Analyse der Apollo-Navigationscomputer hinsichtlich ihrer Softwarequalität und Robustheit

Diplomarbeit

Matthias Seidel
Matr.-Nr. 730 716

Universität Potsdam

15. Juni 2016

Erster Gutachter
Prof. Dr. Gerrit Kalkbrenner

Zweiter Gutachter
Prof. Dr. Horst Zuse

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Berlin, 15. Juni 2016

_____

Matthias Seidel

Danksagung
Mein Dank gebührt meinen Betreuern, die mich stets tatkräftig unterstützt haben: Herrn Prof. Dr. Horst Zuse und Herrn Prof. Dr. Gerrit Kalkbrenner.

Mein Interesse am Apollo-Programm geht bis in meine Kindheit zurück. Durch ein Seminar bei Herrn Prof. Zuse wurde ich auch auf den Aspekt der Computer des Apollo-Programms aufmerksam. Herr Prof. Zuse war stets für mich da und hat mir bei der Beantwortung zahlreicher Fragen außerordentlich weitergeholfen. Ebenso hat mich Herr Prof. Kalkbrenner in meinem Vorhaben stets mit großem Engagement unterstützt und mir eine Vielzahl an thematischen Ratschlägen gegeben.

Nicht zuletzt gebührt mein Dank meiner Kommilitonin Suzanne Linehan Winter, die mir in zahlreichen fruchtbaren Diskussionen wertvolle Anregungen geben konnte.

Zusammenfassung
Diese Arbeit untersucht die Navigationscomputer des Apollo-Programms, im Besonderen den Apollo Guidance Computer. Dieser wird sowohl hardware- als auch softwareseitig beschrieben und anhand veröffentlichter Dokumente der NASA, des MIT und weiterer an der Entwicklung beteiligter Institutionen, sowie eigener Analysen auf seine Fähigkeiten und Robustheit hin analysiert. Die Gefahren für Computer in der bemannten Raumfahrt werden beschrieben und die Besonderheiten, die sich daraus ergeben, werden erörtert. Darüber hinaus werden die Besonderheiten von Benutzerschnittstellen in der bemannten Raumfahrt anhand des Beispiels des Benutzerinterfaces des Apollo Guidance Computers dargestellt. Um diese umfangreiche Thematik adäquat erörtern zu können, wird in dieser Arbeit ein vielschichtiger Ansatz verfolgt: Sowohl eine Vielzahl verfügbarer historischer Dokumente werden für die Untersuchung herangezogen als auch Analysen mit eigens zu diesem Zweck erstellten Programmen und mit Methoden der Software-Analytik. Dadurch wird gezeigt, wie die Entwicklung des Apollo Guidance Computers spätere Verfahren zur zuverlässigen Erstellung von Hard- und Software für die bemannte Raumfahrt beeinflusst hat. Dementsprechend werden auch heutige Computer der bemannten Raumfahrt beschrieben. Da der Apollo Guidance Computer kein isoliert arbeitendes System war, werden auch einzelne Computer der Bodenstation vorgestellt. Der in den 1960er-Jahren stattfindende Paradigmenwechsel im Computerbereich hin zu Time-Sharing-Systemen wird ebenfalls behandelt, da dieser sich auf die Navigationscomputer des Apollo-Programms auswirkte. Am Beispiel von Fly-By-Wire-Systemen wird schließlich ein direktes Resultat der Entwicklung des Apollo Guidance Computers dargestellt und ein Ausblick auf die mögliche weitere Entwicklung gegeben.

Inhaltsverzeichnis

1 Einleitung

Heute, im 21. Jahrhundert, sind Computer allgegenwärtig. Durch fortschreitende Miniaturisierung sind inzwischen zahlreiche Alltagsgegenstände wie z.B. Radios, Uhren oder Automobile, mit Computertechnik ausgestattet, deren Leistungsfähigkeit der der Großrechner von vor einigen Jahrzehnten gleichkommt oder diese sogar noch übertrifft. Betrachtet man die Entwicklung der Computergeschichte der zurückliegenden Jahrzehnte, so fällt auf, dass in zahlreichen Publikationen zu diesem Thema die Entwicklung so dargestellt wird, dass von sehr großen Geräten zu immer kleineren übergegangen wurde (siehe z.B. [21], [136]) Die Entwicklung scheint somit also folgendermaßen verlaufen zu sein:

$$\text{Mainframe} \Rightarrow \text{Mini} \Rightarrow \text{Micro} \Rightarrow \text{PC} \Rightarrow \text{Notebook} \Rightarrow \text{Touchpad} \Rightarrow \text{Wearables}$$

Dabei wird jedoch übersehen, dass die Entwicklung keineswegs so linear erfolgte; Mainframes spielen auch heute noch eine wesentliche Rolle sowohl in der wissenschaftlichen als auch in der wirtschaftlichen Welt. Und umgekehrt gab es bereits Anfang der 1960er Jahre elektronische Tischrechner, die relativ leise und klein waren. Der erste dieser Tischrechner war ANITA (**A** **N**ew **I**nspiration **T**o **A**rithmetik, Großbritannien) von 1962 [159]. Bereits zwei Jahre später kam der erste Tischrechner mit Transistoren auf den Markt, der IME 84 (**I**ndustria **M**acchine **E**lettroniche 84, Italien) [158]. Und nicht nur die Entwicklung von Tischrechnern schritt in den 1960er-Jahren voran, es wurden auch Möglichkeiten erforscht, umfangreichere Rechnersysteme auf möglichst kleinem Raum unterzubringen.

Eine Schlüsselstellung bei dieser Entwicklung spielte das bemannte Raumfahrtprogramm der NASA. Allerdings ist anzumerken, dass die NASA keineswegs ICs, Mikrochips oder gar den Mikrocomputer erfunden hat. Jedoch wurde die zu diesem Zeitpunkt bereits stattfindende Entwicklung mit Sicherheit durch die NASA beschleunigt, da das Raumfahrtprogramm einen hohen Bedarf an elektronischen Komponenten hatte. Die NASA entwickelte auch nicht direkt selbst Computer, sondern beauftragte andere Einrichtungen damit. So bekam z.B. das MIT Instrumentation Laboratory (heute: Charles Stark Draper Laboratory) am Massachusetts Institute of Technology den Auftrag, das Navigationssystem für die Apollo-Flüge zu entwickeln. Das MIT war der erste Vertragspartner der NASA im Apollo-Programm [35], [19, S. 38ff], später folgten noch zahlreiche weitere Einrichtungen. Hersteller der Hardware des Navigationscomputers war der Rüstungskonzern Raytheon[143]. Außer solchen Neuentwicklungen setzte die NASA auch zahlreiche industrielle Großrechner ein, z.B. von UNIVAC, CDC und IBM [162, Kapitel 1, Kapitel 8]. Die NASA testete Mainframes verschiedener Hersteller auf deren Tauglichkeit für das bemannte Raumfahrtprogramm, den Zuschlag für den Zentralcomputer erhielt schließlich IBM [162, Kapitel 8]. Für die Verarbeitung von Telemetrie- und Kommunikationsdaten wurden zusätzlich UNIVAC-Systeme eingesetzt [73]. Somit waren verschiedene wirtschaftliche Unternehmen und wissenschaftliche Einrichtungen in die Entwicklung der Hard- und Software für das Apollo-Programms involviert.

Das Apollo-Programm war von hohem nationalen Prestigewert; um einen Menschen auf den Mond und sicher wieder zurück zu bringen, mussten zahlreiche Probleme aus den unterschiedlichsten Bereichen angegangen und gelöst werden. Dies erforderte natürlich auch eine Zusammenarbeit über Unternehmensgrenzen hinweg. Andererseits werden Industrieunternehmen im Allgemeinen sicher wenig Interesse daran haben, eigene, kostspielige Entwicklungen ohne Weiteres mit Anderen zu teilen. Die Frage, inwieweit das Apollo-Programm und die währenddessen außerhalb stattfindende Computerentwicklung der 1960er-Jahre sich gegenseitig beeinflussten, führt damit zu einer interessanten und komplexen Fragestellung.

Die Computertechnik machte in den 1960er-Jahren rasante Fortschritte; es wurden Methoden entwickelt, um Computerressourcen besser zu verteilen, neue Herstellungstechnologien geschaffen, neuartige Bedienkonzepte entworfen, etc. Solche Entwicklungen waren natürlich auch für das Apollo-Programm von Nutzen.

Insbesondere bei der Entwicklung des On-Bord-Navigationscomputers erhielten Aspekte, wie z.B. eine möglichst einfache Bedienung und geringe Größe, besondere Bedeutung. Der Navigationscomputer musste schließlich in die Raumkapsel passen und sich auch in Stresssituationen routiniert bedienen lassen. Darüber hinaus musste der Rechner natürlich auch robust genug sein, um starke Beschleunigungen schadlos zu überstehen und während der

jeweiligen Gesamtdauer der Mission zuverlässig zu arbeiten. Außerhalb von Apollo wurde der Aspekt der Systembedienung z.B. von Bill English und Douglas Engelbart am Stanford Research Institute (heute SRI International) angegangen, die dort 1963 die Computermaus entwickelten [37]. Ebenfalls 1963 entwickelte Ivan Sutherland die grafische Benutzerschnittstelle Sketchpad [149, S. 5].

Den Aspekt, Computerressourcen besser zu verteilen, ging Anfang der 1960er-Jahre Fernando Corbató an. Er entwickelte das Time-Sharing, was in modifizierter Form auch im Apollo-Navigationscomputer wiederzufinden ist. Synergieeffekte zwischen den Hauptakteuren der Computerentwicklung in den 1960er-Jahren sind daher denkbar, worauf in Abschnitt 2.5 näher eingegangen wird.

Die vorliegende Arbeit untersucht im Besonderen den Apollo Guidance Computer. Da das Apollo-Programm jedoch keine lose Sammlung unzusammenhängender Einzelprojekte war, sondern ein umfangreiches Gesamtkonzept bildete, in dem unterschiedliche Computersysteme zusammen wirkten, wäre es wenig sinnvoll, den Apollo Guidance Computer isoliert zu betrachten. Außerdem sind – mögliche – Synergieeffekte innerhalb von Apollo sicher noch deutlicher sichtbar als zwischen Apollo und „Außer-Apollo-Projekten“.

Aus diesen Gründen werden daher zunächst auch einige der bedeutendsten Computersysteme innerhalb – und teilweise auch außerhalb – des Apollo-Programms kurz dargestellt, die nicht direkt der On-Board-Navigation im Apollo-Raumschiff zuzuordnen sind. Denn auch außerhalb des Apollo-Programms kamen, wie bereits erwähnt, zu Beginn der 1960er-Jahre neue Ideen und Techniken auf, die sich auf die Apollo-Navigationscomputer auswirkten.

Zuvor wird zum besseren Verständnis ein kurzer Überblick über die Geschichte der Computerentwicklung – wiederum mit Blick auf Apollo – gegeben, um die zeitliche Einordnung der o.g. Ären konkretisieren zu können bzw. zu zeigen, *wann* die Entwicklungen in diesen Ären ihre jeweiligen Höhepunkte hatten.

1.1 Themen

Dieser Abschnitt beschreibt die Themen, die im Rahmen dieser Arbeit untersucht werden und die hierfür anzuwendenden Methoden.

1.1.1 Thesen

1. Die Navigationscomputer des Apollo-Programms waren leistungsfähig genug um Menschen zum Mond und sicher wieder zurück zu navigieren.
2. Mehrere Entwicklungen auf dem Gebiet der Computertechnik in den 1960er-Jahren haben sich gegenseitig beeinflusst, wovon die Entwicklung des Apollo Guidance Computers profitierte.
3. Das Konzept des Apollo Guidance Computers hat die Entwicklung von Navigationssystemen, wie z.B. den Fly-By-Wire-Systemen, maßgeblich beeinflusst.
4. Die Entwicklung des Apollo Guidance Computers hat zu neuen Methoden zur Erstellung zuverlässiger Hard- und Software geführt.
5. Das Konzept der Benutzerschnittstelle des Apollo Guidance Computers hat spätere Benutzerschnittstellen, wie z.B. die Menüsteuerung, beeinflusst.
6. Die Zuverlässigkeit der in Apollo eingesetzten Technologien war höher als die der im Space Shutle Programm eingesetzten.

1.2 Methoden

Zur Untersuchung dieser Themen werden in erster Linie historische Dokumente der NASA und anderer Einrichtungen sowie einzelner Forscher herangezogen. Darüber hinaus werden Berichte und Auswertungen aus der Post-Apollo-Zeit betrachtet, sofern diese von Personen bzw. Institutionen stammen, die einen unmittelbaren Bezug zum Apollo-Programm aufweisen.

Außerdem wird an Hand von verfügbarem Quellcode zusammen mit den Informationen zur Hardware die Leistungsfähigkeit einzelner Programme untersucht, die in den bemannten Missionen des Apollo-Programms zum Einsatz gekommen sind.

Darüber hinaus werden spezielle Risiken für die On-Board-Computer des Apollo-Programms betrachtet, insbesondere die Strahlenbelastung aufgrund des Van-Allen-Strahlungsgürtels.

1.2.1 Historische Dokumente

Wesentliche historische Dokumente zur Untersuchung der vorgestellten Themen sind die Folgenden:

1. Eldon Hall
   - Integrated Circuits in the Apollo Guidance Computer, 1962
   - Computer Displays, 1962
   - A Case History of the AGC Integrated Logic Circuits, 1965
   - A Case History of the Apollo Guidance Computer, 1966
2. Hal Laning, Jr.
   - Design Principles for a General Control Computer, April 1960 (mit R. Alonso)
3. Fernando J. Corbató
   - An Experimental Time-Sharing System, 1962
4. Charles Draper
   - Brief an Robert Seamans, 21. November 1961
5. General Motors
   - Apollo Guidance and Navigation Lunar Module Student Study Guide, 1967
6. MIT Instrumentation Lab
   - Apollo Guidance, Navigation and Control, Guidance System Operations Plan - AS-278, 1966
7. NASA
   - Apollo 12 LM G&N Dictionary, SKB32100075-361, 1969
   - Mission Control Center Houston Familiarization Manual, 30. Juni 1967
   - The Manned Space Flight Network for Apollo, August 1968
   - Recent Advances in Display Media, 1968
8. Edward M. Copps Jr.
   - Recovery From Transient Failures of the Apollo Guidance Computer, 1968

1.2.2 Berichte

Außer diesen Dokumenten sind auch Ergebnisberichte und die Erfahrungsberichte einzelner Beteiligter von Bedeutung, hierbei insbesondere die folgenden:

1. Eldon Hall
   - Journey to the Moon: The History of the Apollo Guidance Computer, 1996
   - Reliability History of the Apollo Guidance Computer, 1972
   - The Apollo Guidance Computer, Rede am 10. Juni 1982, Computer Museum Boston
2. David Scott
   - The Apollo Guidance Computer - A Users View, Rede am 10. Juni 1982, Computer Museum Boston
3. Richard S. Johnston
   - Biomedical Results of Apollo, 1975
4. Jeffrey N. Wilkes und Richard A. Gustafson
   - Apollo Experience Report - A Use of Network Simulation Techniques in the Design of the Apollo Lunar Surface Experiments Package Support System, 2005

Zur Untersuchung der Technikgeschichte des Apollo-Programms werden zudem insbesondere folgende NASA-Publikationen herangezogen, die mit Ausnahme von „This New Ocean“ aus der Post-Apollo-Zeit sind:

- Roger E. Bilstein
  - Stages to Saturn: A Technological History of the Apollo/Saturn Launch Vehicle, 1980
- Courtney G. Brooks et Al.
  - Chariots for Apollo: A History of Manned Lunar Spacecraft, 1979
- Barton C. Hacker, James M. Grimwood
  - On the Shoulders of Titans: A History of Project Gemini, 1977
- Loyd S. Swenson, Jr. et Al.
  - This New Ocean: A History of Project Mercury, 1966
- James E. Tomayko
  - Computers in Spaceflight: The NASA Experience, 2005

Eine ausführliche Auflistung ist im Anhang, Abschnitt 17, zu finden

1.2.3 Analysen

Neben der Auswertung von Dokumenten und Berichten werden Analysen mit Hilfe von Methoden der Softwaremetrie durchgeführt, um die Leistungsfähigkeit des Apollo Guidance Computers nicht nur besser einschätzen zu können, sondern auch um einen Vergleich mit modernen Systemen zu ermöglichen.

- Strahlenbelastung und Strahlungsresistenz der On-Board-Computer.
- Codeanalyse der eingesetzten Software der On-Board-Computer.

1.2.4 Anfragen an Beteiligte

Darüber hinaus bietet sich auch die Möglichkeit, einzelne Beteiligte direkt zu fragen. Folgende Personen wurden angeschrieben und haben geantwortet:

- E-Mail-Anfrage an Fernando J. Corbató am 10. Januar 2016
- E-Mail-Anfrage an Margaret Hamilton am 10. Januar 2016

2 Überblick über die Geschichte der Computerentwicklung

Die Geschichte der Entwicklung des Computers ist ein derart umfangreiches Gebiet, dass es den Rahmen der vorliegenden Arbeit bei Weitem sprengen würde, hier eine auch nur annähernd vollständige Darstellung zu geben. Daher können in diesem Kapitel nur einige der für das hier behandelte Thema bedeutendsten Entwicklungen näher beschrieben werden. Der „Hauptcomputer“ des Apollo-Programms bestand aus fünf IBM-Mainframes vom Typ System 360 (siehe Abschnitt 6). Der Navigationscomputer der Saturn V (Abschnitt 8) stammte ebenfalls von IBM. Aufgrund der hohen Bedeutung dieses Unternehmens für das Apollo-Programm wird im Folgenden zunächst kurz die Geschichte von IBM dargestellt. Anschließend werden Entwicklungen auf den Gebieten der Betriebssystemtechnologie und der Benutzerschnittstellen der 1960er-Jahre beschrieben.

2.1 IBM

Bis heute ist IBM eines der weltweit führenden IT-Unternehmen; in den 1960er-Jahren war IBM das unangefochten größte Computerunternehmen der Welt. Die Ursprünge von IBM reichen jedoch zurück bis ins 19. Jahrhundert. Im Jahr 1884 überträgt Herman Hollerith die Lochkartentechnik, die zuvor bereits Verwendung bei Webstühlen fand, auf organisatorische Probleme. Eine bedeutende Neuerung dabei ist, dass sich mit der von Hollerith entwickelten Lochkarte abhängig von der jeweiligen Position der Löcher, unterschiedliche Merkmale codieren lassen [30]. Mit Hilfe dieser Lochkartentechnik entwickelte Hollerith in den 1880er-Jahren ein komplettes Datenerfassungssystem, das schon 1890 erfolgreich bei der amerikanischen Volkszählung eingesetzt eingesetzt wurde. Auf diesen Erfolg aufbauend gründete er 1896 die „Tabulating Machine Company“ (TMC) Aufgrund überzogener Preise blieb der wirtschaftliche Erfolg jedoch gering und im Jahr 1911 ging die TMC schließlich in die „Computing Tabulating Recording Company“ über. Das Unternehmen wurde schließlich 1924 umbenannt in „International Business Machines Corporation“ [69]. IBM baute in den Anfangsjahren vor allem Lochkartengeräte und Tabelliermaschinen. Erst 1952 kam der erste kommerzielle, wissenschaftliche Computer von IBM heraus, der 701 (36-Bit, 4K Wörter). 1959 erschien mit dem IBM 7090 eine Weiterentwicklung (36-Bit, 32K Wörter) des 701. Der 7090 kam auch im Apollo-Programm zum Einsatz; er wurde verwendet um die Schubdüsen für die Saturn-Raketen zu entwickeln [65]. Ebenfalls 1959 kam der IBM 1401 für geschäftliche Anwendungen mit zahlreichen Peripheriegeräten auf den Markt. Der IBM 1401 verfügte über eine variable Wortlänge und einen Speicher von 16K Wörtern.

In den 1960er-Jahren machte die Entwicklung der Computertechnik große Fortschritte, insbesondere durch die Entwicklung des integrierten Schaltkreises (Integrated Circuit - IC) im Jahr 1958 [75]. Trotz der technischen Fortschritte waren die meisten Computer der frühen 1960er noch auf die Anforderungen einzelner Kunden zugeschnitten.

Dies änderte sich schließlich mit der Einführung des IBM System 360, das am Dienstag, 7. April 1964 als Allzweckrechner von Thomas J. Watson Jr., Chairman von IBM, vorgestellt wurde [68].

2.1.1 IBM 360

Bei dem IBM System 360 handelte es sich um eine ganze Familie von Mainframes, die zueinander kompatibel waren. Darüber hinaus wurden zahlreiche Peripheriegeräte, wie etwa Magnetplattenspeicher oder Drucker, auf den Markt gebracht. Geräte des System 360 waren es dann, die die zentrale Großrecheneinheit des Apollo-Programms der NASA bildeten.

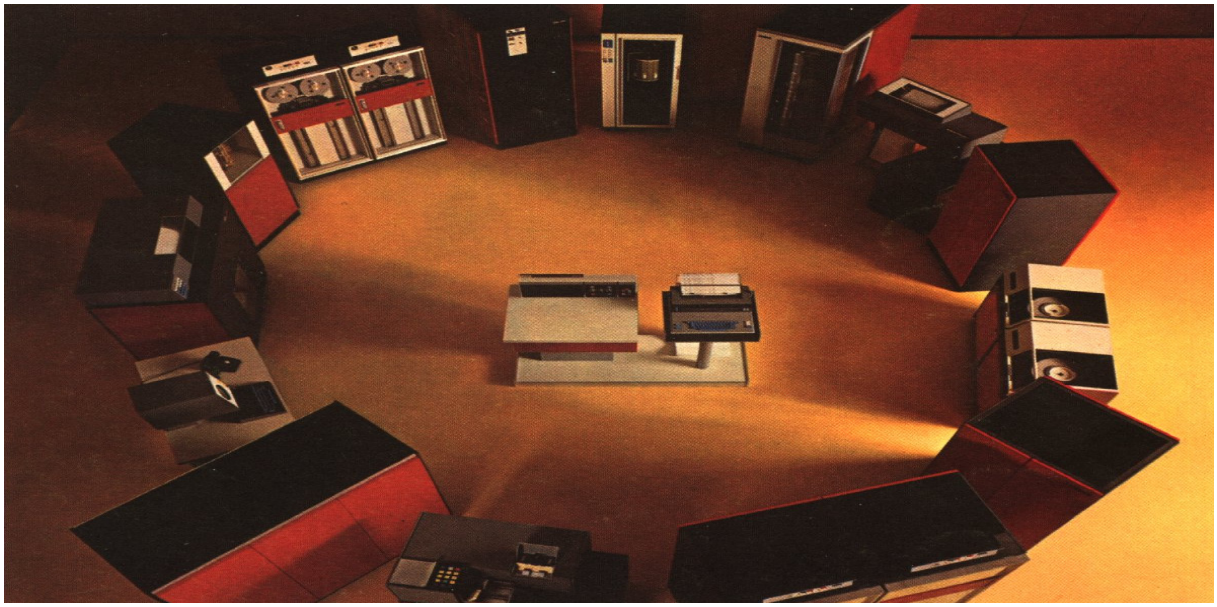


Abbildung 1. IBM System 360 [66]

Am Dienstag, dem 7. April 1964, stellte Thomas J. Watson Jr. das IBM System 360 als Allzweckrechner vor. Die Entwicklungskosten betrugen $ 5.000.000.000,- (damaliger Wert). Aufgrund dieser enormen Summe wurde die Entwicklung des IBM 360 auch „five billion Dollar gamble“ bezeichnet [64]; es hätte durchaus zum Ruin für IBM werden können. Die 360er-Reihe erhielt auch ein neues Betriebssystem, das OS/360, (auch ein Time-Sharing-System TSS/360 war geplant, lief jedoch unzuverlässig und erschien daher nur als Testversion).

Der IBM 360 sollte das gesamte Spektrum von Computeranwendungen abdecken, daher die Bezeichnung 360, von 360°. Er sollte gleichermaßen für wissenschaftliche, geschäftliche und sonstige Anwendungen geeignet sein.

Um diesen ganzen Bereich abzudecken, gab es eine ganze Familie von Computern des Typs IBM 360, die untereinander kompatibel waren. Diese Kompatibilität war eine der Neuerungen, die den 360er schließlich zu einem großen Erfolg werden lassen sollten.

