknappheit Tasks mit Kontext auf ein Speichermedium auszulagern und bei Bedarf wieder von diesem zu laden.

3.6 Data Management

Das **Data Management** entstand als Reaktion auf die wachsende Zahl an verschiedenen Datenträgern und Speichermedien, und die neuen Anforderungen durch immer größere und schnellere Speicher[8]. Statt im Programmcode direkt Gerätetypen zu benennen, wurden diese in Geräteklassen unterteilt, welche dann zur Laufzeit zu tatsächlichen Geräten aufgelöst wurden. Da sich dabei an bestehenden Systemen orientiert wurde, gab es auch hier wieder eine Auswahl an verschiedenen Ein- und Ausgaberoutinen. Um bestehende Programme leichter portieren zu können und kleinere Programme einfach und schnell zu halten, gab es Routinen, welche während der Übersetzung aufgelöst wurden. Diese waren zwar schnell, allerdings sehr unflexibel, da der Programmierer sich schon während der Entwicklung festlegen musste, welche Geräte später zum Einsatz kommen. Etwas flexibler waren interpretierte Routinen, welche beim Laden des Programmes aufgelöst wurden. Diese waren zwar sehr flexibel, dafür aber auch sehr langsam. Neu waren generierte Routinen, welche in Form von Zugriffsroutinen zur Laufzeit aufgelöst wurden. Da diese Routinen von mehreren Programmen gemeinsam genutzt und dadurch im Speicher gehalten werden konnten, war diese Umsetzung sowohl flexibel, als auch sehr schnell.

Wie anfangs schon erwähnt, wurden alle Daten als sogenannte **Data Sets** bezeichnet[8]. Diese bestanden im allgemeinen aus einem **Data Set Label**, welches sich wiederum aus dem Namen, dem Speicherbereich und einigen weiteren Parametern des Data Sets zusammensetzte, und mit welchem sich das Data Set in einem **Volume** adressieren lassen konnte. Als **Volume** wur-

de jeglicher Zusatzspeicher bezeichnet. Diese waren über ein **Volume Label**, bestehend aus Seriennummer und weiteren Informationen, identifizierbar. Wollte man ein bestimmtes **Data Set** finden, tat man dies über den **Data-Set Catalog**. Dieser wurde immer im Speicher gehalten und enthielt alle **Data Set Labels** in einer Baumstruktur.
Eine zusätzliche Funktionalität von **Data Sets** waren die sogenannten **Generation Groups**, welche eine Art einfaches Versionsverwaltungssystem ermöglichten. Durch eine zuvor definierte Änderungstiefe, konnte man mit Indizes auf ältere Versionen eines Data Sets zugreifen.
Durch die Vergabe von Passwörtern für Data Sets ließen sich diese vor unerlaubtem Zugriff durch andere schützen. Wollte man ein solches Data Set laden, musste das Passwort über ein Terminal eingegeben werden.
Auch war es möglich Data Sets als **Buffer** zu nutzen, um beispielsweise Ein- und Ausgabedaten zwischenzuspeichern.

3.7 Datenzugriff

Um für alle Datenorganisationen und Zugriffs-Arten optimierten und einfachen Zugriff zu gewährleisten, wurden die Zugriffsmethoden in Tabelle 1 mit jeweils eigenen Makrosprachen entwickelt.

Tabelle 1: Zugriffsmethoden nach Datenorganisation und Zugriffs-Art

	Queued	Basic
Sequential	QSAM	BSAM
Indexed Sequential	QISAM	BISAM
Direct		BDAM
Partitioned		BPAM
Telecommunication	QTAM	BTAM

Dadurch entstanden zwar eine Vielzahl an sehr spezifischen Makrosprachen, allerdings ermöglichte dies auch sehr schnelle Zugriffsmethoden für nichtsequentielle Datenträger wie Festplatten, oder bestimmte Datenmodelle, wie sie zum Beispiel von Telekommunikationsgesellschaften genutzt wurden.