Die Spannweite umfasste 14 ausgelieferte Modelle, einige besonders bedeutende davon waren:

Abbildung 2. Thomas J. Watson stellt den IBM 360 vor, IBM Niederlassung Poughkeepsie, NY [70]

- Modell 20: 4-32 KB, 2-3 kIPS (schwächstes Modell)
- Modell 75: 256-1024 KB, ca. 1 MIPS (eingesetzt von der NASA für den Real-Time Computer Complex, der die erdgebundene Navigation des Apollo-Programms übernahm)
- Modell 195: 1-8 MB, 10 MIPS (leistungsfähigstes Modell)

Zum ersten Mal war die Entwicklung eines neuen Systems auf Kompatibilität ausgelegt. Doch diese Kompatibilität bestand nicht nur innerhalb der 360er-Reihe, sondern auch zu älteren IBM-Mainframes, und zwar durch Virtualisierung. Darüber hinaus brachte IBM mit dem System 360 auch gleich 44 neue Peripheriegeräte auf den Markt, dabei handelte es sich z.B. um:

- Direct Access Storage Device (14″-Magnetplatten, 7,25 MB)
- 9-spurige Magnetbänder (ca. 43 MB pro Band)
- Drucker
- OCR-Scanner
- Grafisches Display IBM 2250

Doch nicht nur die Kompatibilität war eine Neuerung, die ab ca. 1980 die Verbreitung von Personal Computern stark vorantreiben würde, auch andere Merkmale des System 360 wurden später zu „Computerstandards“. So führte der IBM 360 z.B. die 8-Bit Datenstruktur „Byte“, sowie die Wortbreite von 32-Bit. Die elektronischen Schaltkreise des 360 bestanden aus 6 Prozessoren in Solid Logic Technology (SLT), IBMs neuer Aufbau- und Verbindungstechnologie. Zunächst wurden auf einer Kupferschicht mit Hilfe eines säurefesten Lacks im Siebdruckverfahren Bahnen aufgedruckt. Anschließend wurde die frei liegende Kupferschicht weggeätzt, so dass nur noch die Leiterbahnen übrig blieben und die Leiterplatte dann mit den Bauteilen bestückt werden konnte. Bei SLT wurden nun solche gedruckten Schaltungen zusammen mit aktiven Schaltungen zu Schaltkreisen zusammengefasst. SLT ist somit ein Hybridverfahren. Mehrere solcherart hergestellter Schaltkreise wurden schließlich auf einer Leiterplatte zusammengefasst. Mittels SLT hergestellte Schaltkreise waren dichter, schneller und benötigten weniger Energie als frühere Modelle.

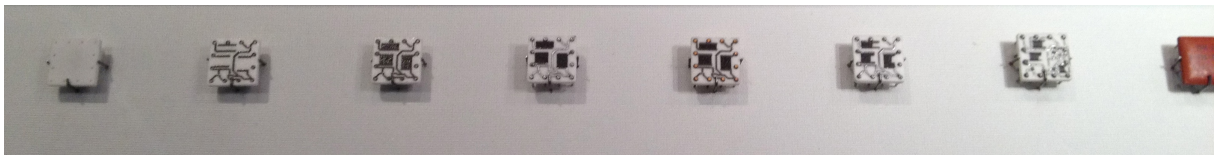


Abbildung 3. Herstellungsschritte von SLT-Schaltkreisen [145]

2.2 Betriebssysteme

Bis etwa Mitte der 1950er-Jahre verfügten Computer nur über sehr rudimentäre Betriebssysteme, z.B. über ein Kontrollprogramm, dass die Abfolge von Aufgaben regeln und den Speicher zwischen den einzelnen Aufgaben leeren konnte. Die Computer konnten nur jeweils eine Aufgabe gleichzeitig bearbeiten.

Anfang der 1960er-Jahre entwickelte Fernando J. Corbató dann ein Time-Sharing-System, dass es mehreren Nutzern erlaubte, gleichzeitig am selben Computer zu arbeiten. Der Computer verarbeitete dabei zwar weiterhin nur jeweils eine Aufgabe, teilte aber freie Rechenkapazität zwischen mehreren Benutzern auf [28]. Während ein Benutzer mit der Eingabe von Daten beschäftigt war, konnte der Rechner die Berechnungen für einen anderen Nutzer durchführen.

Dieses frühe Multi-User-Betriebssystem (CTSS - Compatible Time-Sharing System) wurde der Vorläufer von Multics und besaß bereits Aspekte, die wir auch bei heutigen Betriebssystemen wiederfinden, wie z.B. das Login oder auch der Informationsaustausch mit anderen Nutzern über einen gemeinsamen Bereich (common files).

Am 9. Mai 1963 wurde Corbató für die Sendereihe „MIT Science Reporter“ des Fernsehsenders WGBH-TV in Boston von dem Wissenschaftsreporter John Fitch interviewt. In dieser Folge, die den Titel „A Solution to Computer Bottlenecks“ trug, demonstrierte Corbató das System auf einem IBM 7090 und erklärte die grundlegenden Funktionsprinzipien [86]. In diesem Interview legte er die hohe Bedeutung der Mensch-Maschine-Kommunikation dar; er erklärte, dass das Bündeln von Aufgaben zu Aufgabenstapeln und die Übergabe dieses Stapels zur Abarbeitung an den Computer (Batch-Processing), diesen durchaus besser auslastet als jeweils nur eine Aufgabe einzugeben, zu berechnen und dann ausgeben zu lassen.

Abbildung 4. Fernando Corbató [86]

Das Batch-Verfahren führt jedoch dazu, dass Nutzer oft sehr lange warten müssen, bis ihre „Jobs“ an der Reihe sind. Solche Wartezeiten zu vermeiden und den Computer interaktiv nutzen zu können, dient CTSS. Zur Systembedienung dienten an den Computer angeschlossene Fernschreiber, die sowohl zur Ein- als auch zur Ausgabe verwendet wurden. Corbató beschreibt in diesem Interview auch die Verwendung von Prioritäten, jedoch auf Nutzerebene. nicht auf Prozessebene.

Er spricht auch bereits die Ergonomie an und macht einen auch noch aus heutiger Sicht interessanten Kommentar zum Umgang mit dem Fernschreiber als Ein/Ausgabegerät: „We have to study human engineering too, [..]“. Das komplette Interview ist in [86] zu finden.

CTSS und Multics waren bedeutende Meilensteine auf dem Weg zu modernen Multibenutzer- und Multitasking-Betriebssystemen. Am Multics-Projekt arbeiteten ab 1963 noch weitere Institutionen mit: General Electric und die Bell Labs. Zu den beteiligten Entwicklern in den Bell Labs gehörten Kenneth Thompson und Dennis Ritchie [147]. Das Multics-Projekt kam jedoch nur langsam voran, und die Bell-Labs zogen sich 1969 wieder vom Projekt zurück. Doch Thompson, Ritchie und ihre Kollegen erkannten die Vorteile, die die direktere Interaktion mit dem Computer bot und wollten die Multics-Idee weiterentwickeln. Zu Beginn der 1970er-Jahre entwickelten Thompson und Ritchie, aufbauend auf den Multics-Erfahrungen, dann auch Unix, ein Betriebssystem, das heute weiterhin in Gebrauch ist und selbst wiederum zur Grundlage von Linux wurde. Somit lässt sich eine klar ersichtliche Entwicklungslinie finden, die vom CTSS bis zum heutigen Linux führt. Linux ist heute ein weltweit verbreitetes und beliebtes Betriebssystem; auch die vorliegende Arbeit entstand an einem Linux-System.

IBM baute in den 1960er-Jahren ebenfalls auf den Konzepten von CTSS auf und Begann im Jahre 1964 mit der Entwicklung eines Betriebssystems zur Unterstützung virtueller Maschinen [6, S. 62]. Dieses System, kurz „VM“ für „Virtual Machine“, emulierte die Architektur eines System 360. 1972 kamen die Computer der Reihe IBM System 370 auf den Markt, den Nachfolgemodellen des System 360. Im System 370 wurde auch eine neue Version des VM eingesetzt, die neue Möglichkeiten, wie z.B. die virtuelle Speicherverwaltung, besaß [6, S. 63]. Das Betriebssystem der aktuellen zSeries von IBM, das z/VM, ist ein direkter Nachfolger der frühen VM-Systeme [6, Kap. 3].

2.3 Mensch-Maschine-Schnittstellen

Douglas Engelbart ist vor allem bekannt für die Maus, den Hypertext, Videokonferenzen, Textbearbeitung mittels „drag-and-drop“ und zahlreiche weitere Aspekte, die heute für die meisten Menschen beim alltäglichen Umgang mit dem Computer dazugehören. Weniger bekannt ist vermutlich, dass Engelbart in den 1950er-Jahren für das Ames Laboratory des NACA (National Advisory Committee for Aeronautics) arbeitete, das NACA war der direkte Vorläufer der NASA [36].

Er entwickelte die Maus zu Beginn der 1960er-Jahre am SRI (Stanford Research Institute) und wandte sich mit seiner Idee an die NASA, von der er auch Förderung für seine Forschung erhielt [37] [111]. Die NASA war insbesondere im Hinblick auf das bemannte Raumfahrtprogramm sehr interessiert an neuen Bedienkonzepten. Im Navigationscomputer des Apollo-Raumschiffs wurde nun zwar keine Maus eingesetzt, dennoch aber eine sehr innovative Methode der Mensch-Maschine-Kommunikation, eine wie sie sich auch Douglas Engelbart hätte ausdenken können: das DSKY (siehe Abschnitt 7.6). Hier soll keineswegs die These aufgestellt werden, Engelbart wäre in die Entwicklung des Apollo-Interfaces involviert gewesen, er forschte jedoch zu der gleichen Zeit, in der der Apollo-Navigationscomputer entwickelt wurde an neuen Benutzerschnittstellen und wäre damit sicher eine wertvolle Unterstützung für das Apollo-Programm gewesen. Da Engelbart 2013 gestorben ist, ist es leider auch nicht mehr möglich, ihn zu fragen, inwieweit er etwas von der Entwicklung des Apollo-Navigationscomputers mitbekommen hat.

Etwa zur gleichen Zeit zu der Douglas Engelbart an der Maus arbeitete, entwickelte Ivan Sutherland das „Sketchpad“, ein grafisches Benutzerinterface, das mit Lichtstift gesteuert wurde [154]. Das Sketchpad war ebenso wie die Maus ein bedeutender Schritt in Richtung moderner grafischer Benutzerschnittstellen und CAD-Anwendungen. Die 1960er-Jahre waren somit eine Zeit hoher Innovativität, was die Entwicklung neuartiger Möglichkeiten zur Bedienung von Computern angeht.

2.4 NASA

Auch die NASA selbst war bei der Entwicklung neuer Techniken an vorderster Steller dabei. Der erste *lineare* IC wurde z.B. 1964 entwickelt, die NASA hatte jedoch bereits 1963 eine eigene Version des linearen ICs [57]. Ebenso wurden bei der NASA neue Displaytechniken, wie z.B. Laser- und Plasma-Displays, erprobt und weiterentwickelt [118]. Die Plasma-Displays sind ein Beispiel dafür, dass es sehr lange dauern kann, bis eine neuartige Technologie weit genug für eine industrielle Massenfertigung entwickelt ist. Erst in in 1980er- und 1990er-Jahren verbreiteten sich Plasma-Displays weltweit. Doch bereits in den 1960er-Jahren forschte die NASA an solchen Displays und hatte erste Experimentalversionen.

2.5 Mögliche Verknüpfungen

Ivan Sutherland und Fernando J. Corbató haben zur gleichen Zeit am MIT gearbeitet wie die Entwickler des Apollo Guidance Computers. Abbildung 5 zeigt die Einordnung einiger dieser Entwicklungsschritte. Darin ist auch die Zeitspanne dargestellt, in der Gene Amdahl bei IBM an der Entwicklung des IBM 360 gearbeitet hat.

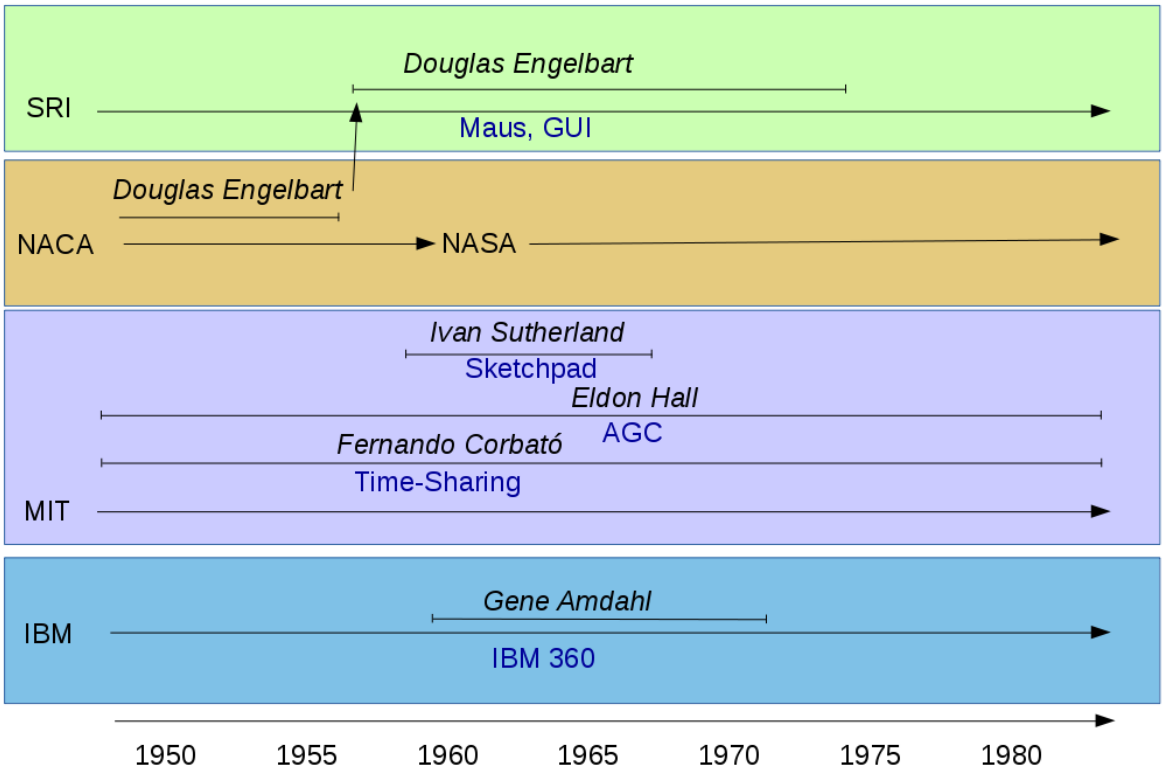


Abbildung 5. Timeline wichtiger Ereignisse in der Computerentwicklung, 1960er-Jahre

Der Apollo Guidance Computer wurde im MIT Instrumentation Laboratory entwickelt, welches vor Apollo bereits durch militärische Forschung bekannt worden war. Die Entwicklung des Navigationscomputers war damit also kein allgemeines MIT-Projekt. Das Apollo-Programm war von Anfang an sehr öffentlich ausgelegt, doch die Entwicklung in einem Institut für militärische Forschung lässt eher Geheimhaltung vermuten.

Wie lassen sich nun also mögliche Verbindungen ermitteln? Dazu kann man die vorgestellten Hauptakteure einfach befragen! Auf meine entsprechende Anfrage an Professor Corbató per E-Mail am 10.01.2016 antwortete dieser folgendermaßen:

> „No connection. Much of the Guidance Computer work was classified or restricted. [...] There was no interaction with the Instrumentation Lab work. .F.J Corbato.“ [27]

Das Interessante an Corbatós Antwort ist nicht etwa nur die Aussage, dass es keine Verbindungen gab, sondern die Begründung: Die Arbeit am Apollo Guidance Computer war geheim. Das ist etwas, was aus den öffentlich zugänglichen Unterlagen nur teilweise ersichtlich ist. Zwar tragen die NASA-eigenen Unterlagen oft noch den „classified“-Vermerk (der auch nach Deklassifizierung meist noch sichtbar ist), aber dies trifft nur selten auf MIT-Dokumente zu. Es sind natürlich auch nicht alle Dokumente in Zusammenhang mit der Entwicklung des Apollo Guidance Computers veröffentlicht worden. Vermutlich hat man zahlreiche Dokumente auch gar nicht aufbewahrt. Doch egal, wie viele der Dokumente heute noch verfügbar sind, Prof. Corbatós Aussage zeigt, dass die Entwicklung des Navigationscomputers für Apollo keineswegs stets so öffentlich ablief wie es heute erscheinen mag.

3 Die Anfänge des Apollo-Programms

US-Präsident Eisenhower hatte Ende Juli 1955 die Entwicklung eines amerikanischen Erdsatelliten in Auftrag gegeben, worauf die UDSSR dann vier Tage später, am 1. August 1955, eine ähnliche Entwicklung ankündigte. Dabei hatte es sich keineswegs um leere Propaganda gehandelt:

Am 4. Oktober 1957 (UTC) startete in Baikonur eine Trägerrakete des Typs R-7 mit dem ersten künstlichen Satelliten der Menschheit, dem Sputnik [110].

Die Folge war in der westlichen Welt der „Sputnikschock“. Man folgerte: Wenn die UDSSR einen Satelliten in eine Erdumlaufbahn bringen kann, dann möglicherweise auch Atombomben; zumindest aber wurde klar, dass sich die USA nun in Reichweite sowjetischer Raketen befanden.

Der Sputnikschock führte unter anderem zur Gründung der NASA und zum bemannten amerikanischen Raumfahrtprogramm. Nach dem Erfolg der Sowjetunion mit dem Sputnik 1 folgte bereits einen Monat später, am 3. November 1957, der Sputnik 2. Sputnik 2 war jedoch kein gewöhnlicher Satellit, er hatte einen lebenden Passagier an Bord: Die Hündin Laika [11, S. 18]. Laika kam zwar nicht lebend zurück zur Erde, doch ihr Flug führte zu neuen Erkenntnissen über die Überlebensmöglichkeiten eines Weltraumfluges.

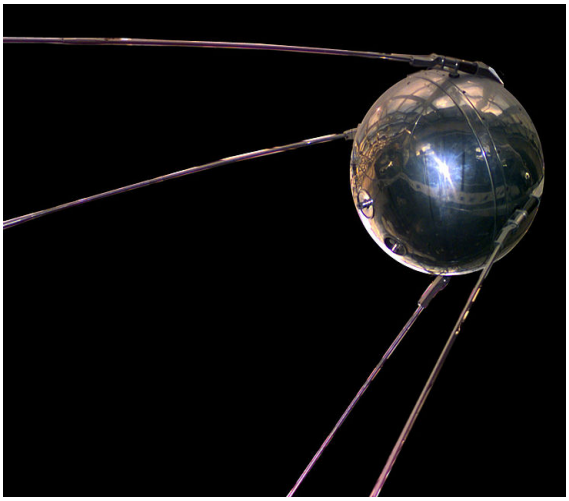


Abbildung 6. Nachbildung von Sputnik 1, [110]

Die USA konterten mit eigenen Entwicklungen: Am 1. Februar 1958 startete mit Explorer 1 der erste künstliche Satellit der USA erfolgreich ins All [120]. Somit konnte der Vorsprung der Sowjetunion in der Satellitentechnik relativ schnell aufgeholt werden, doch die Ambitionen der USA gingen über künstliche Satelliten hinaus, das nächste große Ziel war es, einen Menschen in den Weltraum zu bringen.

Im Herbst 1958 begann das Projekt Mercury, das eben dieses Ziel verfolgte: Einen Menschen in den Weltraum und sicher wieder zurück zu bringen.

3.1 Das Mercury-Programm

Zunächst waren die ersten „Testpiloten“ des Mercury-Projekts jedoch, ebenso wie in der UDSSR, Tiere: Nach Tests mit kleineren Säugetieren wie z.B. Mäusen startete am 31. Januar 1961 der Schimpanse Ham zu einem 16,5-minütigen Flug mit einer Redstone-Rakete und erreichte eine Höhe von ca. 252 km [156, S. 313ff]. Dies war höher als ursprünglich vorgesehen, da zum einen der Flugwinkel steiler als geplant war, und zum anderen das Capsule Escape System zu früh von der Raumkapsel getrennt wurde.

Durch diese frühzeitige Trennung erhielt die Kapsel mehr Schub als geplant (siehe bzgl. des Escape-Systems auch das „Launch Escape System“ im Abschnitt 8.1).

Doch trotz dieser Abweichungen vom geplanten Verlauf der Mission führte Ham die ihm antrainierten Arbeiten (das Verstellen von Hebeln) durch und kehrte wohlbehalten zur Erde zurück [156, S. 310ff].

Am Mittwoch, dem 12. April 1961 war es schließlich soweit, dass auch ein Mensch in den Weltraum flog und gesund wieder auf der Erde landete. Doch handelte es sich nicht um einen Amerikaner, sondern um den Sowjetrussen Juri Gagarin, der mit der Vostok 1 einen 108-minütigen Orbitalflug durchführte [115]. Wiederum handelte es sich also um einen sowjetischen Erfolg (zum Aspekt des Wettlaufs ins Weltall siehe auch Abschnitt 7).

Abbildung 7. Mercury-Astronaut Ham, 31. Januar 1961 [91]

Abbildung 8. Mercury-Astronaut Alan Shepard, 5. Mai 1961 [102]

Doch bereits am 5. Mai 1961, einem Freitag, flog dann auch ein US-Amerikaner in den Weltraum: Alan Shepard. Shepards Flug dauerte nur 15 Minuten und 28 Sekunden [125], bzw. 15 Minuten und 22 Sekunden gemäß dem Werk „This New Ocean“ [156, S. 341]. Diese Diskrepanz ist vermutlich darauf zurückzuführen, dass in [125] nur Shepards Flug allgemein dargestellt ist, in [156] jedoch explizit die „in flight“-Zeit genannt wird.

Die USA hatten nun also ihre ersten Erfahrungen mit der bemannten Raumfahrt gemacht, und die Zuversicht war groß, dass man noch wesentlich weiter mit dem Raumfahrtprogramm gehen könne.

3.1.1 John F. Kennedy erklärt die Mondlandung zur nationalen Aufgabe

Weniger als drei Wochen nach Shepards Flug erklärte Präsident John F. Kennedy die Mondlandung zu einer nationalen Aufgabe, der sich die USA stellen sollten. Am 25. Mai 1961 sagte er vor dem Congress:

> I believe this nation should commit itself, to achieving the goal, before this decade is out, of landing a man on the Moon and returning him safely to the earth. No single space project in this period will be more impressive to mankind, or more important in the long-range exploration of space; and none will be so difficult or expensive to accomplish. [123]

Abbildung 9. Dr. Werner Von Braun erklärt Präsident John F. Kennedy das Saturn-System, 16. November 1963 [117].

Abbildung 10. John F. Kennedy am 12. September 1962 in Houston, Texas [130]

Im darauffolgenden Jahr, am Mittwoch, dem 12. September 1962, hielt er im Rice-Stadion in Houston, Texas, eine begeisternde Rede, in der er den festen Entschluss zu einer bemannten Mondlandung noch im gleichen Jahrzehnt bekräftigte. Aus dieser Rede stammt sein berühmtes Zitat:

> We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and which we intend to win, and the others, too. [130]

Das Ziel stand fest, doch der Weg dahin noch nicht. Zunächst mussten weitere Erfahrungen mit der bemannten Raumfahrt gesammelt werden. Dazu diente das weiterhin laufende Projekt Mercury. Für die Flüge zum Mond war das Apollo-Programm vorgesehen. Apollo war als Nachfolgeprojekt von Mercury angesetzt und erhielt bereits 1960 seinen Namen von Abraham Silverstein, Direktor des NASA Space Flight Programms in den späten 1950er-Jahren [114]. Zu Beginn war Apollo auch nicht ausschließlich auf eine bemannte Mondlandung ausgelegt, auch bemannte Mondumkreisungen gehörten zu den potentiellen Zielen [40]. Spätestens nach Kennedys Rede vor dem Congress (s. o.) war jedoch die bemannte Mondlandung das Ziel des Apollo-Programms.

Um zum Mond und wieder zurück zu gelangen, mussten große Lasten in den Weltraum befördert werden: Treibstoff, Sauerstoff, Wasser, technische Systeme, etc. – dafür wiederum waren leistungsstarke Trägerraketen nötig. Auch musste geklärt werden, ob der Mond im Direktflug erreicht werden konnte, oder ob man andere Möglichkeiten nutzen sollte. Wernher von Braun bevorzugte zwar zunächst die Direktmethode, war aber offen für andere Vorschläge und Ideen. Für einen Direktflug, bei dem auch das gesamte Mondlandefahrzeug wieder vom Mond startete, wären noch größere Lasten in den Weltraum zu bringen gewesen als es selbst die später gebaute Saturn V (siehe Abschnitt 8.1) ermöglicht hätte. Es wurde bald klar, dass die Lasten verringert werden mussten. Was dazu führte, das Raumfahrzeug modular zu bauen: Im Mondorbit würde sich das Raumschiff teilen und nur eine Landefähre auf dem Mond landen. Von dieser Mondlandefähre würde wiederum nur ein Teil, die Aufstiegsstufe, zum Mutterraumschiff im Mondorbit zurückkehren. Die Teile, die also nicht mehr benötigt wurden, würden vor Ort zurückgelassen. Eine solch komplexe Vorgehensweise bedingte aber Rendezvous- und Andockmanöver im Mondorbit. Um solche Manöver sowie weitere nötige Techniken für eine bemannte Mondlandung beherrschen zu können und um die Zeit bis zu den ersten Flügen im Rahmen des Apollo-Programms zu überbrücken, wurde das Projekt Gemini gestartet, in dem zahlreiche spätere Apollo-Astronauten tätig waren.

3.2 Das Gemini-Programm

Im Gemini-Programm waren erstmals zwei Astronauten gemeinsam im Weltall, darauf und auch auf den Aspekt der Andockmanöver, also der Verbindung zweier Raumfahrzeuge, weist auch der Name Gemini hin. Erforscht wurden neben den bereits angesprochenen Rendezvous- und Andockmanövern auch EVAs (Extra-Vehicular Activity - Außerbord-Aktivitäten), längere Aufenthalte im Weltall, neue Raumanzüge sowie zahlreiche weitere Aspekte, wie z.B. die Versorgung der Besatzung mit Lebensmitteln und Wasser, neue Navigationssysteme und vieles mehr. Die Gemini-Raumkapsel war auch das erste Raumfahrzeug der Amerikaner mit einem On-Board-Computer. Die Mission Gemini VII war die längste der Gemini-Missionen und dauerte zwei Wochen. Damit wurde erprobt, welche Auswirkungen ein längerer Weltraumflug auf Mensch und Maschine haben würde. Gemini wurde bis 1966 weitergeführt; die letzte Mission war Gemini XII vom 11. bis zum 15. November 1966 [49, 380ff]. Zu dieser Zeit fanden bereits die ersten Flüge des Apollo Programms statt.

3.3 Das Apollo-Programm

Bereits von Oktober 1960 bis Mai 1961 wurden Machbarkeitsstudien für das Mercury-Nachfolgeprojekt Apollo durchgeführt. Dazu hatte die NASA im Juli und August 1960 mehrere Konferenzen veranstaltet, bei denen Verträge für sechsmonatige Machbarkeitsstudien zu einer Mondumkreisung angeboten wurden [19, S. 15]. Die Zuschläge

für diese Studien erhielten schließlich Convair/Astronautics, General Electric und die Martin Company [19, S. 16-17]. Die Studien waren im Mai 1961 beendet und die drei beteiligten Unternehmen stellten ihre Designs für mögliche Raumfahrzeuge vor. Nur eine Woche nach Ende der Machbarkeitsstudien hielt Kennedy dann seine Rede vor dem Congress; das Apollo-Programm bekam dadurch nun eine enorme Bedeutung. Doch die NASA war zu dieser Zeit noch nicht wirklich auf ein solches Mammut-Unternehmen vorbereitet. Neue Einrichtungen, qualifiziertes Personal und industrielle Vertragspartner wurden benötigt. Im weiteren Verlauf des Jahres wurden Komitees und Arbeitsgruppen gegründet, neues Personal eingestellt und die Hauptbereiche des Programms festgelegt, in denen externe Vertragspartner benötigt würden. Der erste Vertrag ging, wie bereits in der Einleitung erwähnt, an das MIT Instrumentation Laboratory und betraf das Navigationssystem [19, S. 38ff].

Nachdem auch die weiteren Verträge, wie z.B. die für Mondlandefähre, Kommandokapsel und andere Teilaspekte des Apollo-Programms an die einzelnen Institutionen und Unternehmen vergeben waren, gingen die eigentlichen Entwicklungsarbeiten los. Die ersten Zeitpläne für die unbemannten und bemannten Flüge von Kommandomodul, Servicemodul und Mondlandefähre waren jedoch zu optimistisch und sahen bereits Mitte der 1960er-Jahre bemannte Flüge vor [53, S. 56].

Da zahlreiche neue Systeme entwickelt und gebaut werden mussten, die vorher nicht dagewesene Herausforderungen darstellten, mussten die Zeitpläne jedoch überarbeitet werden. Die ersten Flüge des Apollo-Programms starteten daher erst 1966, die bemannten Flüge sogar erst 1968. In den letzten Jahren der 1960er-Jahre nahm das Apollo-Programm jedoch an Fahrt auf und noch 1968 erfolgte mit Apollo 8 der erste bemannte Flug um den Mond. 1969 folgten dann mit Apollo 11 und 12 zwei bemannte Landungen auf dem Mond. Somit wurde Kennedys Plan, noch vor Ende des Jahrzehnts einen Menschen auf den Mond und sicher wieder zurück zu bringen, gleich doppelt erreicht; noch 1969 betraten vier Amerikaner die Mondoberfläche.

Nachdem dieses Ziel erreicht war, wurden stärker wissenschaftlich orientierte Missionen in den Vordergrund gerückt. Die erste Mission in den 1970er-Jahren, Apollo 13, sollte den Krater Fra Mauro erforschen; die Mission musste jedoch aufgrund der Explosion eines Sauerstofftanks vorzeitig abgebrochen werden [98]. Die Astronauten konnten jedoch gesund zur Erde zurückkehren [98]. Apollo 14 holte schließlich die Erkundung des Kraters Fra Mauro erfolgreich nach.

Durch Leistungsverbesserung der Triebwerke Saturn V wurde es auch möglich, in den späteren Missionen größere Lasten zum Mond zu befördern. Dies wurde in den letzten drei Apollo-Missionen, Apollo 15, 16 und 17 genutzt. Jeweils ein Lunar Roving Vehicle wurde mit auf den Mond befördert und man konnte größere Mengen Mondgestein zurück zur Erde zu bringen [11, S. 376ff].

Am 7. Dezember 1972 schließlich startete mit Apollo 17 der letzte bemannte Flug zum Mond innerhalb des Apollo Programms. Bei diesem Flug war erstmals ein Geologe, Harrison Schmitt, dabei. Dies zeigte wiederum die zunehmend wissenschaftlich ausgeprägte Natur des Apollo-Programms. Apollo 17 landete am 19. Dezember 1972 mit mehr als 110 kg Mondgestein wieder auf der Erde, damit kam die Apollo-Ära zu einem erfolgreichen Ende.

Es mag bedauerlich erscheinen, dass gerade in einer solchen Phase der wissenschaftlichen Erforschung des Mondes das Apollo-Programm zu seinem Ende kam; jedoch ist zu bedenken, dass ohne die politische Komponente und ohne den technologischen Wettlauf zwischen den USA und der Sowjetunion das Apollo-Programm möglicherweise überhaupt nicht zustande gekommen, zumindest aber bei weitem nicht mit der Geschwindigkeit verwirklicht worden wäre, mit der es schließlich verwirklicht worden ist, denn dann hätte es ja für beide Nationen keinen politisch zwingenden Grund für solch ein technologisch umfangreiches und anspruchsvolles Unternehmen gegeben.

4 Navigationsberechnungen

Der Aufgabenbereich, den das Navigationssystem des Apollo-Raumschiffs abdeckt, wurde Guidance & Navigation, kurz G&C, genannt. Da in den weiteren Ausführungen wiederholt von Guidance und Navigation die Rede ist, ist es wichtig, diese Begriffe klar zu definieren. Charles Stark Draper, der Leiter der Entwicklung des Apollo-Navigationssystems (siehe Abschnitt 7.1), hat dies in der Publikation „Space Navigation Guidance and Control“ von 1965 getan. Seine Definitionen sind somit insbesondere in Bezug auf Apollo gut anwendbar. Guidance wird dort folgendermaßen definiert:

> „Guidance is the process of collecting all pertinent available data and generating the maneuver commands necessary for Mission accomplishment“ [5, S. I-26]

Und in Bezug auf Navigation:

> „Navigation is the process of collecting all pertinent available data, generating information on position and motion, and indicating this information for the purposes of display and recording“ [5, S. I-26]

Bei Guidance geht es also um die Steuerung, bei Navigation in erster Linie um Anzeige und Aufzeichnung der Kurs- und Bewegungsdaten. Beide Definitionen ähneln sich jedoch, daher heißt es auf der gleichen Seite weiter:

> „Because the operations required are generally similar for guidance and for navigation the two functions are carried out by the same system – called the guidance and navigation system“ [5, S. I-26]

Der zusammenfassende Begriff Guidance & Navigation wird somit aufgrund der Ähnlichkeit der durchzuführenden Arbeitsvorgänge verwendet. Eine solche Zusammenfassung ist auch schon deshalb sinnvoll, weil dadurch ähnliche Aufgaben gebündelt angegangen werden können. Das G&C-System besteht dabei aus zahlreichen Komponenten wie Gyroskopen, Steuerdüsen und Computern.

Um das Apollo-Raumschiff auf dem richtigen Kurs zu halten, wurde ein Gyroskop-basiertes Navigationssystem eingesetzt, die Inertial Measurement Unit, kurz: „IMU“. Doch Gyroskope driften mit zunehmender Laufzeit von ihrer Ausrichtung ab, daher wurden Methoden benötigt, diese Gyroskop-Abdrift zu korrigieren. Dazu dienten feste Markierungen, wie z.B. Sterne und Planeten, deren Positionen in den On-Board-Computern abgespeichert waren. Anhand dieser Markierungen und unter Zuhilfenahme trigonometrischer Berechnungen ließ sich nun feststellen, an welcher Position relativ zur Erde bzw. zum Mond sich das Raumfahrzeug gerade befand. Die Gyroskope der IMU liefen dadurch nicht anders oder genauer, hatten nun aber wieder einen korrigierten Bezugspunkt und konnten damit wiederum weiter zur Navigation eingesetzt werden. Im Anhang, Abschnitt 16.1, wird eine Liste von Sternen bzw. Bezugspunkten wiedergegeben, wie sie in den Apollo-Missionen zum Einsatz gekommen ist, die Liste der Apollo 12-Mission.

Für die sich ergebenden Berechnungen zur Standortbestimmung wurden Computer benötigt, die zum einen klein genug waren, um in die Kapsel zu passen, zum anderen leicht genug bedienbar waren, um schnell und einfach von den Astronauten benutzt werden zu können. Zu diesem Zweck wurde der Apollo Guidance Computer, kurz AGC, entwickelt. Dieser war sehr robust gebaut und verfügte mit dem „Display and Keyboard“ (DSKY) über eine für die damalige Zeit revolutionäre Ein/Ausgabeeinheit (siehe auch Abschnitt 7.6).

Die Aufgabe, die Gyroskop-Abdrift zu korrigieren, gehörte noch zu den einfacheren Aufgaben des AGC. Schwieriger war, wie leicht nachvollziehbar sein dürfte, die Landung auf dem Mond, für die der AGC ebenfalls mehrere Programme besaß. Auch die Phase des Wiedereintritts in die Erdatmosphäre ist durch zahlreiche Berechnungen und Steueraufgaben durch den AGC gekennzeichnet. Dementsprechend umfangreich ist auch das entsprechende Programmpaket, das in mehrere Einzelprogramme unterteilt war: Nach erfolgreicher Abarbeitung eines Teilprogramms wurde automatisch das jeweils nächste gestartet.

Da die Berechnungen für die Navigation eines Raumfahrzeugs sehr komplex werden können, wurden in den Bodenstationen zusätzlich Großrechnerkomplexe eingesetzt, deren Hauptkomponente war der „Real-Time Computer

Complex“ (RTCC), bestehend aus fünf IBM 360-Mainframes. Zusätzlich zu diesen setzte die NASA zahlreiche weitere Rechenanlagen ein, wie z.B. UNIVAC-Mainframes zur Verarbeitung der Funksignale. Bei den Computern des Apollo-Programms handelt es sich also weder um *eine einzelne* Entwicklung, noch um eine einheitliche Linie von Computern. Die NASA setzte sowohl eigene Entwicklungen (bzw. im eigenen Auftrag entwickelte Systeme) als auch am Markt verfügbare Industriegroßrechner ein. Auf solchen Großrechnern wurden auch die grundlegenden Flugrouten berechnet. Dass dies sehr kompliziert werden kann, ist leicht ersichtlich, wenn man sich vor Augen hält, dass Erde und Mond ja ein sehr dynamisches System bilden: Sowohl Erde als auch Mond drehen sich umeinander und zusätzlich um sich selbst. Allein um den Mond von der Erde aus überhaupt treffen zu können, muss das Raumschiff in einem bestimmten Zeitfenster (abhängig von der Erddrehung) starten und auf einen Punkt zielen, an dem der Mond sich zu diesem Zeitpunkt noch gar nicht befindet, sondern erst dann dort sein wird, wenn sich auch das Raumschiff in der entsprechenden Entfernung zur Erde befindet. Um nun wiederum Zeit und Entfernung synchronisieren zu können, muss die Geschwindigkeit des Raumschiffs überprüft und durch Schubmanöver korrigiert werden. Diese Schubmanöver verringern nun aber wiederum die Masse des Raumschiffs, was sich dann wieder auf das Beschleunigungsvermögen auswirkt. Bereits an diesen wenigen Beispielen sieht man, dass die Navigation zum Mond und zurück sehr komplex werden kann; die Unterstützung durch Großrechner ist daher sehr nützlich für die Navigation, um verschiedene Flugrouten und Möglichkeiten für verschiedene Zeitfenster durchzurechnen, so dass deren jeweilige Vor- und Nachteile abgewogen werden können, lange bevor die eigentliche Mission startet.

Wenn im Folgenden von „Apollo-Computern“ allgemein die Rede ist, so ist damit dementsprechend die gesamte Vielfalt der Rechenanlagen gemeint, die von der NASA im Rahmen des Apollo-Programms eingesetzt worden ist.

5 Die Computer des Apollo-Programms - Übersicht

Sowohl die Planung als auch die Durchführung der Mondmissionen wurde durch zahlreiche Computersysteme nicht nur unterstützt, sondern auch wesentlich getragen: Ein Unternehmen, wie die Landung eines Menschen auf dem Mond und dessen sichere Rückkehr zur Erde, war noch nie zuvor durchgeführt worden. Die Anforderungen an das Navigationssystem konnten dementsprechend schwer beurteilt werden und daher war der erforderliche Aufwand für einen Navigationscomputer schwer einzuschätzen. Doch selbst wenn solche Anforderungen konkret festgestanden hätten, wie hätte man solch ein System während der Entwicklung testen sollen? Dabei halfen Computersimulationen weiter. Die Computer, auf denen solche Simulationen laufen konnten, mussten natürlich wesentlich leistungsfähiger sein als der eigentliche Navigationscomputer, da ja auch die äußeren Faktoren (Gravitation, Treibstoffverbrauch, etc.) simuliert werden mussten.

Bei dem umfangreichen Einsatz von Computern im Apollo-Programm erhebt sich die interessante Frage, ob die Mondlandung auch ohne Computer möglich gewesen wäre. Um dies zu beantworten ist ein Blick auf eine Apollo-Mission hilfreich, bei der Computer eine besondere Rolle gespielt haben: Apollo 13.

Während des Fluges zum Mond explodierte ein Sauerstofftank, so dass die Mission abgebrochen werden musste. Während der Rückkehr von Apollo 13 musste Jim Lovell den Kurs des Raumschiffs per Handsteuerung korrigieren. Dies gelang ihm auch, insofern wäre es theoretisch denkbar, dass man auch „per Hand“ fliegen kann. Doch zum einen war dies enorm schwierig für ihn, zum anderen hatte er die Informationen über Kursabweichung von Mission Control erhalten, die den Flug mittels Radioteleskopen überwachten. Dem ersten Einwand könnte man entgegensetzen, dass hier mehr Training im Umgang mit der Handsteuerung hätte helfen können. Der zweite Punkt lässt sich jedoch nicht so leicht beseitigen: Die Arbeit in Mission Control beanspruchte nicht nur viele Menschen sondern auch Computer, die die Flugrouten berechnen mussten. Eine manuelle Berechnung hätte für die zeitkritische Rettungsaktion von Apollo 13 viel zu lange gedauert. Im Raumschiff musste Strom gespart werden, daher wurde auch das Navigationssystem heruntergefahren. Auch die einzelnen Möglichkeiten zum Stromsparen und zum Neustarten des Command Modules wurden mit Computerhilfe simuliert. Der Flug von Apollo 13 wäre daher ohne Computerunterstützung höchstwahrscheinlich nicht so gut ausgegangen.

Was das Apollo-Programm insgesamt angeht, so lässt sich feststellen, dass es von vornherein auf Computerunterstützung ausgelegt war. Ein Mondfahrtprogramm ohne Computer wäre sicher denkbar, hätte aber von Beginn an

völlig anders aufgebaut sein müssen als es Apollo war. Wie ein solches alternatives Mondflugprogramm aussehen könnte, ist allerdings sehr spekulativ, da die technische Entwicklung einerseits fortschrittlich genug sein müsste, um leistungsstarke Raketen und Navigationsmöglichkeiten zu erlauben, andererseits jedoch keine Computer vorhanden bzw. genutzt werden sollten. Eine rein optische Navigation per Teleskop und Sextant, sowie ein Inertialnavigationssystem mit rein mechanischen Ein/Ausgabeeinheiten wäre denkbar, jedoch ohne zusätzliche Absicherung sehr riskant. Die Systeme müssten also redundant ausgelegt sein, was wiederum mehr Platzbedarf bedeutet. Dafür würde aber der Computer eingespart werden. Die Landung auf dem Mond und der Start zurück zur Erde müssten ebenfalls entweder komplett anders ausgelegt sein (z.B. in Form der zu Beginn des Apollo-Programms in Betracht gezogenen Landung des gesamten Apollo-Raumschiffs auf dem Mond) oder die Berechnungen für die nötigen Manöver hätten komplett im Voraus erfolgen müssen. Dies hätte dann natürlich weniger Toleranz gegenüber Abweichungen geboten als wiederholte Neuberechnungen während der Missionen.

5.1 Planung

Nicht nur an der Durchführung der Mondflüge waren zahlreiche Computersysteme beteiligt, auch die Planung und Berechnung der Flugbahnen erfolgte mit Computerunterstützung. In diesem Abschnitt erfolgt ein Überblick über den Computereinsatz während dieser Planungsphasen der Mondmissionen.

Mittels Großrechnern war es möglich, verschiedene Flugbahnen mit ihren jeweiligen Vor- und Nachteilen durchzurechnen, bevor eine Entscheidung zugunsten einer konkreten Flugbahn getroffen wurde. Wie sinnvoll hier die Rechnerunterstützung ist, lässt sich bereits an der scheinbar simplen Fragestellung erkennen, ob das Raumfahrzeug in einer freien Rückkehrbahn (free return trajectory) zum Mond fliegen soll oder auf einer anderen Flugbahn. Eine freie Rückkehrbahn bietet einerseits mehr Sicherheit für die Astronauten, schränkt aber andererseits auch die möglichen Landeplätze auf dem Mond ein. Mittels Computersimulationen können unterschiedliche Szenarien getestet werden und deren jeweilige Vor- und Nachteile ermittelt werden. Die Berechnungen sind natürlich auch von Hand möglich, aber die wesentlich höhere Rechengeschwindigkeit eines Computers erlaubt es, in der gleichen Zeiteinheit wesentlich mehr Flugbahnen zu untersuchen als es ohne Computerunterstützung möglich wäre.

Die Planung einer spezifischen Mission musste etwa ein Jahr vor dem jeweiligen Start beginnen [105]. Ein so langer Zeitraum war nötig, da eine Mission aus sehr vielen Einzelaspekten bestand. Das Raumschiff musste zum richtigen Zeitpunkt unter bestimmten Wetterbedingungen starten, der Einschuss des Raumfahrzeugs aus dem Erdorbit in Richtung Mond (TLI) musste zum richtigen Zeitpunkt im korrekten Winkel erfolgen, etc. Schließlich musste das Raumschiff auch wieder im richtigen Winkel in die Erdatmosphäre eintreten und die Fallschirme mussten sich zur festgesetzten Zeit in festgelegter Reihenfolge öffnen. Diese Liste ließe sich beliebig fortsetzen, z.B. mit der Landung auf dem Mond, dem Start von dort oder möglichen Missionsabbrüchen. Ohne Computerunterstützung wären solch umfangreiche Vorbereitungen und die Simulationen der unterschiedlichen Missionsaspekte höchstwahrscheinlich nicht durchführbar gewesen.

Für die Computersimulationen kam hauptsächlich der Real-Time Computer Complex zum Einsatz (siehe Abschnitt 6). Der RTCC war aber nicht das einzige System hierfür, auch andere Großrechnersysteme wurden verwendet, so z.B. der Analog-Digital-Computer-Complex, ein Hybridsystem [105].

5.2 Durchführung

Zur Durchführung der Missionen des Apollo-Programms waren drei Computersysteme von zentraler Bedeutung:

- Real-Time Computer Complex (RTCC)
- Launch Vehicle Digital Computer (LVDC)
- Apollo Guidance Computer (AGC)

Die fünf IBM 360 Mainframes, aus denen der RTCC bestand, waren Großrechner, wie sie in den 1960er-Jahren auch industriell genutzt wurden. Der LVDC und der AGC waren jedoch völlige Neuentwicklungen. Die Anforderungen an diese Rechner waren sehr hoch, schließlich sollten mit ihrer Hilfe Menschen zum Mond und sicher

wieder zurück gelangen können. Vom AGC, dem Hauptcomputer im Apollo-Raumschiff, wurde dementsprechend folgendes erwartet [162]:

- Eigenständiges Navigieren von der Erde zum Mond
- Kontinuierliche Berechnungen der aktuellen Zustandsvektoren
- Berechnung von Navigationsfixpunkten mit Hilfe von Sonne, Sternen und Planeten
- Remote-Updates von der Bodenstation aus
- Manuelle Übernahme der Steuerung der Saturn V in Notfällen
- Fluglagensteuerung mittels digitalem Autopiloten
- Landung auf dem Mond
- Aufstieg vom Mond
- Durchführung des Rendezvousmanövers mit der Kommandokapsel
- Echtzeit-Informationsanzeige
- Einfach bedienbare Benutzerschnittstellen
- Multiprogrammfähigkeit

6 Real-Time Computer Complex

Im Mission Control Center in Houston, Texas, befand sich der Real-Time Computer Complex (RTCC), der „Hauptcomputer“ der Apollo- und Gemini-Missionen. Die Entscheidung, einen Zentralrechner zu verwenden wurde getroffen, um mögliche Kommunikations- und Datenübertragungsprobleme zu vermeiden, die bei verteilten Anlagen auftreten könnten. Die Mission Gemini IV im Jahre 1965 war die erste Mission, bei der der RTCC zum Live-Einsatz kam. Zu dieser Zeit bestand der Zentralrechner aus fünf IBM 7094 Rechnern mit jeweils 64K Speicher bei einer Wortgröße von 36 Bit.

Nach dem Feuer von Apollo 1 (27. Januar 1967) wurde der RTCC umgebaut, die IBM 7094 wurden durch die neuen IBM 360 Großrechner ersetzt. Diese Geräte verfügten über 1 MB Hauptspeicher mit einer Zugriffszeit von 750 ns sowie über weitere 4 MB Large Core Storage, einem Erweiterungsspeicher mit einer etwas geringeren Zugriffszeit von 3,6 µs. Die Bytegröße betrug 8 Bit (anstelle der zu dieser Zeit üblichen 4 oder 6 Bit), daher lässt sich die Angabe der Hauptspeichergröße auch leicht in MB angeben ohne Umrechnungen vornehmen zu müssen; die Wortbreite betrug 32 Bit. Der Rechner beherrschte Fließkommaberechnungen. Zur Datensicherung wurden 9-spurige Magnetbänder eingesetzt, acht der Spuren nahmen die Bits jeweils eines Bytes auf, der Inhalt der neunte Spur war ein zu diesem Byte gehörendes Prüfbit.