4 Auswirkungen von OS/360

Obwohl es sehr langsam und schwer zu bedienen war, wurde OS/360 ein großer finanzieller Erfolg[6]. Allerdings musste es durch Preis- und Zeitdruck in mehrere Versionen geteilt werden, von denen die Entwicklung einer Version sogar komplett eingestellt wurde. Trotz allem schaffte es IBM einen Architekturstandard zu definieren, weiter Marktführer zu bleiben und sogar die Konkurrenz von sieben auf fünf Unternehmen zu reduzieren.

4.1 Auswirkung für weitere Betriebssysteme

Einige der Ideen in OS/360 hatten aber auch Auswirkungen, welche sich weit über IBM und IBMs nachfolgende Betriebssysteme erstreckte.
Vornehmlich waren dies[6]:

- Verwendung von 32 Bit Wortlänge
- 32 und 64 Bit Fließkommazahlen
- Strings variabler Länge
- Verwendung von nahezu ausschließlich Universalregistern
- 32 Bit Adressen
- 8-Bit-Byte als Standard
- Look-Ahead Scheduling
- Gemeinsam genutzter Speicher
- Ein Betriebssystem für verschiedene Anwendungsfälle und Hardware

Nicht alle dieser Ideen waren komplett neu oder allein von IBM verwendet, durch die weite Verbreitung von OS/360 bekamen sie allerdings ein viel höheres Gewicht. Besonders die Verwendung des 8-Bit-Byte sorgte für eine vermehrte Entwicklung von standardisierten Ein- und Ausgabegeräten.

4.1.1 Multics und Unix

Ein Betriebssystem, dass heute nicht mehr wegzudenken ist und welches ohne OS/360 vielleicht nie entwickelt worden wäre, ist Unix.
1965 begann am MIT, in Zusammenarbeit mit General Electrics und Bell Labs, die Entwicklung für das in PL/1 geschriebene Betriebssystem Multics[11]. Ken Thompson und Dennis Ritchie arbeiteten zu dieser Zeit mit Multics, waren aber weitgehend unzufrieden mit einigen Konzepten[12]. Allerdings wurde die Entwicklung von Multics eingestellt, als Bell Labs aus dem Projekt ausstieg und General Electrics die Computer-Abteilung an Honeywell verkaufte. Dadurch sahen sich Thompson und Ritchie 1969 gezwungen, selbst einen Nachfolger zu entwickeln, welchen sie 1971 in Form von Unix veröffentlichten.

5 IBM heute

Auch heute noch hat IBM einen Marktanteil von etwa 90% im Bereich Mainframes[13]. Mit dem IBM PC 5150 waren sie zwar auch sehr erfolgreich im Bereich der Heim- und "Personal-Computer", zogen sich allerdings 2005, mit dem Verkauf der Thinkpad-Serie an Lenovo, endgültig daraus zurück[14].

Abbildung 3: System z Mainframes[(5)]

5.1 System z

Das Aktuelle Mainframe ist das System z mit z/OS. Auch hier zeigt sich der bahnbrechende Erfolg des von Fred Brooks entwickelten Designs. In Fortführung der Betriebssystemserie (OS/360 MVT, OS/370 MVS, OS/390, z/OS) ist es immer noch kompatibel mit Software, die ursprünglich für OS/360 entwickelt wurde[15].

[(5)] https://commons.wikimedia.org/wiki/File:System_z_Frames.JPG

5.2 Virtualisiertes MVS

Durch Virtualisierung ist es allerdings auch auch möglich, die Hardware zu emulieren und damit bestehende Programme weiterhin in OS/370 zu betreiben.

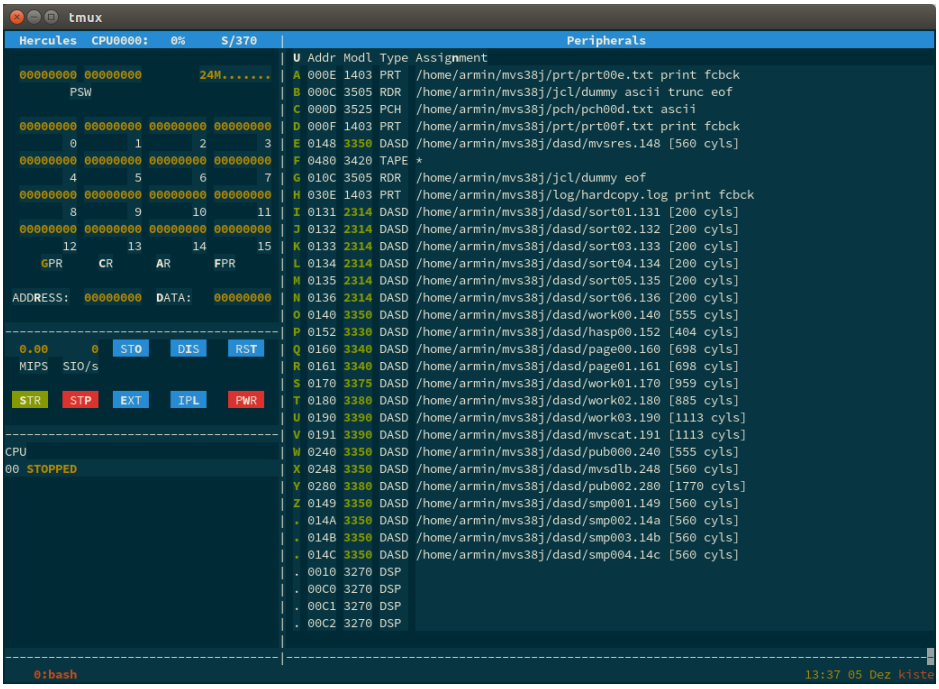

Abbildung 4: MVS/370 in Hercules in Ubuntu 15.10

6 The Mythical Man Month

1964 wechselte Brooks, noch vor der endgültigen Fertigstellung von OS/360, an die University of North Carolina at Chapel Hill[1]. Dort gründete er den Lehrstuhl für Informatik, welchen er für 20 Jahre leitete.
Hier fing er auch an, eine "Analyse der Erfahrungen aus OS/360"[16, Vorwort] in Form von Essays nieder zu schreiben. Diese Essays veröffentlichte er 1975 gesammelt als "The Mythical Man Month". Das Buch wurde allgemein sehr positiv aufgenommen und ist auch heute, nach über 40 Jahren, immer noch zu großen Teilen gültig.

6.1 Inhalt

Das Buch befasst sich mit den administrativen und technischen Lehren der Softwareentwicklung und Projektleitung[16, Vorwort]. Auch ist ein zentrales Thema des Buches, warum die Entwicklung großer Softwareprojekte so schwer ist und wie, beziehungsweise ob, sich dies lösen lässt. Es ist nach Brooks' Aussage eine Zusammenfassung der Erfolge und Fehler der Entwicklung von OS/360[1]. An vielen Stellen enthält es, teils sehr harte, Selbstkritik und Kritik an OS/360. Auch entspringt dem Buch das nach ihm benannte Brooks'sche Gesetz:

> "Der Einsatz zusätzlicher Arbeitskräfte bei bereits verzögerten Software-Projekten verzögert sie noch mehr"[16, Kapitel 2]

Das Buch beginnt mit einer Aufzählung und Erklärung von Gründen, warum Programmieren so viel Spaß macht und was Programmierer bei der Arbeit antreibt und motiviert, beziehungsweise welche Faktoren diesen Spaß trüben und die Arbeit anstrengend und unangenehm machen können[16, Kapitel 1]. Ließt man sich als Softwareentwickler diesen Abschnitt heute durch, kann man sich, trotz der großen Zeitdifferenz, in vielen der genannten Punkte wiedererkennen.
Über den weiteren Verlauf des Buches[16, Kapitel 2-15] zählt er die Hauptgründe auf, warum seiner Meinung nach damals Softwareprojekte scheiterten und versucht Ansätze zu geben, wie sich diese Probleme lösen lassen könnten. Als ein Hauptproblem beschreibt er den Trugschluss in der Gleichsetzung zwischen Arbeitskräften und Arbeitszeit[16, Kapitel 2], dem Mann-Monat, aus welchem sich auch der Buchtitel ableitet. Dies veranschaulicht er mit dem bekannten Zitat "Das Austragen eines Kindes dauert nun einmal neun Monate, egal wie viele Frauen damit beschäftigt sind"[16, Kapitel 2]. Schon damals setzte er sich für Rapid Prototyping ("Das Pilotprojekt für den Abfalleimer"[16, Kapitel 11]) und iterative Entwicklungsverfahren[16, Kapitel 5] ein.
Neben vielen Punkten, die durch Management- und Entwicklungsprozesse, neue Programmiersprachen, technologischen Fortschritt, Entwicklungsumgebungen oder allgemein anerkannte Konventionen gelöst wurden, enthält das Buch aber auch noch Punkte, die nach wie vor nicht vollständig gelöst sind.