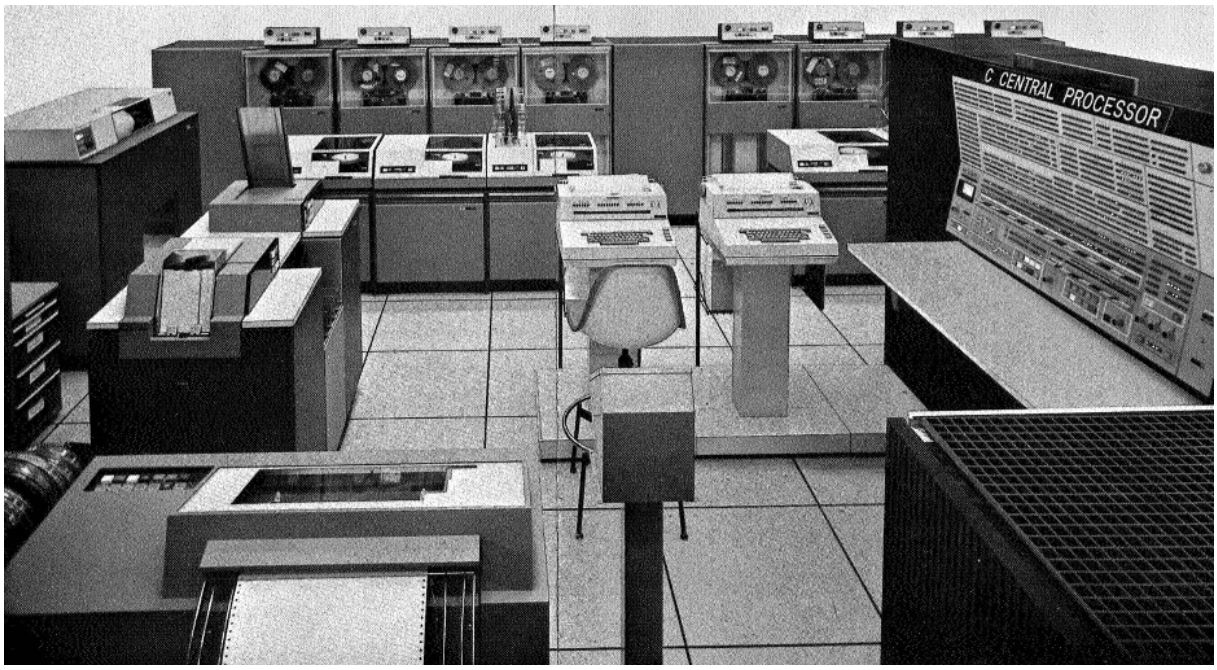


Abbildung 11. Ausschnitt des RTCC im Manned Spacecraft Center, Houston [157]

Die Aufgaben der einzelnen Maschinen des RTCC waren folgendermaßen aufgeteilt: Zwei Geräte wurden für die Softwareentwicklung eingesetzt, auf dem dritten Gerät liefen Flugsteuerungssimulationen, ein weiteres Gerät stand als Backup ständig in Bereitschaft und das fünfte Gerät schließlich war der „Mission Operation Computer“ (MOC), der sämtliche Berechnungen, die während eines Raumflugs nötig waren, durchführte. Zur Steuerung dieser Rollenverteilung gab es die „System Selector Unit“, mit der jeder einzelne der fünf Mainframes in einen beliebigen der verfügbaren Operationsmodi geschaltet werden konnte. Diese Operationsmodi waren: „Mission Operations“, „Dynamic Standby“, „Checkout and Training“, und „Off-Line“ [109, Sec. 2-3-1].
Die Berechnungen des RTCC umfassten den gesamten Missionsablauf:

Die Daten, die zur Steuerung jeder einzelnen Phase der Mission nötig waren, wurden gesammelt, verarbeitet und dann aufbereitet an Mission Control weitergeleitet [109, Sec. 2-3ff, Sec. 3-4ff].

Zur Überwachung der Flugparameter des Raumschiffs wurden Soll-Ist-Vergleiche durchgeführt, der RTCC berechnete die aktuellen Daten über Position, Lage und Geschwindigkeit des Raumschiffs und verglich die Ergebnisse mit den vorher festgelegten Soll-Werten. Diese Berechnungen erfolgten in Echtzeit, man musste nicht auf die Berechnungsergebnisse warten (dementsprechend daher auch der Name Real-Time Computer Complex).

Zu den Aufgaben des RTCC gehörte auch die Berechnung von Zündungszeitpunkten und Zündungslängen für das TLI-Manöver. Dieses sowie weitere Manöver waren vor den jeweiligen Missionen simuliert worden. In diesen Simulationen konnten zwar Werte vorberechnet werden, doch um auch auf Abweichungen in Echtzeit reagieren zu können, wurde der RTCC für die Berechnung der aktuellen TLI-Werte eingesetzt. Diese Möglichkeit war bei den späteren Apollo-Missionen, die auf keiner reinen Free-Return-Trajectory flogen sondern einem hybriden Ansatz folgten, noch weitaus wichtiger. Dabei wurden weiterhin möglichst viele Vorteile der Free-Return-Trajectory genutzt, ohne deren Einschränkungen zu sehr zu unterliegen: Bei einer Free-Return-Trajectory sind die möglichen Landegebiete auf dem Mond eingeschränkt, und zwar auf ein Gebiet um den Mondäquator. Demgegenüber ermöglicht der hybride Ansatz ein größeres Spektrum möglicher Landeplätze. Auch wurden aufgrund der hybriden Flugbahn Ressourcen des Raumschiffs geschont, so dass mehr Nutzlast mitgeführt werden konnte. Das Raumschiff verließ die Free-Return-Trajectory erst nach Trennung von der S-IVB (3. Stufe der Saturn V) und konnte auch danach noch mit Hilfe eines zusätzlichen Triebwerkssatzes im Lunar Module zur Erde zurückkehren, falls die Mission an dieser Stelle doch noch hätte abgebrochen werden müssen. Die Simulationen lieferten jedoch für die hierfür nötigen komplexen, dynamischen Berechnungen keine exakten Ergebnisse mehr, ein Einsatz des RTCC während der Missionen war daher für die hybriden Missionen äußerst wichtig [107].

Die eingesetzten Programme waren überaus komplex, so hatte z.B. das Programm zur Überwachung des Raumschiffs und der medizinischen Daten der Astronauten eine Größe von 6 MB und galt als die bis zu diesem Zeitpunkt komplexeste geschriebene Software. Auch die Datensicherung wurde ernst genommen, sie erfolgte alle 90 Minuten auf Magnetband. Zur Bedienung des RTCC waren Terminals mit Bildschirmen vorhanden, die mit Hilfe von Konvertern angeschlossen waren (Digital-to-Television Converter). So wie es einen Standby-Rechner im RTCC gab, gab es auch eine Hilfs-Display-Ausstattung (Auxiliary Display Equipment), die ausschließlich für den Standby-Computer vorgesehen war [109, Sec. 2-2-1].

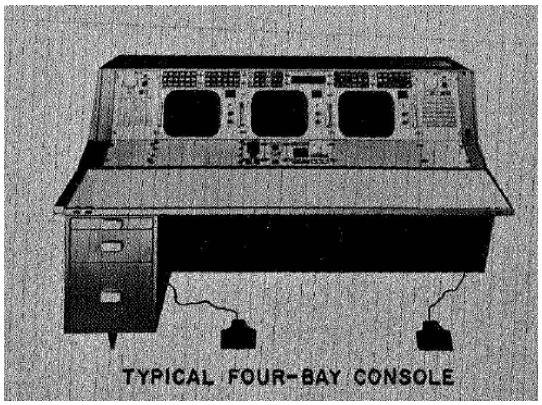


Abbildung 12. RTCC-Konsole [109, S. 2-2-1]

Abbildung 13. Mission Control Center, Apollo 7, [79]

Zum Abruf von Daten gab es an den Konsolen die „Computer Request Keyboards“. Mit diesen ließen sich während einer Mission sowie im Training spezifische Daten abrufen, die dann auf den Terminals, auf Plottern oder auch auf großen Projektionsdisplays dargestellt wurden [109, Sec. 3-3-1]. Diese Terminals sind es auch, die häufig in Dokumentarfilmen, aber auch in fiktiven Filmen mit NASA-Bezug zu sehen sind.

7 Apollo Guidance Computer

Der Apollo Guidance Computer (AGC) war der On-Board-Computer der Apollo-Raumschiffe. Der AGC konnte sowohl Daten von Mission Control empfangen als auch selbst Daten senden. Darüber hinaus ermöglichte er aber auch eine von Fernübertragungen unabhängige Navigation des Raumschiffs. Sowohl im Command Module als auch im Lunar Module war jeweils ein AGC vorhanden. Die beiden Geräte waren von der Hardware her identisch, es lief aber unterschiedliche Software auf ihnen. Im Command Module wurde der Computer CGC (für „Command Module Guidance Computer“), im Lunar Module entsprechend LGC (für „Lunar Module Guidance Computer“) genannt.

Die Hauptaufgaben des AGC waren die folgenden:

- Das Sammeln und Verarbeiten von Fluginformationen in Echtzeit
- Die automatische Navigation des Apollo-Raumfahrzeugs

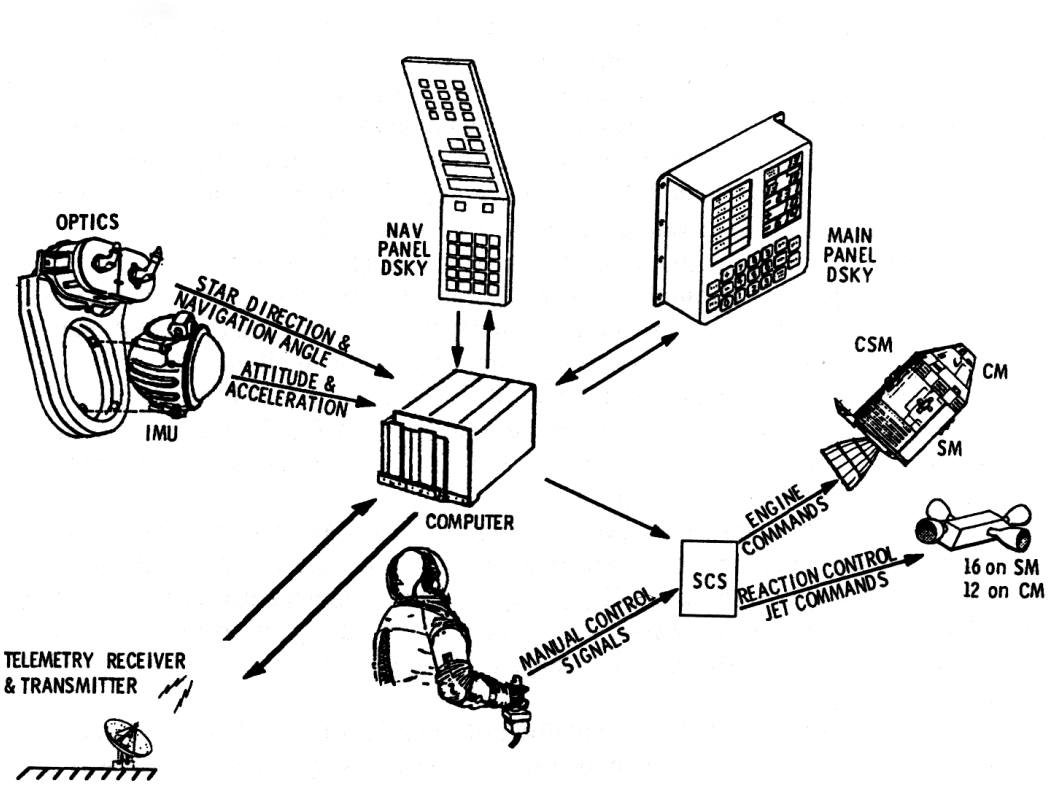


Abbildung 14. Systemschnittstellen AGC Block I [53, S. 66]

Zu Beginn der Entwicklung des AGC wurde das Ziel einer vollständig automatischen Navigation aufgestellt. Diese sollte sicherstellen, dass die Astronauten im Falle von Bewusstlosigkeit oder anderer Handlungsunfähigkeit dennoch sicher zur Erde zurückkehren könnten.

Doch das war nicht der einzige Grund, warum eine vollständige Automatisierung angestrebt wurde: Die USA befand sich in den 1960er-Jahren schließlich in einem Wettlauf mit der Sowjetunion. Ein bemanntes Mondlandeprogramm der Sowjetunion wurde zwar erst nach Auflösung der UDSSR öffentlich zugegeben (siehe z.B. den – undatierten – Artikel von Marcus Lindroos [77]), doch hatte das sowjetische bemannte Raumfahrtprogramm in den

1960ern bereits zahlreiche Pionierleistungen erreicht, wie z.B. den ersten Menschen im Weltall, Juri Gagarin (12. April 1961) [115], und den ersten Weltraumspaziergang durch Alexey Leonov am 18. März 1965 [140, S. 1-2]. Die USA holten diese Vorsprünge jeweils kurze Zeit später auf. Der erste US-Amerikaner im Weltall war Alan Shepard, der am 5. Mai 1961 mit der „Freedom 7“ startete, also nur 23 Tage nach Gagarins Flug [125]. Und am 3. Juni 1965, 77 Tage nach Leonovs Mission, führte Edward White den ersten Weltraumspaziergang der Amerikaner durch [140, S. 2-4]. Die USA und die Sowjetunion befanden sich also in der Tat in einem Wettlauf ins Weltall und die Gefahr einer möglichen Störung durch die jeweilige Gegenseite war nicht auszuschließen.

Darüber hinaus erreichte der Kalte Krieg zwischen den USA und der Sowjetunion in den 1960er-Jahren einen nie zuvor dagewesenen Höhepunkt: Die Kubakrise im Oktober 1962. Der Status der Verteidigungsbereitschaft der USA (DEFCON - Defense Condition) ist in fünf Stufen unterteilt, von DEFCON 5 (Lowest state of readiness) bis DEFCON 1 (Nuclear war) [135]. Im Verlauf der Kubakrise wurde die Verteidigungsbereitschaft der USA zum ersten und bisher einzigen Mal auf DEFCON 2, also der direkten Vorstufe zum Nuklearkrieg, angehoben [44, S. 61]. Die Kubakrise wurde dann in beiderseitigem Einvernehmen beendet, doch blieben weiterhin Spannungen zwischen den Supermächten, wovon Stellvertreterkriege wie z.B. der Vietnamkrieg ein deutliches Zeugnis ablegen. Und zu einer solchen Zeit strebte nun die USA mit enormen Aufwand eine bemannte Mondlandung an. Die Gefahr einer sowjetischen Intervention in das Apollo-Programm war daher durchaus realistisch. Eine vollständig autonome Navigation des Apollo-Raumfahrzeugs war daher auch im Hinblick auf eine mögliche Störung der Funkverbindung durch die Sowjetunion anzustreben. Eldon Hall schreibt hierzu:

> „The quest for autonomy resulted, at least in part, from international politics in the 1950s and 1960s, specifically the cold war between the Soviet Union and the United States. NASA assumed that autonomy would prevent Soviet interference with U.S. space missions.“ [53, S. 59]

Das hochgesteckte Ziel einer vollständigen Automatisierung wurde später zwar wieder fallengelassen, doch der AGC erhielt dennoch die Fähigkeit zur autonomen Navigation. Denn obwohl ein großer Teil der Navigationsberechnungen am Boden durchgeführt wurde, gab es Zeitabschnitte, in denen kein Funkkontakt zur Erde bestand (z. B. während sich das Apollo-Raumschiff auf der erdabgewandten Seite des Mondes befand). Außerdem war es auch schon aus Gründen der Redundanz vernünftig, die Berechnungen der erdgebundenen Computer im Raumschiff verifizieren zu können. Dadurch ließ sich bei bestehender Funkverbindung auch sehr gut einschätzen, wie präzise die Berechnungen des AGC im Vergleich zu den Berechnungen des RTCC waren.

Und gab es nun Interventionsversuche durch die Sowjetunion? Diese Frage ist nur schwer zu beantworten, doch vermutlich gab es zumindest keine gravierenden Störaktionen. Denn hätte es sie gegeben, hätte die USA vermutlich deutlich darauf hingewiesen, dass das Apollo-Programm trotz sowjetischer Störaktionen erfolgreich war.

Wie begann nun die Entwicklung des Apollo Guidance Computers? Im folgenden Abschnitt werden einige der Personen vorgestellt, die für die Entwicklung des AGC von entscheidender Bedeutung waren.

7.1 Die Entwicklung des Apollo Guidance Computers

Dr. Charles Stark Draper vom Massachusetts Institute of Technology (MIT) hat in den 1950er-Jahren ein Gyroskop-basiertes Navigationssystem entwickelt, das „inertial guidance system“. In einem Testflug im Jahre 1953 hat dieses Navigationssystem ein Flugzeug ohne Bezugnahme auf äußere Landmarken sicher von Boston nach Los Angeles geleitet. Charles Draper war bei diesem Flug mit an Bord, da er stets ein großes Vertrauen in seine Entwicklungen hatte. Durch dieses Navigationssystem wurde Draper bekannt als „Vater der Inertialnavigation“ [134]. Daher war es naheliegend für die NASA, Charles Draper für ein so ambitioniertes Projekt wie Apollo an Bord zu nehmen. Er erhielt den Auftrag, das Leitsystem für die Apollo Missionen zu entwickeln, dies war der erste Auftrag der NASA im Rahmen des Apollo-Programms, unterzeichnet am 10. August 1961 [19, Kapitel 2].

Charles Draper konnte dabei auf seine Erfahrungen mit dem „inertial guidance system“ zurückgreifen. Auch bei der Arbeit am Navigationscomputer für das Apollo-Programm zeigte sich wieder sein Vertrauen in die eigene Arbeit und in die seines Teams.

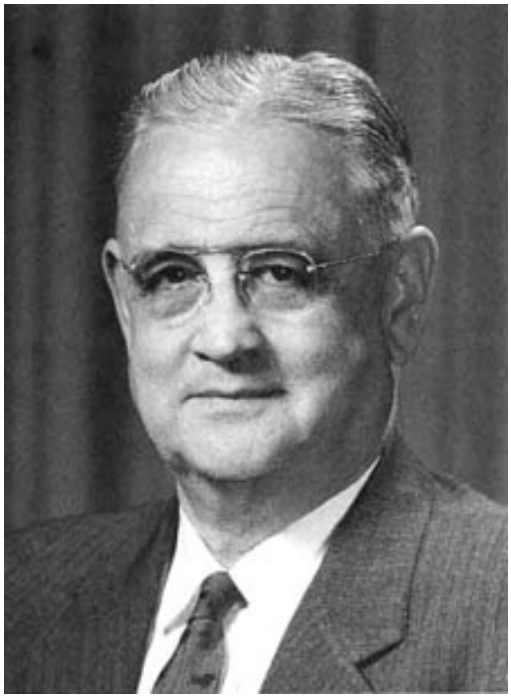


Abbildung 15. Charles Stark Draper (1901 - 1987) [161]

Zu Beginn des Projektes im Jahre 1961 gab es bei der NASA-Führung wiederholt Zweifel daran, dass das MIT ein Navigationssystem entwickeln könne, das Menschen zum Mond und zurück bringen kann. Hugh L. Dryden, Deputy Administrator der NASA, beorderte Draper zu dem verantwortlichen NASA-Administrator James E. Webb, um vor diesem Stellung zu nehmen. Draper betonte nicht nur seine Überzeugung und sein Vertrauen in die Arbeit seines Teams und in die Technik, nein, er ging noch einen Schritt weiter: Am 21. November 1961 schrieb er an den Deputy Administrator Robert Seamans eine Bewerbung als Astronaut im Apollo-Programm. Darin versicherte er nicht nur sein Vertrauen in seine Mitarbeiter am MIT, sondern zeigte mit dem Brief auch, dass er als Apollo-Astronaut sein Leben dem Navigationscomputer anvertrauen würde, den das MIT entwickelte [39].

Charles Drapers Team bestand natürlich aus zahlreichen Mitarbeitern, die hier nicht alle vorgestellt werden können. Dennoch sollen zumindest einige der Personen, die für bestimmte Aspekte der Entwicklung des AGC von besonderer Bedeutung waren, etwas näher betrachtet werden:

Ein weiterer Wissenschaftler von besonderer Bedeutung für die Entwicklung des AGC in Drapers Team war Hal Laning:

Die Computer der 1960er Jahre arbeiteten mit gleich verteilten Zeitfenstern (Time-Slices), um mehrere Aufgaben (quasi-) parallel zu bearbeiten. Die einzelnen Aufgaben (Tasks) erhielten dabei jeweils einen gleichgroßen Anteil an der Rechenkapazität. Das funktionierte zwar, konnte jedoch dazu führen, dass eine einzelne Aufgabe, die nicht mehr reagierte, das ganze System blockierte. Für kritische Anwendungen war dieses Verfahren bei Weitem nicht sicher genug. Bei einem bemannten Flug zum Mond könnte ein solcher „Systemabsturz“ die Astronauten in unmittelbare Lebensgefahr bringen. Eine bessere Lösung als die Aufteilung in gleichgroße Zeitscheiben musste her.

Hal Laning hatte nun die Idee, Prioritäten für die einzelnen Aufgaben einzuführen. Im Apollo Guidance Computer setzte er diese Idee um: Kritische Aufgaben wurden bei Engpässen in der Rechenkapazität vorrangig behandelt, weniger kritische Aufgaben wurden zurückgestellt [35].

Abbildung 16. J. Halcombe „Hal“ Laning, Jr. (1920 - 2012) [133]

Die einzelnen Prozesse überprüften dabei ihre jeweiligen Prioritäten selbst (kooperatives Multitasking). Prioritätenbasiertes Scheduling wurde später zum Bestandteil zahlreicher Betriebssysteme, wie z.B. Linux, MS-Windows (in den Varianten bis einschließlich Windows 3x, also in den 16-Bit-Versionen, wurde kooperatives Multitasking für das Scheduling verwendet [71], [85]) und Solaris [151]. Das im AGC eingesetzte Scheduling-Verfahren war jedoch kein reines kooperatives Multitasking, sondern eine Mischung aus kooperativem und preemptivem Multitasking (siehe auch Abschnitt 7.7); somit wurden hier bereits Techniken eingesetzt, die noch über das Scheduling-Verfahren von Systemen wie Windows 3x hinausgingen und bereits preemptive Komponenten beinhalteten.

Die hohe Bedeutung von Lanings prioritätenbasiertem Scheduling zeigte sich bei der Landung von Apollo 11 auf dem Mond, was später noch ausführlicher behandelt werden wird (Abschnitt 12.1).

Hal Laning hatte zu Beginn seiner Arbeit am Apollo-Programm bereits umfangreiche Erfahrungen mit der Entwicklung von Softwaresystemen; bereits 1952 entwickelte er einen Compiler für den MIT Whirlwind, der die Eingaben von Gleichungen in den Computer gegenüber der Programmierung in Assembler vereinfachte [74]. In den 1950er-Jahren arbeitete er mit Richard Battin zusammen. Battin wurde im Rahmen des Apollo-Programms „Director of Mission Development“ des AGC und war damit verantwortlich für das Softwaredesign des Navigationssystems.

Leiter der Hardwareentwicklung des AGC war Eldon C. Hall (Division Director of Digital Computer Development). Er überzeugte die NASA im Jahr 1962 davon, Integrierte Schaltkreise (Integrated Circuit - IC) für den Navigationscomputer zu verwenden [32]. Der IC wurde, wie bereits dargelegt, erst 1958 entwickelt. Dieser IC bestand jedoch noch aus zwei Bipolartransistoren (Transistoren, bei denen sowohl negative als auch positive Ladungsträger zum Ladungstransport Verwendung finden), die auf einen Träger aus Germanium aufgebracht und durch Golddrähte verbunden waren, was zu hohen Produktionskosten führte. 1959 entwickelte Robert Noyce, Mitbegründer der Firma Fairchild Semiconductor, den ersten aus einem einzelkristallinen Stück gefertigten IC. Robert Noyce entwickelte jedoch nicht nur diesen IC, sondern auch das fotografische Fertigungsverfahren zur Herstellung.

Bereits wenige Jahre später wurde die NASA durch die Entwicklung des AGC der erste Großabnehmer dieser noch jungen Technologie und förderte dadurch auch die Weiterentwicklung von IC-Produktionsstätten. Eine solche Förderung war auch dringend nötig, denn Hall schreibt in seinem Buch „Journey to the Moon“, dass die IC-Technologie zunächst keine besondere Begeisterung bei der Industrie hervorrief, da für die industrielle Produktion sowohl Leistungsmerkmale als auch Kosten wichtiger als die Miniaturisierung mittels ICs waren [53, S. 17].

Wenn man bedenkt, dass in der Zeit, da die Entscheidung zugunsten der ICs fiel, diese erst seit etwa einem Jahr auf dem Markt verfügbar waren, scheint es eine riskante Entscheidung gewesen zu sein, sich bei einem so bedeutenden Projekt wie das Apollo-Programm auf eine derart neue Technologie einzulassen.

Abbildung 17. Eldon C Hall [52]

Doch es war nicht nur der Wunsch nach Miniaturisierung, der zu dieser Entscheidung führten, sondern auch die Zuverlässigkeit: Zwar gab es noch keine längerfristigen Erfahrungen mit ICs, aber bereits die Tatsache, dass hierbei weniger Bauteile zur Erzielung eines bestimmten Schaltkreises benötigt wurden als bei der Verwendung einzelner Transistoren, bedeutete auch, dass weniger Fehlerpotential bestand.

Außerdem war bereits zu Beginn der 1960er-Jahre eine Verringerung der Preise für ICs in den kommenden Jahren absehbar, da die Preise bereits zu sinken begonnen hatten. In einem Brief von Charles Draper an Charles W. Frick, Manager des Apollo Spacecraft Project Office (ASPO), legte Draper die Vor- und Nachteile der IC-Technologie für den Apollo Guidance Computer dar. In diesem Brief stellte Draper schon im November 1962 fest, dass die Verwendung von ICs bereits innerhalb eines Jahres zu einer kostengünstigeren Produktion führen könnte als unter Verwendung separater Komponenten, da der Bau des Computers mit ICs einfacher wäre als ohne [38]. Darüber hinaus machte Draper auch gleich den Vorschlag, ICs in den Geräten der Bodenunterstützung einzusetzen; dies würde natürlich den Absatz an ICs weiter erhöhen und dadurch schließlich auch einen Einfluss auf die Preisentwicklung haben.

Die IC-Technologie führte auch dazu, dass sich die Designarbeit für neue Systeme von den Computerherstellern zu den Chipherstellern hin verschob, was wiederum Ängste bei den Computerherstellern auslöste, da diese befürchteten, ihre Arbeit würde nicht mehr benötigt [53, S. 20]. Heute, im 21. Jahrhundert, wird die Designarbeit nun auch hauptsächlich von den Prozessorherstellern, wie z.B. Intel oder AMD, durchgeführt. Designer werden also immer noch gebraucht, auch wenn sich das Arbeitsgebiet verschoben hat. Der Computer ist als PC heute ein Alltagsgegenstand, der meist nur noch „zusammengeschraubt“ wird. Designarbeit spielt dabei sicher nur eine untergeordnete Rolle. Aber dennoch werden auch heute noch neue Computer entworfen, man denke nur an die unterschiedlichen Mainframes, wie sie in der Industrie eingesetzt werden, oder auch an Hochleistungsrechner wie diejenigen von Seymour Cray; bei diesen ist die Designarbeit ein wesentlicher Faktor auf dem Weg zum

fertigen Rechner. Darüber hinaus werden auch heute, im 21. Jahrhundert, zahlreiche neue miniaturisierte Computer entworfen, die in Form von Handys oder Armbanduhren auf den Markt kommen. Die Befürchtungen der Computerdesigner in den 1960er-Jahren waren also größtenteils unbegründet.

Wie waren nun die eingesetzten ICs bestückt? Wie aus der Booleschen Logik bekannt ist, lassen sich sämtliche logischen Funktionen aus NOR-Gattern zusammenstellen. Diesen Umstand machten sich die Entwickler des AGC zunutze: Die gesamte Logik des AGC basiert auf NOR-Gattern.

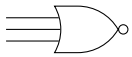


Abbildung 18. NOR-Gatter mit drei Eingängen (ANSI-Symbol)

Der AGC Block I, die auf unbemannten Testflügen eingesetzte Version des AGC, enthielt 4100 ICs, jedes davon bestückt mit einem Drei-Eingangs-NOR-Gatter. Der Block II, die Version, die in den bemannten Raumflügen eingesetzt wurde, enthielt zwar „nur“ 2800 ICs, dafür enthielt jedes davon aber zwei NOR-Gatter mit drei Eingängen. Der Block I war jedoch keineswegs die erste Version des AGC, vor diesem gab es bereits mehrere Entwicklerversionen, wie folgende Tabelle zeigt [53, Kap. 3-6] [13]:

Version	Jahr	Kommentar
Mod 1A	1959	Als „Christmas-Computer“ bezeichnet, da er bis Weihnachten 1959 fertig sein sollte. Der Assembler dafür wurde deshalb auch „YUL-Sytem“ genannt [13]
Mod 1B	1961	Aus Mars-Mission-Study
Mod 3C	1962	Erster AGC-Prototyp (AGC3)
AGC4	1963	Sollte Prototyp für Flugversion werden, wurde jedoch ein weiterer Versuchsaufbau [53, S. 87]
AGC4B	1963/64	Prototyp für Block I [53, S. 118]
AGC5	Mai 1964	Weiterentwicklung
AGC6	August 1964	Erster Block I AGC von Raytheon
AGC Block II	1966	Die in den bemannten Flügen eingesetzte Version

Tabelle 1. AGC-Entwicklerversionen

Übersicht IC-Bestückung:

- Block I (4,100 ICs,)
- Block II (2,800 ICs, jedes davon mit zwei Drei-Eingangs-NOR-Gattern)

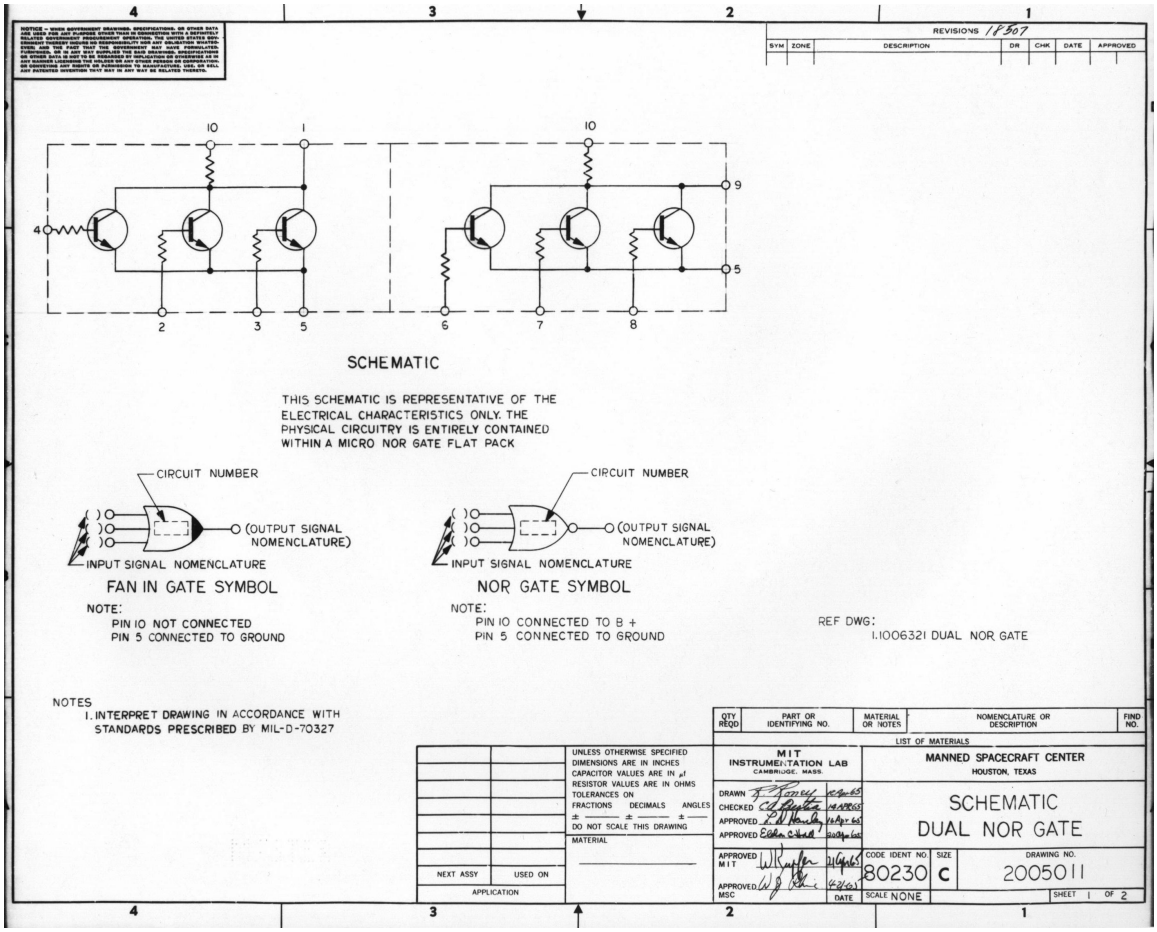


Abbildung 19. Schema der doppelten NOR-Gatter des AGC (1965) [50]

Eine weitere Entscheidung, die möglichst früh in der Designphase getroffen werden musste, war die der Wortlänge: Durch größere Wortlängen könnten größere (bzw. genauere) Dateneinheiten in einem Arbeitsschritt verarbeitet werden sowie größere Mengen an Speicherplätzen direkt angesprochen werden. Doch auch die Hardware würde mit wachsender Wortgröße umfangreicher werden. Da bei der Entwicklung des AGC die Größe ein entscheidender Faktor war, fiel die Entscheidung zugunsten eines relativ kleinen beschreibbaren Speichers (RAM), aber eines umfangreicheren Read-only Memory (ROM) und einer relativ geringen Wortlänge. Die gewählte Wortlänge betrug schließlich 16 Bit (14 Datenbit, 1 Vorzeichenbit, 1 Prüfbit). Genauere Berechnungsergebnisse, wie sie für Navigationsberechnungen nötig waren, konnten durch Double-Precision-Berechnungen durchgeführt werden [53, S. 69]. Als Größen für den Speicher sahen die ursprünglichen Planungen nur 120 Wörter für den RAM und 4000 Wörter ROM vor [53, S. 70]. Dies erwies sich später als zu klein dimensioniert; die Entwickler hatten zunächst zu sehr die Miniaturisierung in den Vordergrund gestellt, anstatt sich einen umfassenden Überblick über die Anforderungen zu verschaffen, wie Hall es beschreibt [53, S. 69].

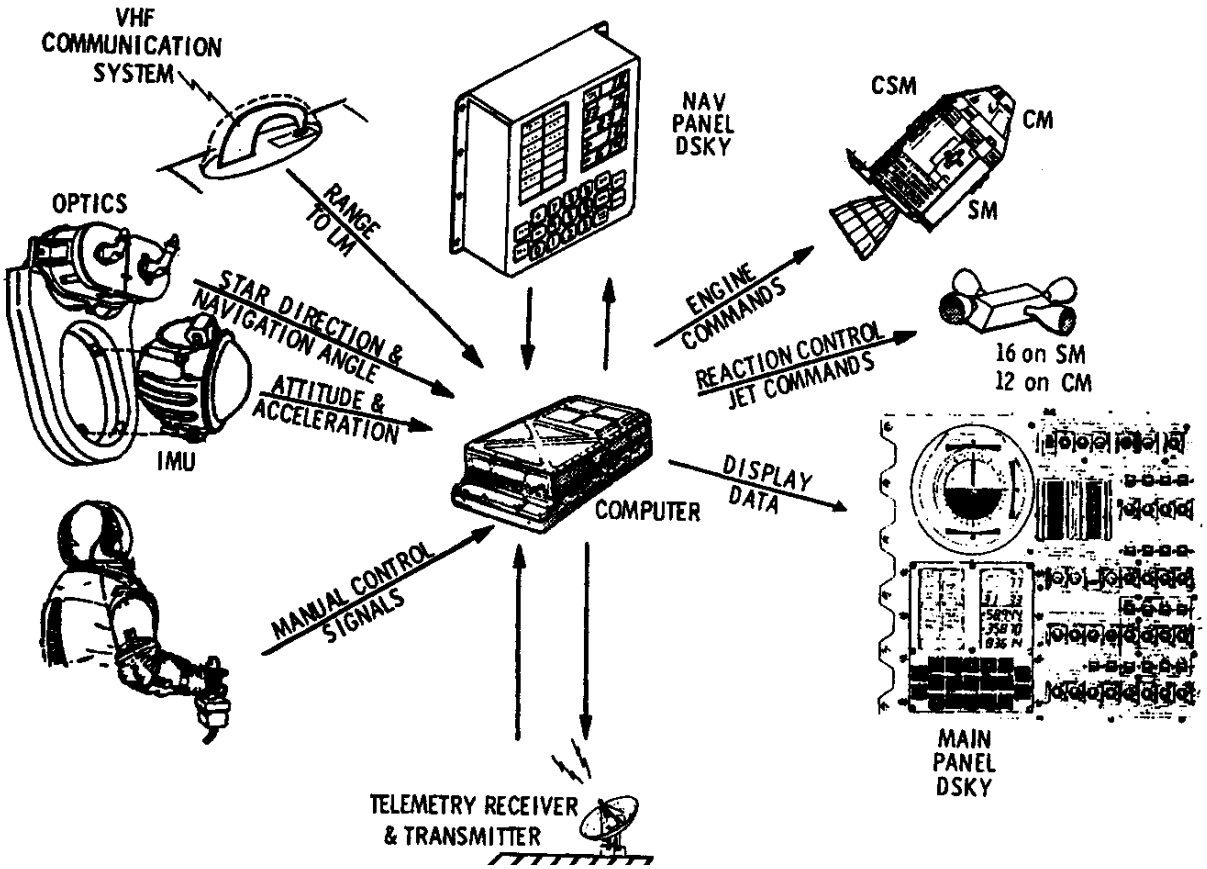


Abbildung 20. Systemschnittstellen AGC Block II [33],[53, S. 67]

7.2 Funktionsüberblick

Nie zuvor wurde ein Navigationscomputer für den Betrieb im Flug entwickelt, der einen derart umfangreichen Funktionsumfang hatte, wie der AGC ihn erhalten sollte. Bereits die Block I Version, die auf den unbemannten Flügen eingesetzt wurde, hatte ein großes Aufgabengebiet, inklusive Ein- und Ausgabe für den – auf den unbemannten Flügen natürlich nicht vorhandenen – Astronauten (siehe Abbildung 14). In der Block II Version kam dann noch die Steuerung der VHF-Kommunikation und die direkte Steuerung der RCS-Thruster (Reaction Control System) hinzu (siehe Abbildung 20). Aus Abbildung 20 ist auch ersichtlich, dass die manuelle Steuerung ebenfalls über den AGC läuft und somit keineswegs unabhängig vom Computer ist.

Die Architektur des AGC lässt sich in sieben Bereiche unterteilen [101]:

- Zeitgeber (timer) - für Synchronisierungsimpulse
- Sequenzgenerator (sequence generator) - regelt die Ausführung der Maschineninstruktionen durch Steuerimpulse
- Zentralprozessor (central processor) - Berechnungen
- Speicher (memory) - aufgeteilt in Festspeicher und beschreibbaren Speicher
- Prioritätensteuerung (priority control) - Steuerung der Prioritäten einzelner Operationen
- Ein/Ausgabe
- Energieversorgung (power) - 4-Volt- und 14-Volt-Zufuhr von Brennstoffzellen

Bei der AGC-Architektur handelt es sich also um eine Von-Neumann-Architektur. Der AGC war zwar speziell für die Navigation im Weltraum konzipiert worden, war aber dennoch ein Allzweckrechner, der unterschiedlichste Funktionen übernehmen konnte, wie z.B. die Übernahme der Steuerung der Saturn V oder die Durchführung von Missionsabbrüchen, welche je nach aktueller Flugphase unterschiedliche Anforderungen an das System stellten. Darüber hinaus bildet die Prioritätensteuerung ein auch heute noch aktuelles Scheduling-Verfahren, das in den 1960er-Jahren nicht nur hoch modern war, sondern auch nie zuvor für einen Navigationscomputer in einem Flugzeug oder Raumschiff eingesetzt worden war. Es gab vorher schlicht keine Computer, die zum einen leistungsfähig genug für die Navigation im Weltraum, zum anderen derart kompakt gebaut wie der AGC waren.

Die folgende Übersicht zeigt die technischen Daten des AGC Block II, also der Version, die in den bemannten Flügen zum Einsatz kam:

Taktfrequenz	1,024 MHz Taktgeber: Quarzoszillator mit 2,048 MHz Signalteilung für Mehrphasentakt
Wortlänge	16 Bits
Memory cycle time	11,7 μs
RAM	2.048 Words
ROM	36.864 Words
Maschineninstruktionen	34 reguläre + extended
Prioritäten Interrupts	11
Additionszeit	23.4 us
Multiplikationszeit	46.8 us
Input/Output	227 Circuits
Integrated Circuits	5.600
Energiebedarf	55 Watt
Gewicht	31,8 kg
Volumen	27,5 l

Tabelle 2. Technische Daten des AGC (Überblick), nach [53, Kapitel 1]

Neben der technischen Leistungsfähigkeit und der kompakten Bauweise musste der AGC natürlich auch die enormen physischen Belastungen des Starts in der Saturn V problemlos überstehen. Dies wurde zum einen durch stabile Umhüllungen des AGC, zum anderen durch äußerst robuste Bauteile erreicht. So basierte z.B. der Festspeicher, der sämtliche Navigationsprogramme enthielt, auf einer innovativen Technologie, die es ermöglichte, sehr stabilen Speicher mit dennoch geringem Volumen bzw. hoher Speicherdichte zu bauen.

7.3 Speicher

Der ROM des AGC war als Core Rope Memory realisiert. Bei diesem verlaufen Drähte durch bzw. neben ringförmigen Eisenkernen. Im Gegensatz zu Magnetkernspeichern wird hier nicht der magnetische Zustand der Kerne zum Speichern genutzt, sondern die Anordnung der Drähte durch bzw. neben den Kernen. Geht der Draht durch den Kern, so entspricht dies einer logischen 1, geht der Draht neben dem Kern vorbei, entspricht dies einer logischen 0. Es werden hier also auch Ringkerne genutzt, jedoch dienen diese als Transformatoren und eben nicht zum Speichern durch magnetische Ausrichtung der Kerne. Der Vorteil des Core Rope Memory ist seine Stabilität: Das Geflecht aus Eisenringen und Drähten hält auch starke Erschütterungen aus, was bei einem Flug zum Mond sehr wichtig ist. Darüber hinaus kann ein einzelner Ring mehrfach genutzt werden, indem mehrere Drähte hindurchgehen, wie in Abbildung 21 zu sehen ist.

Der Nachteil ist die nahezu unmögliche Fehlerkorrektur: Wenn ein Programm fehlerhaft ist, ist es äußerst schwierig herauszufinden, an welcher Stelle ein Draht durch einen Eisenkern verläuft anstatt neben dem Kern

(das gleiche gilt natürlich auch für den umgekehrten Fall, wenn ein Draht neben dem Kern verläuft, anstatt hindurch). Die Programme des Core Rope Memory wurden von Menschen, allerdings mit Maschinenunterstützung, gewebt. Filmaufnahmen des Webeprozesses sind in [87] zu finden, einem Bericht aus der Fernsehreihe „MIT Science Reporter“ von 1965.

Abbildung 21. Detail eines Core Rope Memory [62]

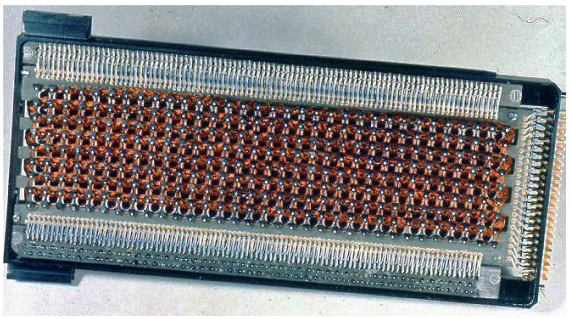


Abbildung 22. Der Core Rope Memory eines Apollo Guidance Computers [104]

7.4 Register

Die Wortlänge betrug 16-Bit, davon standen 15-Bit für Daten (inkl. 1-Bit für Vorzeichen) zur Verfügung, 1 Bit diente als Prüfbit [53, S. 69].

P	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Das „Hauptregister“ für Berechnungen war der Akkumulator, der als einziges Register eine volle Länge von 16-Bit besaß. Für Berechnungen mit doppelter Genauigkeit wurde zusätzlich das L-Register verwendet. Das Instruktionsformat besteht aus einem 3-Bit-Opcode und einer 12-Bit-Adresse:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Die drei Bits für den Opcode würden nur 8 Instruktionen ermöglichen. Die Entwickler nutzten verschiedene Techniken, um den Vorrat an möglichen Instruktionen über diese 8 Möglichkeiten hinaus zu erweitern. Hierzu zählte das Memory-Management, das als Memory-Banking den Speicher in mehrere Bereiche (Bänke) aufteilte [53, Kap. 10]. So wurden z.B. die Bits 12 und 11 verwendet, um zwischen beschreibbaren Speicher und mehreren Bereichen des Festspeichers zu unterscheiden. Daraus wiederum ergab sich die Möglichkeit, diese Bits als zusätzliche Bits für Instruktionen zu verwenden, wenn von vornherein feststand, auf welchen Speicherbereich ein bestimmter Befehl zugriff. Das traf auf Befehle wie TS (Transfer to Storage) oder ADS (Add to Storage) zu, bei denen es sich um Befehle handelte, die in den beschreibbaren Speicher schreiben.

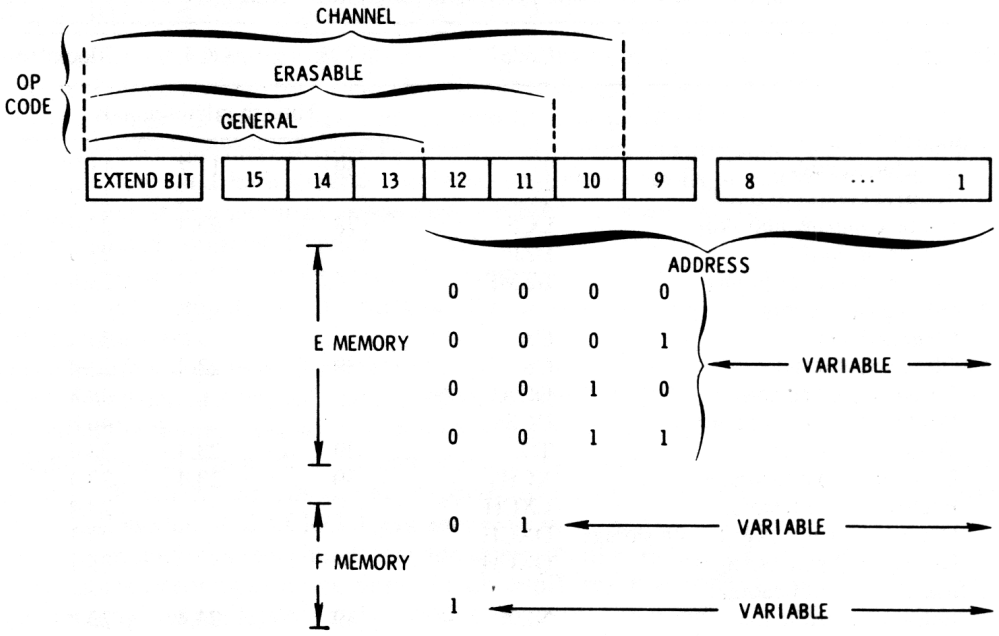


Abbildung 23. Befehlswort-Aufbau im AGC, Block II [53, S. 123]

Eine weitere Technik, die eingesetzt wurde, um den Befehlssatz zu erweitern, war es, einen zusätzlichen Satz an Opcodes bereitzustellen, der über eine spezielle Anweisung, EXTEND, angesprochen wurde [53, Kap. 5, 10]. Vor einer dieser erweiterten Anweisungen musste also jeweils die EXTEND-Anweisung aufgerufen werden, was zusätzlichen Overhead bedeutete. Damit der Overhead nicht zu groß wurde, verteilten die Entwickler die einzelnen Instruktionen aufgrund deren Verwendungshäufigkeit zwischen den „normalen“ und den „erweiterten“ Anweisungen. Abbildung 23 (aus „Journey to the Moon“) zeigt den Aufbau eines Befehlswortes im AGC (Block II) zur Nutzung solcher erweiterten Befehlssätze.