6.2 No Silver Bullet

Der Artikel "No Silver Bullet" wurde ursprünglich 1986 als Paper auf der Konferenz "International Federation of Information Processing" veröffentlicht, wurde aber anschließend auch in der Neuauflage zum 20-jährigen Bestehen des Buches und in der deutschen Erstauflage mit abgedruckt. Im Gegensatz zum ursprünglichen Buch, welches durchwegs akzeptiert und positiv aufgenommen wurde, wurde dieser Artikel sehr kontrovers diskutiert und Brooks als Pessimist bezeichnet[16, Kapitel 18].
Grundlage für den Artikel waren seine "Erfahrungen als Leiter einer Studie des Defense Science Boards"[16, Vorwort der Neuauflage]. Er behandelt die Frage, warum Software-Entwicklung so komplex ist, woraus sich diese Komplexität ergibt und ob sich diese, zumindest in Teilen, beseitigen lassen könnte[16, Kapitel 16]. Nach Erläuterung der Ursachen und einer Auflistung derer, die sich in den kommenden Jahren beseitigen lassen könnten, kommt er zu der Voraussage, dass keine einzelne Änderung in den nachfolgenden 10 Jahren für eine Verbesserung um den Faktor 10 sorgen wird. Wobei er mit Verbesserung die Produktivität, Zuverlässigkeit und Einfachheit zusammenfasst.
Da die Neuauflage 9 Jahre nach der Veröffentlichung des Artikels erscheint, schreibt er in ihr auch ein, auf den Artikel eingehendes, rückblickendes Kapitel, in welchem er auf die Kritik antwortet, einige fehlerhafte Voraussagen entschuldigt, aber zeigt, dass er im Großen und Ganzen mit seiner Einschätzung richtig lag[16, Kapitel 17].

7 Weitere Arbeiten und Errungenschaften

Neben den bereits genannten Leistungen war Brooks Mitglied des "National Science Board", Chairman der "Military Software Task Force" des "Defense Science Board", Mitglied des "National Science Foundation Advisory Committees" und des "National Research Council"[1].

Er veröffentlichte, zusätzlich zum bereits vorgestellten Buch "The Mythical Man Month", mehrere bedeutende und viel zitierte Arbeiten[(6)]. Darunter das 1997 mit Gerrit Blaauw verfasste Buch "Computer Architecture: Con-

[(6)]Bibliografieübersicht der ACM Digital Library: `http://dl.acm.org/author_page.cfm?id=81100077256&CFID=745618589&CFTOKEN=66174959`]

cepts and Evolution" und sein neustes Buch "Design of Design", in welchem er Designprozesse fachübergreifend analysiert und die Verbindung zwischen dem Design von Computer-Architekturen, Software, Häusern, Büchern und Organisationen herstellt[17, Vorwort].

7.1 Ehrungen und Auszeichnungen

Für seine Arbeiten erhielt er eine Vielzahl an Ehrungen und Auszeichnungen. Besonders hervorzuheben sind dabei:

- National Medal of Technology[16]
- ACM A.M. Turing Award[1]
- IEEE John von Neumann Medal
- IEEE McDowell Awards
- IEEE Computer Pioneer Award
- ACM Allen Newell and Distinguished Service Awards
- AFIPS Harry Goode Award
- Eckert-Mauchly Award
- Ehrendoktor am Swiss Federal Institute of Technology (ETH Zürich)

Zusätzlich ist er Mitglied der:

- British Computer Society (Distinguished Fellow)[1]
- ACM (Fellow)
- IEEE (Fellow)
- American Academy of Arts and Science (Fellow)
- Royal Academy of Engineering (Foreign Member)
- Royal Netherlands Academy of Arts and Science (Foreign Member)
- National Academy of Science (Member)
- National Academy of Engineering (Member)