7.5 Software

Eine der größten Herausforderungen, die sich bei der Entwicklung des AGC ergaben, war die Softwareentwicklung. Die Ingenieure, die beauftragt wurden, den Navigationscomputer zu entwickeln, waren auch damit beauftragt, die nötigen Programme zu erstellen. Es gab jedoch weder Spezifikationen noch konkrete Vorstellungen darüber, was diese Programme zu leisten hatten [35]. Da aber eben diesen Programmen das Leben von Menschen anvertraut werden sollte, mussten zuverlässige Lösungen her.

Die Probleme nahmen zu. 1966 wurde klar, dass das MIT es nicht schaffen würde, einen derart leistungsfähigen und zuverlässigen Rechner zu bauen, wie es sich Charles Draper vorgestellt hatte, einen Computer, der die gesamte Navigation übernehmen konnte und dennoch in die Raumkapsel passte [35].

Daher wurde nun die Entscheidung getroffen, die Hauptnavigation vom Boden aus durchzuführen. Der AGC sollte nun nur noch als Backup dienen. Allerdings musste er trotzdem noch die Möglichkeit haben, die Navigation selbstständig durchzuführen: In der Zeit, in der sich das Apollo-Raumschiff hinter dem Mond befand, konnten es die Navigationsinformationen von der Erde nicht erreichen. Außerdem sollte der AGC eben auch dann übernehmen können, wenn die Funkverbindung zur Erde aus irgendwelchen anderen Gründen abbrach. Interviews mit den Entwicklern bezüglich der Software-Thematik und den damit verbundenen Entscheidungen sind in der Dokumentation „Moon Machines“ zu finden, und zwar in der Folge „The Navigation Computer“ [35].

Diese gegenüber den ursprünglichen Wünschen reduzierten Anforderungen wurden nun mit großem Engagement umgesetzt. Dabei bestand weiterhin das Problem, dass zahlreiche unterstützende Techniken des Software-

Engineering noch gar nicht existierten. Zahleiche Aspekte wurden während der Entwicklung angegangen und führten zu neuen Lösungen, wie sich Margaret Hamilton, eine der Software-Entwickler des AGC, erinnert [61].

Margaret Hamilton entwickelte später die Universal Systems Language, USL. Dabei handelt es sich um eine Systemmodellierungssprache mit Parallelen zu UML oder SysML. USL basiert auf den Erfahrungen, die während der Arbeit an der AGC-Software gemacht wurden. Hamilton entwickelte zusammen mit William R. Hackler auch eine integrierte Entwicklungsumgebung, die „001 Tool Suite“, die auf USL basiert und Tools zur Modellierung, Analyse, Simulation und Codegenerierung beinhaltet [60],[59].

Es ist nachzuvollziehen, dass Hamilton einem solchen ganzheitlichen Ansatz folgte, da während dar Entwicklung der AGC-Software die Arbeit nicht in spezielle Abschnitte wie z.B. Planen, Modellieren oder Implementieren eingeteilt war; die Aufgabe des Entwicklerteams war es eben, die Programme für den Navigationscomputer zu liefern. Diese wenig spezifische Aufgabenstellung gab den Entwicklern aber auch viel Freiheit, die Entwickler wechselten daher häufig ihre Rollen. Auch durch Erfahrungen, die während der Arbeit gemacht wurden, änderte sich die Art der Entwicklungsarbeiten. Hamilton kam aufgrund dieses gesamtheitlichen Ansatzes zu dem Schluss, das ein System ein System ist, egal ob es auf Algorithmen-Ebene, auf Software-Ebene oder auf Ebene des Systems betrachtet wird, welches es ausführt („...a system is a system, whether in the form of higher-level algorithms, software that implements the algorithms, or systems that execute them.“) [61].

Das Beispiel USL zeigt damit bereits, dass die Erfahrungen, die während Apollo gemacht wurden, sich auch auf dem Gebiet des Software-Engineerings weit über Apollo hinaus ausgewirkt haben.

Die Software für den AGC wurde auch in der Phase der bemannten Flüge des Apollo-Programms weiterentwickelt. Die grundlegenden Programme zur Verwaltung von Systemressourcen blieben zwar nahezu unverändert, jedoch hatten die einzelnen Missionen des Apollo-Programms jeweils spezielle Aufgaben bzw. Missionsziele zu erfüllen. Auch war der Umfang der späteren Missionen gegenüber den früheren stark erweitert. So führten z.B. die letzten drei Apollo-Missionen jeweils ein Lunar Roving Vehicle mit sich. Aus solchen Erweiterungen der Missionsziele und den speziellen Aufgaben in den einzelnen Missionen ergab sich der Bedarf an neuer bzw. geänderter Missionssoftware. Die in den bemannten Missionen eingesetzten Versionen sind die folgenden Versionen:

Name	Datum	Einsatz
Colossus 1, build 249	Oktober 1968	Command Module, Apollo 8 und 9
Colossus 2 [1]	1969	Command Module, Apollo 10
Colossus 2A (Comanche 055)	Juli 1969	Command Module, Apollo 11
Colossus 2C	nach Juli 1969	Command Module, Apollo 12
Colossus 2 (Comanche 072)	nach Juli 1969	Command Module, Apollo 13
Colossus 2E	1970	Command Module, Apollo 14
Colossus 3	Februar 1971	Command Module, Apollo 15-17
Luminary 1, revision 069	Dezember 1968	Lunar Module, Apollo 9 und 10
Luminary 1A, revision 099	Juli 1969	Lunar Module, Apollo 11
Luminary 1B, revision 116	nach Juli 1969	Lunar Module, Apollo 12
Luminary 1C, revision 131	Dezember 1969	Lunar Module, Apollo 13-14
Luminary 1E/G [1]	Mai 1971	Lunar Module, Apollo 15-17

Tabelle 3. Versionen der AGC-Software nach [137, S. 274] und Colossus/Luminary-Listings

Eine umfangreichere Darstellung der Programmversionen ist in Abschnitt 11.2 zu finden.

7.6 Display and Keyboard (DSKY)

Die Interaktion mit einem Computer erfolgte zur Zeit der Entwicklung des AGC noch größtenteils über Lochkarten, die dem Computer in Stapeln („Batches“) zugeführt wurden. Die direkte Kommunikation mit einem Compu-

[1] Quellenlage nicht eindeutig

ter steckte noch in den Kinderschuhen; 1961 entwickelte Fernando J. Corbató das bereits angesprochene Time-Sharing-Verfahren [166], welches er 1962 beschrieb [28]. Dabei wird die Prozessorkapazität auf mehrere Benutzer aufgeteilt, so dass freie Kapazitäten besser genutzt werden können. Corbatós erstes Time-Sharing-System, das Compatible Time-Sharing System (CTSS), entwickelte er 1963 weiter zu Multics, was zum Vorläufer von Unix wurde. CTSS ermöglichte eine direkte Interaktion zwischen Benutzer und Computer, allerdings handelte es sich bei diesen Computern um Mainframes, so nutzte Corbató z.B. den IBM-7094 für sein System.

Bei einem derart kleinen Computer, wie er im Apollo-Raumschiff eingesetzt wurde, war ein System wie Multics oder CTSS nicht einsetzbar, hier mussten andere Lösungen her. Doch wie gestaltet man die Interaktion mit einem Computer möglichst einfach und effizient? Fragen wir uns dazu, wie ein Computer heute, im 21. Jahrhundert, benutzt wird. Sehen wir uns einige Beispiele an:

```
jim@Neptun:~$ cd ~jim/Dokumente/
```

Dieser Unix-Befehl setzt das aktuelle Verzeichnis auf das Unterverzeichnis „Dokumente“ des Home-Verzeichnis des Benutzers „jim“. Hierbei wird zunächst eine Aktion angegeben (cd für change directory) und dann ein Objekt (das Verzeichnis jim/Dokumente). Diese Abfolge findet sich auch bei menügesteuerten Anwendungen wieder, wenn auch ggf. durch Menüabfolgen zusätzlich strukturiert. In einem LaTeX-Editor z.B. kann man mit den Menübefehlen „Datei“$\longrightarrow$ „Speichern als“ die auszuführende Aktion wählen und dann das Objekt, nämlich den Dateinamen. Auch bei Benutzersteuerungen, die auf klassische Auswahlmenüs weitgehend verzichten, wie z.B. die Art der Steuerung in den MS-Office-Programmen ab der Version 2007, bei denen es anstelle der klassischen Menüs die sogenannten Menübänder gibt, findet sich diese Struktur wieder: Um z.B. eine Grafik in ein MS-Word-Dokument (ab Version 2007) einzufügen, wählt man im Menüband die Registerkarte „Einfügen“, dann klickt man auf „Grafik“ und wählt schließlich das Objekt (die Grafik) für die Aktion „Einfügen“.

Die Entwickler des AGC haben eine solche Abfolge von Aktionen und Objekten zur Grundlage für die Interaktion mit dem Computer gemacht und das „Display and Keyboard“, kurz DSKY, als Benutzerschnittstelle entworfen. Das DSKY war die Zentrale Schnittstelle zwischen Nutzer und Computer. Dabei handelt es sich um eine digitale Ein- und Ausgabeeinheit. Die Aufteilung der Eingaben in Aktionen und Objekte wurde hier mittels eines „Verb-Noun“-Schemas umgesetzt. Bei den Verben handelt es sich um die jeweils auszuführende Aktion, die Nouns (also Hauptwörter) benennen die dazugehörigen Objekte.

Beide Eingaben erfolgen in oktaler Form. Die Notation eines Verb-Noun-Befehls erfolgt nach folgendem simplen Schema:
V[Oktalzahl]N[Oktahlzahl]E wobei V für Verb, N für Noun und E für Enter steht. Also zum Beispiel: V16N65E (V16: Kontinuierliche, dezimale Anzeige, N65: Zeit). Auf ein Verb muss nicht zwangsläufig ein Noun folgen; der Befehl zur Wahl eines neuen Programms lautet beispielsweise V37E. Anschließend gibt man die Oktalzahl des neuen Programms ein und bestätigt nochmals mit Enter. Um den Rechner z.B. in den Idle-Mode zu versetzen, wird Programm 00 aufgerufen. Die Befehlsfolge lautet also V37E00E.

Damit bricht der Wechsel eines Programms gewissermaßen aus dem Verb-Noun-Schema aus, da eine Eingabe von V37N00E durchaus auch Sinn machen würde. Durch die Verwendung von V37E wird ein Programmaufruf also gewissermaßen zu einer besonderen Art von Verben.

Das Idle-Programm 00 (von den Astronauten schlicht POO genannt) ist von besonderer Bedeutung, da bestimmte Aktionen, wie z.B. ein Upload von der Bodenstation aus, nur im Idle-Mode erfolgen kann. Nicht nur Programme und Aktionen lassen sich über das DSKY eingeben, auch auf den Speicher kann zugegriffen werden. Mittels V27N02E[Adresse]E lässt sich der Core-Rope auslesen:

Beispiel (Apollo 11 Comand Module Software): Eingabe: V27N02E02000E Ausgabe: 00063. Im Anhang, in Abschnitt 16.2, sind weitere bedeutende Verbs und Nouns aufgelistet.

Abbildung 24. Die DSKY-Einheit aus Odyssey, Apollo 13

Ob die DSKY-Steuerung aber die Menüsteuerung moderner Systeme beeinflusst hat, lässt sich nicht mit Sicherheit feststellen. Die Aussage von Prof. Corbató zur Geheimhaltung in der Entwicklung des AGC (siehe Abschnitt 2.5) lässt einen solchen Einfluss eher unwahrscheinlich erscheinen. Allerdings wurde das DSKY-Konzept als solches der Öffentlichkeit recht früh vorgestellt, z.B. 1965 in [87]. Somit könnten andere Entwickler durchaus Anregungen für eigene Entwicklungen in der DSKY-Steuerung gefunden haben. Ebenfalls denkbar ist natürlich, dass die Entwicklungen der verschiedenen Benutzerschnittstellen in den 1960er-Jahren (wie eben z.B. das DSKY und die Entwicklungen von Douglas Engelbart) zwar unabhängig voneinander verliefen, aber auf gemeinsamen Grundlagen beruhten. Solche Grundlagen könnten z.B. die Arbeiten von Noam Chomsky gebildet haben, wie z.B. sein 1956 erschienener Artikel „Three models for the description of language“ [22].

Die Sichtweise der Nutzer des Systems war bei der Entwicklung natürlich auch sehr wichtig und könnte ebenfalls zum Konzept der Benutzerschnittstelle beigetragen haben. Als Verbindungsmann zwischen den Entwicklern und den Nutzern diente David Scott, der am MIT Instrumentation Lab trainiert wurde. David Scott flog zusammen mit Neil Armstrong die Gemini VIII Mission, bei der das erste erfolgreiche Andockmanöver im All stattfand. Im Apollo-Programm fuhr Scott dann den ersten Lunar Rover auf dem Mond. David Scott lobte auch später noch den AGC und dessen Benutzerinterface [148].

7.7 Betriebssystem Executive

Als Betriebssystem kam ein prioritätengesteuertes Multiprozesssystem mit dem Namen „Executive“ zum Einsatz. Dieses ermöglichte die gleichzeitige Verarbeitung von bis zu acht Prozessen, wobei einer davon ein Leerlaufprozess war, die übrigen sieben waren „arbeitende“ Prozesse. Das Scheduling erfolgte dabei hauptsächlich mittels

kooperativem Multitasking (siehe auch Abschnitt 7.1). Die Prozesse mussten das „NEW JOB“-Register periodisch (alle 20 ms) abfragen, dieses Register zeigte an, ob ein Prozess höherer Priorität vorlag. Da es sich beim AGC um einen Computer mit festgelegtem Bestand an Programmen handelte, der auch nicht laufend Software-Updates erhielt, funktionierte dies gut. Die Entwickler hatten immerhin die volle Kontrolle über die Programme des AGC. Dennoch waren sie sich durchaus bewusst, dass es möglich wäre, dass ein einzelner nicht reagierender Prozess die Arbeit anderer Prozesse blockieren könnte. Daher wurde im AGC auch eine preemptiv arbeitende Komponente, ein „Nachtwächter“, eingesetzt, die „Night Watchman“-Fehlererkennung: Das „NEW JOB“-Register enthielt die Information darüber, welcher Prozess gerade die höchste Priorität hatte. Erfolgte nun die Abfrage des NEW JOB-Registers durch die laufenden Prozesse nicht oft genug, verursachte der „Night Watchman“ einen Restart [53, Ch. 6, 14], [26]. Daher auch der Einschub „Hauptsächlich“, zu Beginn dieses Abschnitts. Das Scheduling des AGC war also eine Mischung aus kooperativem und preemptivem Multitasking. Bei einem Restart eines Prozesses musste dieser auch nicht zwangsläufig komplett von vorn beginnen und seine (möglicherweise komplexen) Berechnungen komplett wiederholen; Restart-Tables ermöglichten es, einen Prozess an einem Restart-Point fortsetzen zu können. Da mehrere, nicht synchron laufende Prozesse gleichzeitig an der Arbeit sein konnten, die somit auch unterschiedliche Abarbeitungsfortschritte aufweisen könnten, war es nötig, Restart-Points unabhängig voneinander fortzuschreiben. Zu diesem Zweck waren die Restart-Tables in fünf Gruppen unterteilt [26]. Damit dieses unabhängige Fortschreiben der Restart-Tables für die laufenden Prozesse funktionierte, mussten sich die gleichzeitig laufenden Prozesse dementsprechend in unterschiedlichen Restart-Gruppen befinden [26].

Darüber hinaus existierte noch die „Waitlist“, eine interruptgesteuerte Komponente, die für zeitgesteuerte Prozesse zum Einsatz kam. Bei diesen Prozessen, den sogenannten „Waitlist-Tasks“, handelte es sich um kurze Anweisungsblöcke, wie z.B. der Signalisierung des Ablaufs eines Timers oder auch das Auslesen der Beschleunigungssensoren. Durch einen Interrupt wurde der Prozessor in seiner aktuellen Tätigkeit unterbrochen, Fortsetzungsinformationen zum laufenden Prozess wurden gespeichert und schließlich der Code der jeweiligen „Waitlist-Tasks“ ausgeführt. Während der Ausführung von „Waitlist-Tasks“ waren (weitere) Interrupts unterbunden, was einen weiteren Grund dafür lieferte, in diesem Modus nur kurze Anweisungsblöcke auszuführen. Wenn ein längerer Prozess über die Waitlist ausgeführt werden sollte, so wurde nur die Planung dieses Prozesses unter deaktivierten Interrupts durchgeführt. Auch die Interrupt-Prozesse unterlagen Sicherheitsmaßnamen: Das Betriebssystem prüfte, ob eine Waitlist-Task zu viel Zeit im Modus mit deaktivierten Interrupts verbrachte („Rupt-Lock“) [56, Abschnitt 3].

7.8 Abort Guidance System

Das Abort Guidance System (AGS) war ein Backup-Computer für den AGC im Lunar Module. Es handelte sich dabei um einen MARCO 4418 18-Bit-Computer mit 2K Festspeicher und 2K beschreibbaren Speicher. Wie der Name bereits vermuten lässt, war dessen Hauptaufgabe ein geregelter Missionsabbruch. Genauer gesagt: ein Abbruch der Mondlandung.

Das LM enthielt das erste fahrzeugfeste inertiale Navigationssystem („Strapdown Inertial Measurement Unit“). Bei einen solchen Navigationssystem sind die Beschleunigungsmesser fest mit dem Fahrzeug verbunden und machen dessen Bewegungen mit. Dies ist robuster als ein inertiales Navigationssysteme mit kardanischer Aufhängung, erfordert aber einen höheren Rechenaufwand. Das AGS erhielt seine Navigationsinformationen von diesem System. Für einen Abbruch wurde das AGS nie eingesetzt, es gab aber oft Hilfestellungen. So konnte es z.B. die Navigationsdaten des AGC verifizieren, wenn das LM hinter dem Mond war und daher kein Funkkontakt zur Erde bestand. Der Abort Guidance Computer verfügte über keine so ausgeklügelte Benutzerschnittstelle wie der AGC. Zur Dateneingabe in den AGS diente die „Data Entry and Display Assembly“ (DEDA). Diese verfügte über eine Eingabe- und eine Ausgabezeile sowie über eine Tastatur, die eher an einen Taschenrechner erinnerte als an einen Navigationscomputer. Speicherinhalte wurden über die Eingabe direkt manipuliert, ein Verb-Noun-Verfahren oder etwas Vergleichbares existierte hierfür nicht. Es war allerdings auch möglich, Daten vom AGC an das AGS zu übertragen.

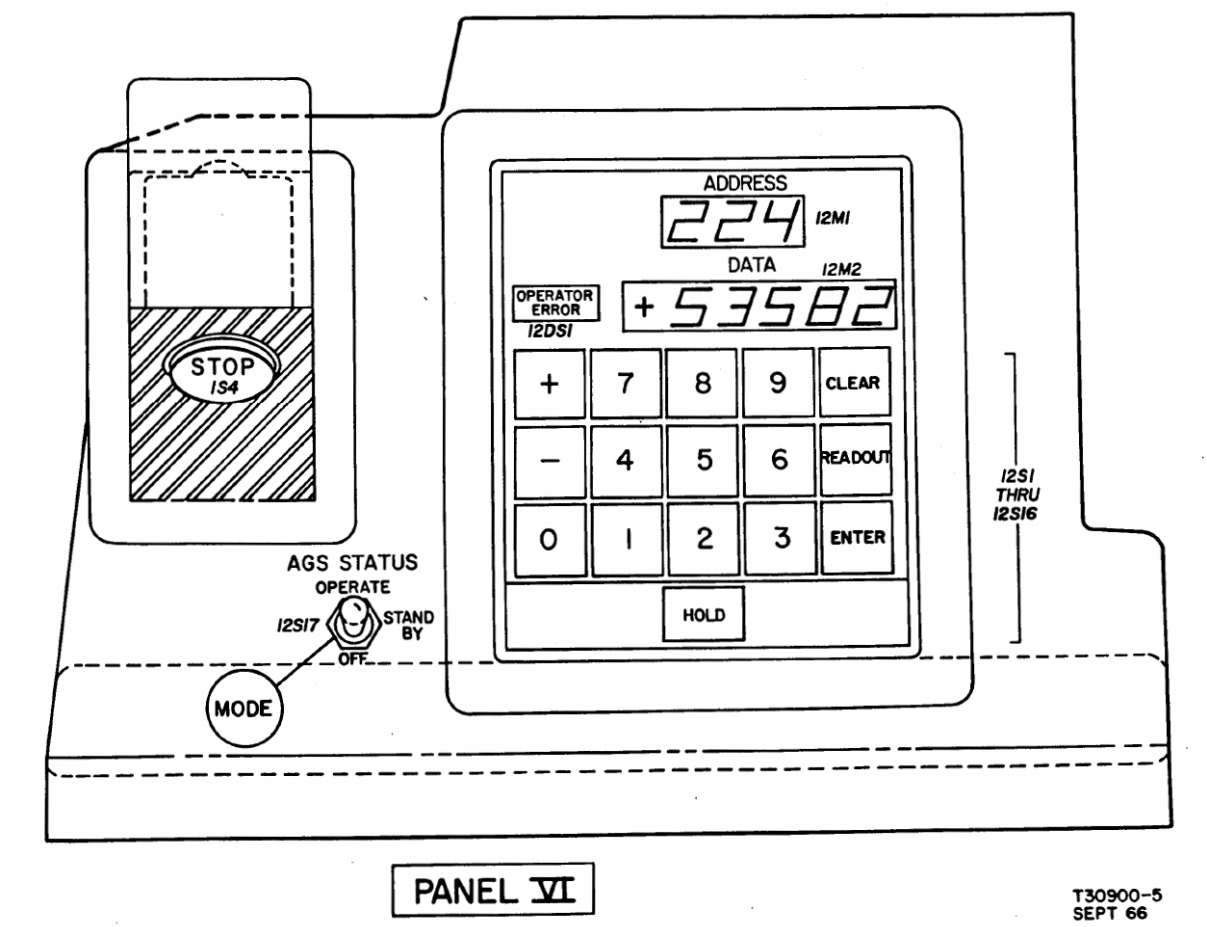


Abbildung 25. Skizze der „Data Entry and Display Assembly“ (DEDA) des Abort Guidance Systems [48]

Vergleicht man die Steuerung des AGS über die DEDA mit der DSKY-Steuerung des AGC, wird deutlich, welchen Fortschritt das DSKY darstellte, insbesondere für einen Navigationscomputer aus den 1960er-Jahren. Auch das AGS war ein fortschrittlicher Computer, hatte jedoch aufgrund des relativ geringen Speichers von 4K nicht die Möglichkeiten, eine solch ausgeklügelte Benutzerschnittstelle zu unterstützen wie sie der AGC besaß.

8 Launch Vehicle Digital Computer

Der Launch Vehicle Digital Computer (LVDC) war der Computer, der die Saturn V Rakete nach dem Start auf der korrekten Flugbahn hielt.

8.1 Saturn V

Aufbauend auf seinen Erfahrungen mit der V2 entwickelte Wernher von Braun nach dem Zweiten Weltkrieg die Raketentechnik zusammen mit etwa 700 weiteren deutschen Ingenieuren in den USA weiter. Die Rakete für die Flüge zum Mond erhielt den Namen Saturn, die Entwicklung der Saturn begann 1959.

Geplant wurde die Trägerrakete in drei Haupttypen: Saturn A, Saturn B und Saturn C. Diese Haupttypen waren wiederum jeweils in mehrere Untertypen aufgeteilt, so dass es acht Konfigurationen gab: A-1, A-2, B-1 und C-1 bis C-5.

Tabelle 4 zeigt die verschiedenen, ursprünglich geplanten Konfigurationsmöglichkeiten. Bei „Titan“ und „Centaur“ handelte es sich um eigenständige Trägerraketen bzw. -stufen. Trägerraketen der Titan-Reihe, genauer Titan II Raketen, kamen im Gemini-Projekt zum Einsatz [11, S. 21]. Bei der Centaur handelte es sich um eine sehr erfolgreiche Raketenstufe, die auch weiterhin im Einsatz ist [122]. Bei den Stufen mit „S“ am Anfang handelte es sich um die für die Saturn neu entwickelten Stufen.

Die Entscheidung fiel schließlich auf die Variante C-5. Die C-1 wurde dennoch weiterentwickelt, da diese schneller fertig zu stellen war als die C-5, sie war dann auch bereits im Jahr 1961 fertiggestellt.

1962 entschied die NASA, dass eine stärkere Version der C-1 benötigt werden würde. Diese erhielt die Bezeichnung C-1B. 1963 wurde schließlich das „C“ aus den Bezeichnungen gestrichen, die Raketen hießen nun Saturn I, Saturn IB und Saturn V. 1966 startete die erste Saturn IB. Diese war für die Apollo Missionen von großer Bedeutung: Der erste bemannte Flug innerhalb des Apollo-Projekts nach der Brandkatastrophe von Apollo 1 war der erfolgreiche Flug von Apollo 7. Für diese knapp 11-tägige Mission vom 11. bis zum 22. Oktober 1968 wurde als Trägerrakete eine Saturn IB eingesetzt [11, S. 342-344].

Konfiguration	***Stufen***
A-1	Saturn S-I, Titan, Centaur
A-2	Saturn S-I, Jupiter, Centaur
B-1	Saturn S-I, Titan, S-IV, Centaur
C-1	Saturn S-I, S-IV,
C-2	Saturn S-I, S-II, S-IV
C-3 bis C-5	S-IC (5 F-1), S-II, S-IVB

Tabelle 4. Konfigurationen der Saturn-Raketen, nach [11, S. 47ff]

Abbildung 26. Start der ersten Saturn IB am 26. Februar 1966 [103]

Abbildung 27. Saturn I (1963) [119]

1967 war schließlich die größte der Saturn-Raketen fertig, die Saturn V. Die Saturn V ist bis heute die größte, schwerste und leistungsstärkste Trägerrakete, die je in den operativen Einsatz gelangte [112]. Die für die erste Stufe verwendeten F-1 Triebwerke sind die leistungsstärksten Raketentriebwerke, die je gebaut wurden. Die Saturn V besaß 5 dieser Triebwerke in der S-IC-Stufe.

Die technischen Daten sind daher auch aus heutiger Sicht noch beeindruckend:

Saturn V	
Höhe (inkl. Apollo-Raumschiff)	110.6 m (363 ft.) [113]
Durchmesser	10,06 m
Startgewicht	2812 t (6.486.873 lb) [2]
Startschub	33851 kN
Nutzlast	133 t

Tabelle 5. Saturn V, technische Daten

Oberhalb der dritten Stufe der Saturn V befand sich das Apollo-Raumschiff als Nutzlast. An der Spitze des Apollo-Raumschiffs befand sich noch eine weitere Rakete, das Launch Escape System. Dieses diente dazu, das Command Module mit der Crew im Falle eines Fehlers beim Start von der Saturn wegzuziehen. Dieses kam in keiner der bemannten Missionen des Apollo-Programms zum Einsatz, wurde jedoch in unbemannten Flügen erfolgreich getestet, z.B. in den Tests mit der Trägerrakete Little Joe II (1964-1966) [19, Anhang C] und beim Start von SA-6 (28. May 1964) [131, Kapitel 8], [11, S. 104, 328-329]. Die Position des Raumschiffs an der Spitze der Saturn-Rakete zusammen mit dem Launch Escape System machten das Apollo-Saturn-Raumfahrzeug zu einem sehr sicheren System; beim späteren Space Shuttle gab es kein Launch Escape System. Darüber hinaus war bei diesem das eigentliche Shuttle seitlich an den Haupttank angebracht, dies brachte die die Gefahr mit sich, dass sich lösende Teile des Haupttanks bzw. der Trägerraketen das Shuttle treffen konnten. Dies ist in der Tat dann leider auch geschehen: Beim Start der Mission STS-107 mit dem Shuttle Columbia am 16. Januar 2003 löste sich ein Stück Schaumstoff vom Haupttank und beschädigte den linken Flügel der Columbia. Diese Beschädigung führte dann dazu, dass die Columbia beim Wiedereintritt in die Erdatmosphäre am 1. Februar 2003 auseinanderbrach und alle sieben Besatzungsmitglieder ums Leben kamen [14, S. 9, 34, 49ff]. Ein solches Szenario hätte beim Start einer Saturn V nicht auftreten können. Ebenso natürlich auch nicht bei anderen Raketen, bei denen sich die Crew bzw. der Nutzlastbereich oberhalb der Trägerraketen befindet.

[2] Daten von Apollo 10, die Startgewichte variierten von Mission zu Mission [106]

Aktuelle NASA-Entwicklungen wenden sich sich heute, nach Ende der Space-Shuttle-Ära, auch wieder dieser erfolgreichen Technik zu und verwenden wieder Launch Escape Systeme. Zukünftige bemannte Missionen dürften daher eher an die Apollo-Ära als an die Shuttle-Ära erinnern. Dies ist ein Beispiel dafür, dass modernere Systeme nicht unbedingt „besser“ und vor allem nicht sicherer für die Crew sind als bereits erfolgreich eingesetzte ältere Systeme. Näheres hierzu ist in Abschnitt 14 zu finden.

Abbildung 28. Saturn V beim Start von Apollo 11, 16. Juli 1969 [90]

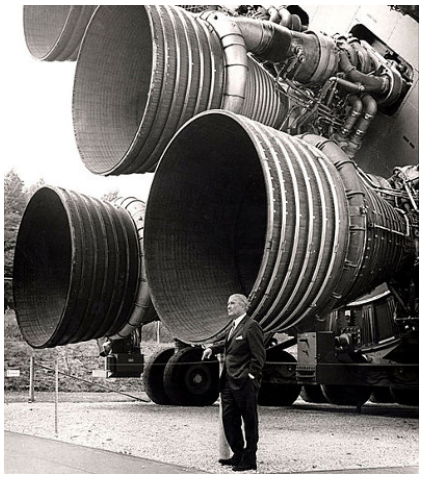


Abbildung 29. Wernher von Braun vor den F1-Triebwerken der ersten Stufe (S-IC) einer Saturn V [99]

Wie lief nun der Start einer Saturn V ab?

Die Saturn V war bis zum Start an der Startplattform verankert, musste dann aber in Sekundenbruchteilen von den Verankerungen gelöst werden. Bei T-8,9 Sek. beginnt die Zündsequenz der ersten Stufe [11, Anhang 2, S. 409]. Die Zündung des ersten F-1 Triebwerks dauerte bis ca. T-3,3 Sekunden [96]. Als erstes zündete das mittlere Triebwerk, ca. 0,3 Sekunden später zündeten dann zwei sich gegenüberliegende Triebwerke und nach nochmals ca. 0,3 Sekunden zündeten die letzten beiden Triebwerke [96]. Der volle Schub wurde zwei Sekunden nach erfolgter Zündung des ersten Triebwerks erreicht [96]. Nachdem dies geschehen war, begann bei T-1,0 Sekunden das Lösen der Verankerungen [11, Anhang 2, S. 409].

In den ersten Sekunden des Fluges war die Rakete noch recht langsam, beschleunigt dann aber stark. Dies ist in Filmaufnahmen von Starts der Saturn V erkennbar und geht auch aus den Transkripten des Apollo Flight Journal hervor (siehe z.B. [90] zu Apollo 11).

Zur Steuerung der Fluglage kam ein digitaler Computer zum Einsatz, der sich in der Instrument Unit befand.

8.2 Instrument Unit

Zwischen der dritten Stufe und dem Apollo-Raumschiff befand sich die „Instrument Unit“ (Abbildung 30).

In dieser ringförmigen Geräteeinheit befanden sich u.a. das Inertialnavigationssystem ST-124-M3, welches die Fluglage und die Beschleunigung maß, sowie verschiedene weitere Messsysteme, die zusätzliche Flugparameter und andere Systemparameter wie Druck, Temperatur, elektrische Spannung, etc. aufzeichneten. In der Instrument Unit befanden sich außerdem Kommunikations- und Telemetriesysteme, die die ermittelten Daten zu den Bodenstationen funkten und ein digitaler Computer, der Launch Vehicle Digital Computer (kurz LVDC), der als Autopilot fungierte.

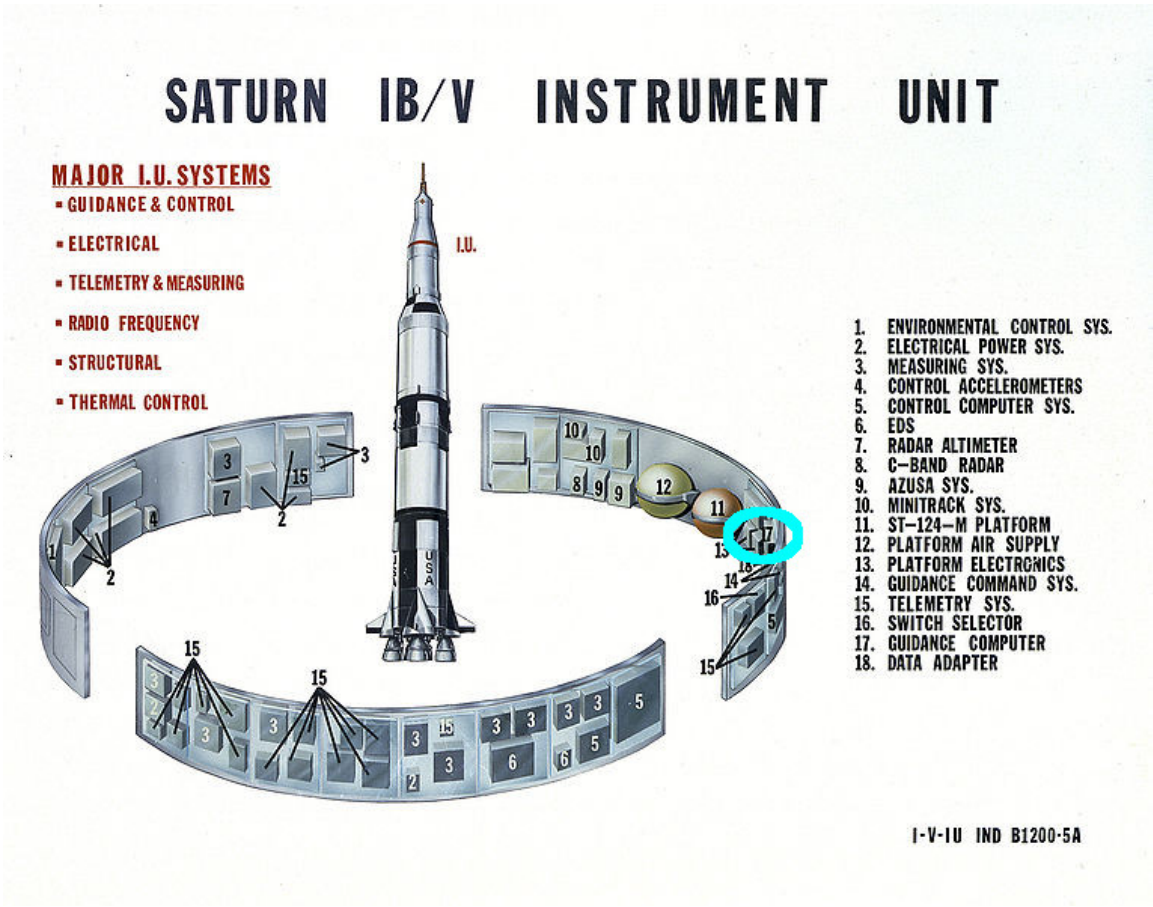


Abbildung 30. Instrument Unit der Saturn IB/V mit Position des LVDC (Hervorhebung von mir) [88]

Nach dem Start bis zur Trennung der ersten Stufe führte der LVDC ein zeitgesteuertes Programm durch, um die Rakete in die richtige Position und Lage zu bringen. Anschließend berechnete der LVDC durch Soll-Ist-Vergleiche der Flugbahn Kursdaten und erzeugte Steuersignale für die Korrekturtriebwerke. Dies lief folgendermaßen ab:

In den sogenannten „Major-Loops“ (1x pro Sekunde) berechnete der Computer den jeweils nächsten Soll-Punkt der Flugbahn. Nach einer Koordinatentransformation (raumbezogen – raketenbezogen) verarbeitete der Autopilot diese Daten dann in den „Minor-Loops“ weiter, indem er die jeweils aktuellen Messwerte der Fluglage und anderer Parameter mit den jeweiligen Sollwerten verglich. Zur Kurskorrektur wurden daraufhin die Triebwerksservos angesteuert. Dies führte zu 25 Korrekturimpulsen pro Sekunde

8.3 Die Technik des LVDC

Die folgende Übersicht zeigt die technischen Daten des LVDC:

- Taktfrequenz: 2,048 MHz
- Ringkernspeicher von 32.768 Datenworten, doppelt redundant.
- Datenwort bestehend aus 28 Bit
  - 25 Datenbit, 1 Vorzeichenbit, 2 Paritätsbits
- Siebenstufige Logikpipeline

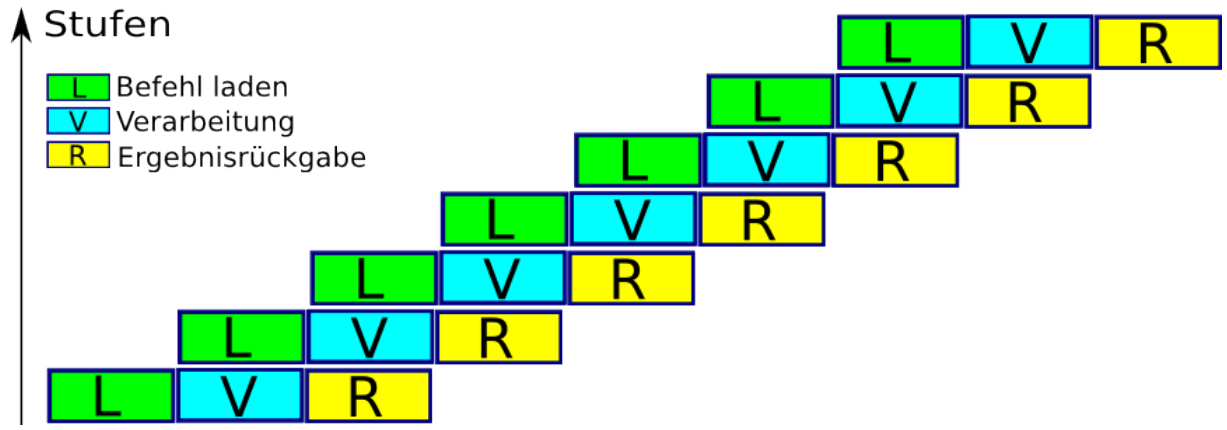


Abbildung 31. Siebenstufige Logikpipeline (Grafik des Autors)

Durch die siebenstufige Logikpipeline, wie sie Abbildung 31 zeigt, war der LVDC in der Lage, mehrere Aufgaben gleichzeitig zu bearbeiten. Dieser Computer ist somit ein frühes Beispiel für ein parallelverarbeitendes System.

Die Logikpipeline ist darüber hinaus dreifach redundant angelegt. Abbildung 32 zeigt ein Beispiel für eine redundante Schaltung mit Mehrheitsentscheidung, wie sie John von Neumann bereits 1956 darstellte [80]. Die Schaltung besteht aus drei identischen Gattern. Wenn eines davon versagt, funktioniert die Schaltung dank des Mehrheitsgatters dennoch korrekt. Doch hier besteht ein Problem, es sind ja nicht nur drei sondern vier Gatter in der Schaltung. Das Mehrheitsgatter könnte selbst ja auch fehlerhaft arbeiten. Und wenn die Forderung, dass eines der Gatter defekt sein darf, ohne die Funktion der Schaltung zu beeinträchtigen weiterhin aufrecht erhalten werden soll, so genügt der Ansatz aus Abbildung 32 nicht mehr. R. E. Lyons und W. Vanderkulk haben das Konzept von John von Neumann 1962 daher erweitert: Beim Launch Vehicle Digital Computer ist auch das Mehrheitsgatter selbst dreifach redundant vorhanden, wie Abbildung 33 aus der Arbeit von Lyons und Vanderkulk zeigt [80].

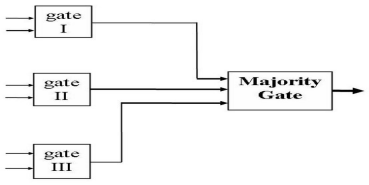


Abbildung 32. Drei identische Gatter mit Mehrheitsentscheidung [80]

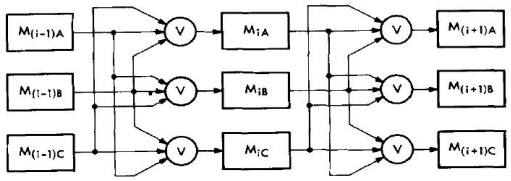


Abbildung 33. Redundante Gatter in der LVDC-Konfiguration [80]

Der LVDC hatte also Möglichkeiten zur Fehlerkorrektur, eine hohe Wortbreite und die Fähigkeit zur (quasi-) parallelen Verarbeitung. Warum wurde der LVDC dann nicht auch im Apollo-Raumschiff zur Navigation eingesetzt?

Das wurde durchaus ernsthaft überlegt, Eldon Hall zeigte jedoch die Nachteile einer solchen Lösung auf [55]. Der vermutlich bedeutendste Aspekt seiner Kritik bezieht sich auf die mangelnden Möglichkeiten, auf dem LVDC komplexe Navigationsprogramme auszuführen. Der Apollo Guidance Computer verfügte über eine zusätzliche Software-Ebene oberhalb der Assembler-Struktur: den Interpreter (siehe hierzu auch Abschnitt 11.4.2). Der Interpreter war speziell auf die Anforderungen der Navigationsprogramme hin entwickelt worden und unterstützte dementsprechend zahlreiche mathematische Operationen, wie z.B. Matrix- und Vektoroperationen, trigonometrische Funktionen und dergleichen mehr. Dies erhöhte die Flexibilität des Systems deutlich, jedoch auf Kosten der Geschwindigkeit, da die Interpreter-Instruktionen durch Assembler-Befehle des Basis-Befehlssatzes implementiert

waren. Eine solche Flexibilität war im LVDC nicht vorgesehen, von daher wäre es in der Tat schwieriger gewesen, die Navigationsprogramme auf dem LVDC zu implementieren.

Umgekehrt war der Apollo Guidance Computer durchaus in der Lage, den LVDC zu ersetzen, wenn dieser ausfallen sollte. Der AGC war wesentlich flexibler einsetzbar als der LVDC und erinnerte damit mehr an einen Universalrechner als an einen speziellen Steuercomputer.

Die von Hall vorgebrachten Argumente sind daher durchaus rational, allerdings spielte sicher auch der Wunsch, den Auftrag beim MIT zu behalten, eine Rolle. Hall übte auch Kritik an der Zuverlässigkeit des Systems. Und hier lässt sich schon eher an der sachlichen Berechtigung der Kritik zweifeln, da der LVDC aufgrund der Fehlertoleranz inkl. redundanter Gatter und einem redundanten Speicher durchaus sehr zuverlässig war. Die Aufgabe der Steuerung der Saturn V hat der LVDC auch bei sämtlichen Starts erfolgreich durchgeführt.

9 Navigationsnetzwerke des Apollo-Programms

Das Navigationssystem von Apollo bestand bei Weitem nicht nur aus den Computern an Bord der Raumfahrzeuge und dem RTCC; ein umfangreiches Netzwerk am Boden und in der Luft war nötig, damit die Astronauten sicher zum Mond und wieder zurück gelangen konnten.

Aufgrund der Rotation der Erde musste dieses Netzwerk große Teile der Erdoberfläche umspannen, damit dauerhafte Kommunikation mit dem Raumfahrzeug möglich war. Ein solches globales Netzwerk stellte bereits deshalb eine besondere Herausforderung dar, weil der größte Teil der Erdoberfläche, ca. 70%, mit Wasser bedeckt ist.

Darüber hinaus bestehen unterschiedliche Anforderungen an ein solches Netzwerk, je nachdem, welchen Erfordernissen die Kommunikation unterliegt. So unterscheiden sich z.B. die Anforderungen zur Kommunikation mit Objekten in niedriger Erdumlaufbahn (LEO – Low Earth Orbit) von denjenigen zur Kommunikation mit weiter entfernten Raumfahrzeugen bzw. Satelliten. Befindet sich ein Raumschiff im LEO, so ändern sich die Orte auf der Erdoberfläche, zu denen eine direkte Verbindung besteht, schneller als bei einem weiter entfernten Raumschiff. Daher müssen die Antennen für die LEO-Kommunikation die Raumfahrzeuge schneller verfolgen können als die Antennen, die weiter entfernte Raumschiffe verfolgen, brauchen aber wiederum nicht so leistungsstark zu sein wie diese, da die Entfernung ja geringer ist.

Außerdem stellen bemannte Missionen gegenüber unbemannten eine zusätzliche Herausforderung dar, da bereits aus Sicherheitsgründen möglichst umfassende und zuverlässige Kommunikation gewährleistet sein sollte.

Aus diesen Gründen existieren mehrere Netzwerke der NASA. Die für das Apollo-Programm wichtigsten dieser Netzwerke waren das Manned Space Flight Network, das speziell auf bemannte Missionen wie Mercury, Gemini und Apollo ausgelegt war und das Deep Space Network zur Überwachung weiter entfernterer Raumfahrzeuge und Raumsonden.

9.1 Manned Space Flight Network

Das Manned Space Flight Network (MSFN) war ein umfangreiches Netzwerk am Boden und in der Luft, das zur Flugüberwachung und Ferndatenermittlung (Telemetrie) der Apollo-Flüge eingesetzt wurde. Das MSFN bestand aus [124]:

- 14 Bodenstationen
- 4 Schiffen
- 8 Flugzeugen
- 2 Kommunikationssatelliten.

Die Hauptaufgaben des MSFN bestanden in:

- Telemetrie
- Bahnverfolgung

- Steuerung
- Sprechverbindung mit dem Raumschiff.

Schiffe und Flugzeuge wurden eingesetzt, um auch den mit Wasser bedeckten Teil der Erdoberfläche in das Netzwerk mit einbeziehen zu können. Da während eines Apollo-Fluges zahlreiche kritische Manöver nötig waren, bei denen sich das Raumfahrzeug an unterschiedlichen Positionen und in unterschiedlichen Entfernungen von der Erde befand, war ein solch ausgedehntes Netzwerk nötig, um eine möglichst dauerhafte Kommunikation zwischen Raumschiff und Mission Control zu gewährleisten.

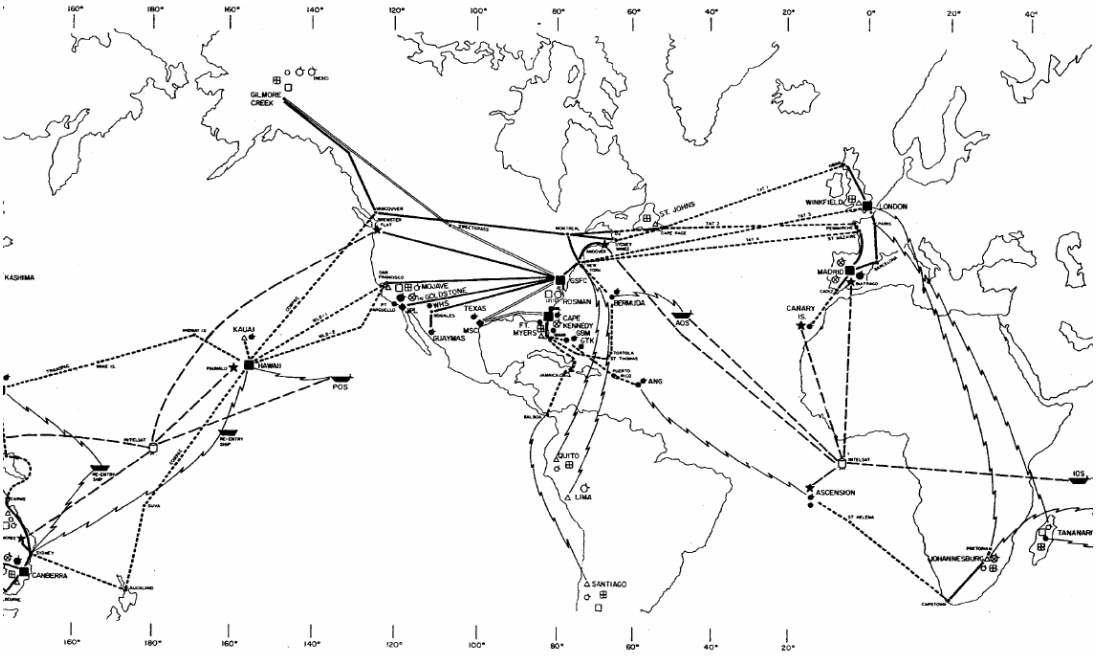


Abbildung 34. Manned Space Flight Network for Apollo, Ausschnitt [124]

Für diese Aufgaben waren zahlreiche Großrechner nötig. Die zentrale Komponente hierfür war der Real-Time Computer Complex im Mission Control Center. Doch auch die weiteren Bodenstationen setzten Großrechner zur Bewältigung der Kommunikation ein. Darüber hinaus wurden die Stationen des Deep Space Network eingesetzt, um den Kurs der Apollo-Raumschiffe zu verfolgen und Daten übertragen zu können. Insbesondere zum Empfang von Fernsehbildern wurden leistungsstarke Antennen benötigt, die sich an verschiedenen Punkten der Erdoberfläche befanden. So sind Fernsehsignale, die vom Mond kommen, also aus einer Entfernung von ca. 384.000 km, sehr schwach wenn sie die Erde erreichen. Daher wurden große Antennen benötigt, um diese Signale empfangen zu können.