8 Fazit

Mit seiner Arbeit prägte Fred Brooks nicht nur den Begriff "Computer-Architektur"[1], sondern ebnete auch, mit der Schaffung einer Jahrzehnte anhaltenden Architektur, den Weg für den general-purpose Computer und die heutigen Betriebssysteme. Er leitete "das größte Entwicklungsprojekt der Industriegeschichte"[18], welches das "Ende der Pionierzeit der EDV"[18] einläutete.
Wenn man die Gründe betrachtet, nach welchen seiner Auffassung nach das Programmieren "Spaß"[16, Kapitel 1] macht, und diese auf die Auswirkungen seines Lebenswerkes überträgt, wird einem schnell bewusst, warum er selbst behauptet, dies "aus reiner Leidenschaft heraus"[16, Epilog] getan zu haben.

Literatur

[1] F. P. Brooks, Jr, "Frederick Phillips Brooks, Jr. - Biography." https://www.cs.unc.edu/~brooks/FPB_BIO.CV.04.2007.pdf. (Aufgerufen am 09.01.2016).

[2] IBM, "IBM's ascc introduction: a.k.a The Harvard Mark I." http://www-03.ibm.com/ibm/history/exhibits/markI/markI_intro.html. (Aufgerufen am 09.01.2016).

[3] IBM, "IBM Archives: 7030 Data Processing System." http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7030.html. (Aufgerufen am 09.01.2016).

[4] M. W. McMurran, *ACHIEVING ACCURACY: A Legacy of Computers and Missiles*. Xlibris US, December 2008. (S. 98-100).

[5] W. A. Hunt, "Early history of harvest computer." http://www.brouhaha.com/~eric/retrocomputing/ibm/stretch/early_history_of_harvest.html. (Aufgerufen am 09.01.2016).

[6] P. E. Ceruzzi, *A History of Modern Computing*. MIT Press, 2003. (S. 144, 151-152).

[7] G. H. Mealy, "The Functional Structure of OS/360: Part I Introductory Survey," *IBM Syst. J.*, vol. 5, pp. 3–11, Mar. 1966.

[8] W. A. Clark, "The Functional Structure of OS/360: Part III Data Management," *IBM Syst. J.*, vol. 5, pp. 30–51, Mar. 1966.

[9] B. I. Witt, "The Functional Structure of OS/360: Part II Job and Task Management," *IBM Syst. J.*, vol. 5, pp. 12–29, Mar. 1966.

[10] F. P. Brooks, Jr, "Vortrag: The IBM Operating System/360," 2001.

[11] T. Van Vleck, "Multics History." http://www.multicians.org/history.html. (Aufgerufen am 09.01.2016).

[12] D. M. Ritchie, "The Evolution of the Unix Time-Sharing System," in *Proceedings of a Symposium on Language Design and Programming Methodology*, (London, UK, UK), pp. 25–36, Springer-Verlag, 1980.

[13] CCIA Staff, "IBM tightens stranglehold over mainframe market; Gets hit with antitrust complaint in Europe." http://www.ccianet.org/2008/07/ibm-with-another-mainframe-antitrust-complaint-in-europe, July 2008. (Aufgerufen am 09.01.2016).

[14] J. G. Spooner, "IBM sells PC group to Lenovo." http://www.cnet.com/news/ibm-sells-pc-group-to-lenovo, Dec. 2004. (Aufgerufen am 09.01.2016).

[15] IBM, "Mainframe strength: Continuing compatibility." http://www-01.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_compatible.htm. (Aufgerufen am 09.01.2016).

[16] F. P. Brooks, *Vom Mythos des Mann-Monats.* mitp Business, mitp-Verlag, 2003.

[17] F. P. Brooks, *Erfolgreiches Design: Essays über universelle Designprozesse mit Beispielen aus IT und Software-Entwicklung.* Hamburg : mitp, 2011.

[18] D. Borchers, "Vor 40 Jahren: der perfekte Computer | heise online." http://heise.de/-96683, Apr. 2004. (Aufgerufen am 09.01.2016).