9.2 Deep Space Network

Das heute noch existierende Deep Space Network (DSN) ist ein weltweites Netzwerk von Antennen und Kommunikationseinrichtungen. Die Stationen des DSN wurden während der Apollo-Missionen auch dafür verwendet, Fernsehbilder aus dem Raumschiff und von der Mondoberfläche zur Erde zu übertragen.

Für die Apollo-Missionen waren die folgenden Stationen von besonderer Bedeutung:

- Honeysuckle Creek (Australien)
- Parkes (Australien)
- Goldstone (USA)
- Fresnedilla (Spanien)

Die ersten Fernsehbilder vom Mond wurden von den NASA-Stationen in Honeysuckle Creek und in Goldstone sowie von der Antenne des Parkes-Observatoriums empfangen und übertragen. Die NASA schaltete zwischen diesen Stationen hin und her, um die jeweils bestmögliche Übertragungsqualität zu erreichen. Da die Qualität der Bilder, die das Parkes-Observatorium übertrug, bei weitem besser war als die der NASA-Stationen, wurde die TV-Übertragung ab der neunten Minute für die restlichen nahezu zweieinhalb Stunden komplett von der Parkes-Antenne übernommen. Die Ereignisse um die Übertragung der Mondlandung von Apollo 11 sind – allerdings in fiktiver Form – in dem Film „The Dish“ (Australien 2000) dargestellt.

Abbildung 35. Radioteleskop (64m) des Parkes-Observatoriums, [167]

Abbildung 36. DSS46-Antenne (26m), Honeysuckle Creek [81]

Die Parkes-Antenne wird nicht von der NASA betrieben, sondern gehört zur Commonwealth Scientific and Industrial Research Organisation (CSIRO) und ist Teil der Australia Telescope National Facility (ATNF) [25]. Die Fernsehbilder der ersten Mondlandung wurden somit nicht von NASA-eigenen Anlagen übertragen. Dieser Umstand wird von den Zweiflern bezüglich der Echtheit der Mondlandungen leicht übersehen. Betrieben wurde die Anlage auch während der Übertragung von CSIRO-Mitarbeitern und stand nicht unter der Kontrolle der NASA [24]. Das Fernsehsignal wurde von Parkes nach Sydney geleitet und dort aufgeteilt; ein Signal ging an das australische Fernsehen, das andere ging nach Houston. Dadurch erreichte das Fernsehsignal die australischen Zuschauer 0,3 Sekunden früher als die restliche Welt [24]. Bei einer vermeintlichen „Verschwörung“ hätten also die Mitarbeiter im australischen Parkes in diese mit eingeweiht sein müssen, was die Anzahl an „Mitwissern“ über den Bereich der NASA hinaus erhöht. Allein schon die große Anzahl von Mitarbeitern bei der NASA und beauftragter Institutionen lässt eine „Verschwörung“ sehr unwahrscheinlich erscheinen. Denn auch Jahrzehnte nach den Mondmissionen ist noch keiner dieser Mitarbeiter mit irgendwelchen Belegen einer gefälschten Mondlandung an die Öffentlichkeit gegangen. Doch aufgrund der Fernsehübertragung von Parkes aus, müssen auch noch die dortigen Mitarbeiter miteinbezogen werden, was eine bis heute anhaltende „Verschwörung“ doch äußerst unwahrscheinlich werden lässt.

Außer der Parkes-Antenne war die Station in Fresnedilla für die telemetrischen Daten von Apollo 11 von besonderer Bedeutung:

Beim LOI1-Manöver (Lunar Orbit Insertion 1) befand sich die Mondfähre hinter dem Mond und damit ohne Funkkontakt zur Erde. Bei einem erfolgreichen Manöver würde die Funkverbindung zu einem bestimmten, vorher berechneten Zeitpunkt wieder hergestellt sein. Dieser erste Funkkontakt und damit die Gewissheit, dass das Manöver erfolgreich verlaufen war, kam zum korrekten Zeitpunkt durch die Station in Spanien zustande.

Abbildung 37. DSN-Station Fresnedilla bei Madrid, Spanien [139]

Abbildung 38. DSS61-Antenne, Spanien (26m) [41]

9.3 Command Communication and Telemetry System

Zur Verarbeitung und Weiterleitung der telemetrischen Daten existierte in Houston ein eigenständiger Computerkomplex: das Command, Communications and Telemetry System (CCATS). Im CCATS liefen die Daten der einzelnen Stationen zusammen. Doch bereits in diesen Stationen erfolgte eine Vorverarbeitung der Daten, bevor diese an CCATS weitergeleitet wurden. Hierfür verfügten die Stationen über eigene Rechenanlagen. In Abbildung 37 sind z.B. UNIVAC-Rechner in der Station Fresnedilla zu sehen. Auch für die Übertragung von Steuerungsdaten in Richtung des Raumschiffs wurden diese Anlagen verwendet.

Doch nicht nur in den einzelnen Stationen des DSN wurden UNIVAC-Systeme eingesetzt, das zentrale CCATS im Mission Control Center in Houston bestand ebenfalls aus mehreren[3] UNIVAC-Rechnern, aus UNIVAC-494.

[3] Jesco von Puttkamer schreibt in „Abenteuer Apollo“, dass im CCATS drei UNIVAC-494 eingesetzt worden sind [141, S. 31], im Dokument APOLLO 10 FINAL FLIGHT MISSION RULES ist von zwei UNIVAC-494 die Rede [94, Section 4], was aber keinen Widerspruch bedeuten muss, da [94] ja speziell auf die Anforderungen von Apollo 10 eingeht und nicht auf die Kommunikationsanforderungen des gesamten Apollo-Programms.

Die NASA setzte also keineswegs nur Großrechner von IBM ein, auch der UNIVAC-494 war ein leistungsstarker Großrechner.

Die wichtigsten Daten der UNIVAC-494 Mainframes sind die folgenden [153]:

- Wortbreite 30 Bit
- Hauptspeicher 131 K Wörter
- Bis zu 24 I/O Kanäle
- Kommunikationssystem für bis zu 64 Fernbedienungsterminals
- Echtzeit-Betriebssystem

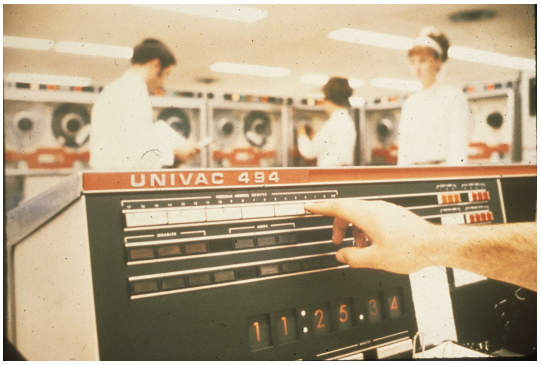


Abbildung 39. UNIVAC-494 [152]

Das Betriebssystem des UNIVAC-494 ist ein Multiprogramm-System, das sowohl als Real-Time-System genutzt werden kann als auch zur Batch-Verarbeitung. Dieses Betriebssystem ist laut Hersteller ausdrücklich modular aufgebaut, um zukünftige Erweiterungen zu unterstützen [153, S. 92]. Der Aspekt der Modularität, allerdings in Bezug auf den Apollo Guidance Computer, wird in den nächsten Abschnitten, in denen es um die Zuverlässigkeit geht, noch von Bedeutung werden. Auch die Peripherie des UNIVAC-494 war modular, es gab unterschiedliche Geräte zur Eingabe, zur Ausgabe und zum Speichern, wie z.B. CRT-Konsolen, Drucker, Lochkartenleser und -schreiber und verschiedene Magnettrommelspeicher (FH 880, FASTRAND II, FASTRAND III).

Bei den Trommelspeichergeräten hatten die FASTRAND-III-Geräte das größte Speichervermögen: 38.928.384 Wörter pro Einheit. Bis zu 8 Einheiten lassen sich zusammenschalten, was einem Gesamtspeichervermögen von 311.427.072 Wörtern entspricht (also mehr als 1100 MB, bzw. mehr als 1 GB, bei 1024 KB = 1 MB [4]).

Man sieht, dass auch für die Kommunikationssysteme moderne Großrechner mit hoher Leistungsfähigkeit eingesetzt worden sind. Die Mainframes im CCATS waren außerdem redundant ausgelegt, ein Gerät diente als Backup [94]. Sollten alle diese Kommunikationssysteme ausfallen, so wäre die Apollo-Besatzung dennoch nicht verloren gewesen, denn sie verfügte ja aber den Apollo Guidance Computer. Doch wie sehr widerstand dieser den Bedingungen im Weltraum und wie zuverlässig war er? Dies wird in den folgenden Abschnitten untersucht werden.

[4] Die Einheit KB mit großgeschriebenem „K“ verweist auf 1024 Byte, dies erfolgt zur Abgrenzung von „k“ für 1000

10 Zuverlässigkeit der Apollo-Computer

War der AGC leistungsfähig genug, um Menschen sicher zum Mond und wieder zurück zu bringen? Eldon Hall schreibt dazu:

> Requirements for a reliable computer with sufficient capacity and speed were quite clear, but what were sufficient capacity, speed, and reliability? How can these be achieved within the allowable size, weight, and power? Was it even possible with the technology available to meet the program schedules? If we knew then what we learned later or if a complete set of system specifications had been available at the time the contract was granted to MIT Instrumentation Laboratory, the designers might have concluded that there was no solution with the technology available in the early 1960s. [53, S. 66]

Lässt sich aus dieser Aussage schließen, dass der AGC mit der Technik der 1960er-Jahre überhaupt nicht hätte gebaut werden können? Natürlich nicht! Vielmehr zeigt sich hier, dass zu Beginn des Apollo-Programms noch keine genauen Spezifikationen festgelegt werden konnten, da noch nie zuvor ein solches Projekt angegangen worden war; ja die bemannte Raumfahrt selbst hatte gerade erst begonnen, Juri Gagarins Flug fand am 12. April 1961 statt [115], ein paar Wochen später, am 5. Mai 1961, folgte dann Alan Shepard als erster Amerikaner im All [93] und bereits am 25. Mai 1961 kündigte John F. Kennedy vor dem Kongress an, noch vor Ende des Jahrzehnts einen Menschen zum Mond und sicher wieder zurück bringen zu wollen [123]. Die einzelnen Elemente des Apollo-Programms wurden sozusagen „auf dem Weg zum Ziel“ entwickelt. Wenn zu bestimmten Themenbereichen, wie z.B. zu der Methode der Landung auf dem Mond, neue Informationen vorlagen, hat dies wiederum die weitere Entwicklung anderer Teilprojekte beeinflusst. Die Entwicklung war also höchst dynamisch; und das musste sie auch sein, denn aufgrund der eng gesetzten Frist zur Erreichung des Ziels, musste parallel gearbeitet werden. So begann die Arbeit am Apollo Guidance Computer auch bereits dann als die Methode der Mondlandung noch längst nicht feststand.

Es gab die unterschiedlichsten Bereiche, die berücksichtigt werden mussten. Technische Aspekte zur Stabilität des AGC wurden bereits im Abschnitt 7 behandelt. Doch welchen belastenden Faktoren war der Computer während seines Einsatzes ausgesetzt? Diese waren vor allem Punkte wie die Belastung durch Beschleunigung sowie Umwelteinflüsse wie Druck und Strahlung. Gerade die Strahlung, der sowohl Menschen als auch Material auf einer Reise zum Mond und zurück ausgesetzt sind, ist dabei von besonderer Bedeutung, denn: Wie sollte man die Auswirkungen von energiereicher Strahlung auf einen Computer (natürlich auch auf Menschen) testen? Und wie lässt sich für Abschirmung sorgen? Man könnte nun aber als Einwand die Gegenfrage stellen: Besteht denn überhaupt ein Strahlenrisiko bei einem Flug zum Mond und zurück?

Im Jahr 1958 wurde durch die von James Van Allen geleiteten Explorer-Missionen ein Bereich energiereicher Strahlung nachgewiesen [120]. Dieser Strahlengürtel, der die Erde in der Form eines Torus umgibt, wird nach James Van Allen als Van-Allen-Strahlungsgürtel bezeichnet.

10.1 Der Van-Allen-Strahlungsgürtel

Das Magnetfeld der Erde fängt energiegeladene Teilchen ein, die den Van-Allen-Strahlungsgürtel bilden. Dr. James Van Allen (University of Iowa, USA) konnte diese Strahlung 1958 mit den Explorer-Missionen nachweisen und mit Pioneer III (Dezember 1958) genauer vermessen.

Dieser Strahlungsgürtel besteht aus mehreren Bereichen, so dass man genauer von „den Strahlungsgürteln“ sprechen sollte. Während geomagnetischer Stürme können die Gürtel durchaus auch zu einem zusammenhängenden Gebilde verschmelzen, behalten eine solche zusammenhängende Form jedoch nicht dauerhaft bei [129]. Die Hauptgürtel beginnen in einer Höhe von ca. 1000 km über der Erdoberfläche und dehnen sich bis zu einer Entfernung von ca. 60000 km aus. In einem Bereich im Südatlantik vor der Küste Brasiliens, der Südatlantischen Anomalie, kann der innere Gürtel der Erde dabei wesentlich näher kommen als die genannten 1000 km, dort kann er der Erdoberfläche auf 200 km nahe kommen.

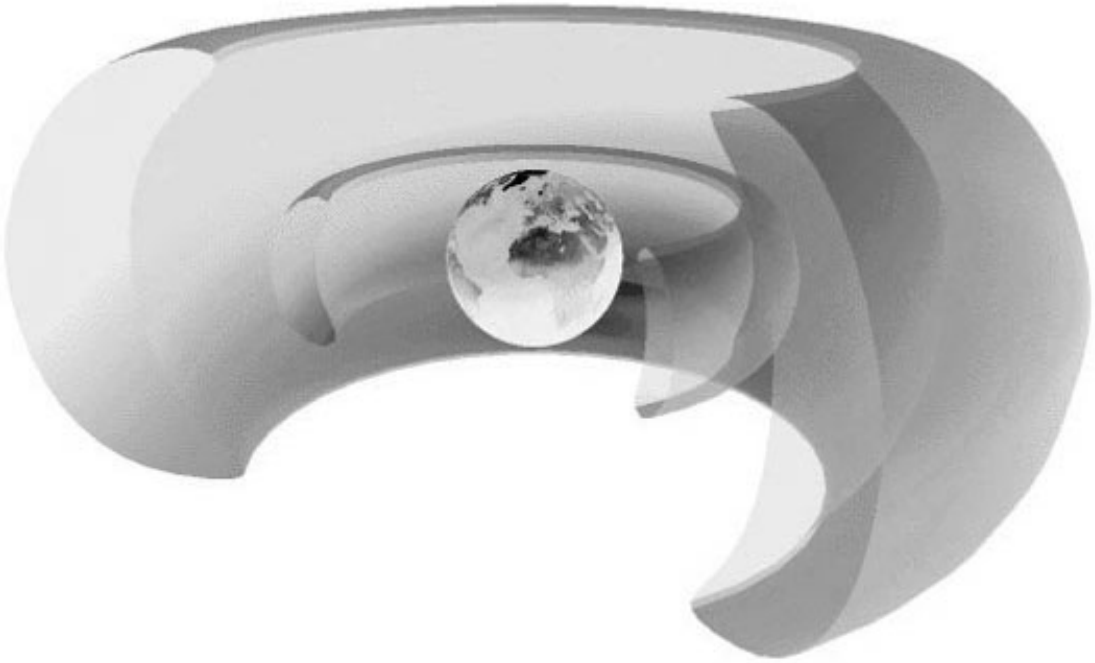


Abbildung 40. Der aus zwei Bereichen bestehende Van-Allen-Strahlungsgürtel [128]

Der äußere Gürtel beinhaltet hauptsächlich leichtere Teilchen, und zwar hoch energetische Elektronen, wohingegen der innere Gürtel sowohl energiereiche Elektronen als auch hoch energetische Protonen enthält. Dies ist darauf zurückzuführen, dass im Bereich des inneren Gürtels das Magnetfeld der Erde stärker wirkt und dadurch auch mehr Teilchen einfängt. Die Teilchen bewegen sich dabei spiralförmig entlang der Magnetfeldlinien der Erde (siehe Abb. 41).

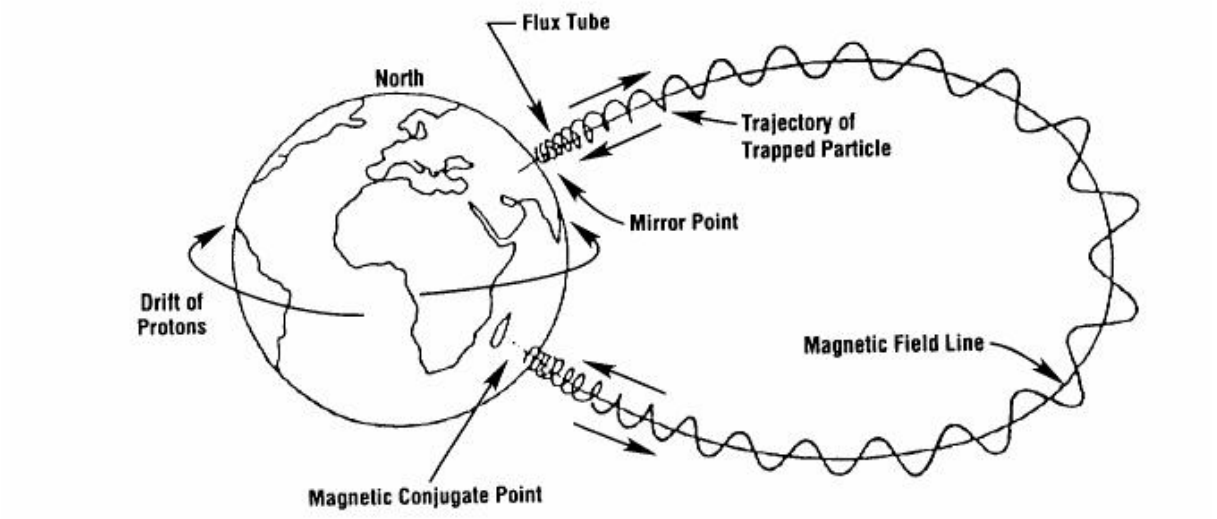


Abbildung 41. Spiralförmige Bewegung eingefangener Teilchen entlang der Magnetfeldlinien der Erde [10, S. 21]

Die Flussdichte der Teilchen innerhalb der Gürtel erreicht jeweils in deren zentralen Regionen die höchsten Werte. Dies stellte bereits James Van Allen 1958 mit der Raumsonde Pioneer 3 fest, die zwei Geiger-Müller-Zählrohre an Bord hatte und die Strahlungsgebiete kartografierte (S. Abb. 42) [163].

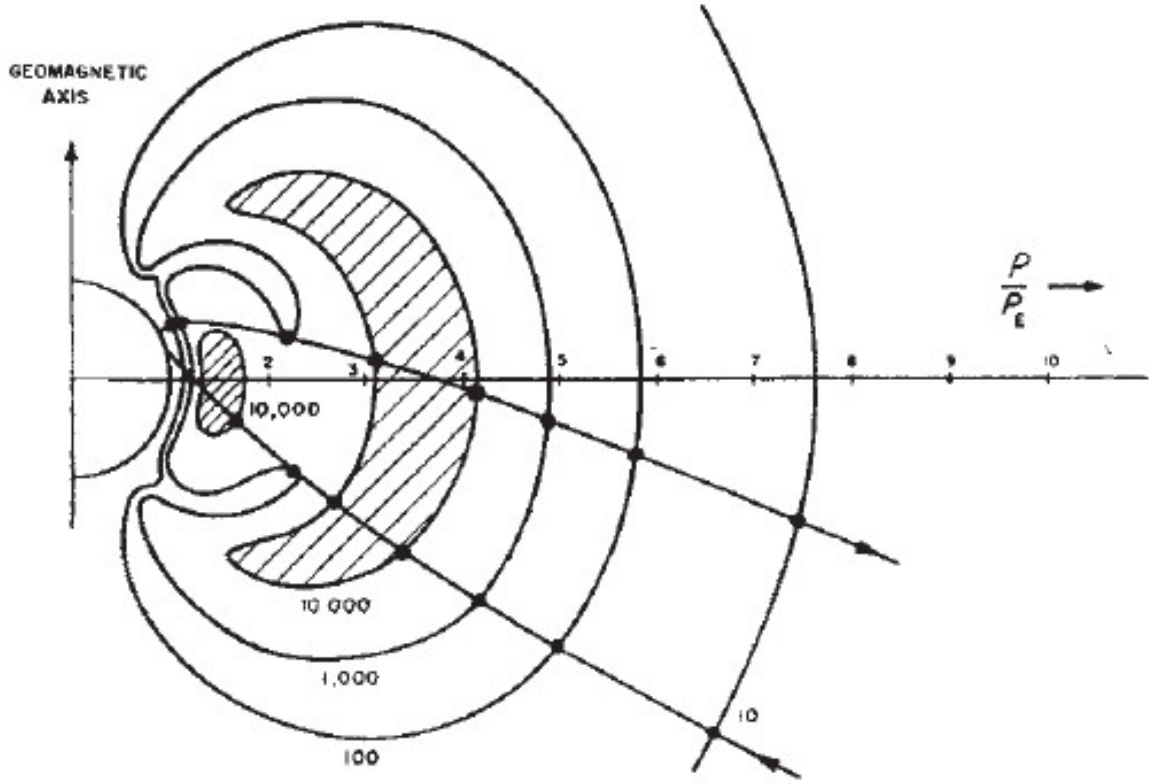


Abbildung 42. Intensitätsstruktur der eingefangenen Strahlung, dargestellt in einer geomagnetischen Meridianebene. Die Zahlen an den Regionen konstanter Intensität beziehen sich auf die Zählungen der Geiger-Müller-Zählrohre. [163, S. 433]

Aus Abbildung 42 ist auch zu entnehmen, dass der Bereich zwischen dem inneren und dem äußeren Gürtel zwar eine wesentlich geringere Strahlungsdichte als die Gürtel selbst hat, aber dennoch nicht frei von der Strahlung ist.

In späteren Missionen konnten die Strahlungswerte in den Van-Allen-Strahlungsgürteln wesentlich genauer gemessen werden. Am 30 August 2012 startete die NASA die „Radiation Belt Storm Probes“, die am 9. November 2012 in „Van Allen Probes“ umbenannt wurden [127]. Dabei handelt es sich um zwei Sonden, die die gleichen Messungen in räumlichen Abständen voneinander durchführen. Dadurch lässt sich feststellen, ob besonders auffällige Messwerte einzelne Spitzen sind oder Teil eines größeren Ablaufs. Das Verfahren lässt sich mit schwimmenden Korken auf einer Wasserfläche veranschaulichen: An den Auf-und-Ab-Bewegungen eines einzelnen Korken lässt sich nicht viel über die mögliche Ausdehnung einer Welle aussagen, nimmt man aber einen zweiten Korken hinzu und beobachtet, wie dieser (ggf. mit zeitlichem Abstand) ähnliche Bewegungen ausführt wie der erste Korken, so lässt sich die Größe der zu untersuchenden Welle wesentlich präziser ermitteln. Die Van Allen Probes konnten unter anderem einen weiteren, nur temporär (ca. 1 Monat) vorhandenen Strahlungsgürtel entdecken und streifenartige Muster in den Gürteln, die durch hindurch fliegende Materie hervorgerufen werden [126].

Sowohl der innere als auch der äußere Gürtel verändern aufgrund von Sonnenwinden ihre Form und Größe, der innere bleibt dabei aber relativ stabil, wohingegen der äußere wesentlich stärker variiert. Satelliten, die sich in

einer geosynchronen Umlaufbahn befinden (ca. 36.000 km Entfernung von der Erdoberfläche), unterliegen noch dem Einfluss der Van-Allen-Strahlungsgürtel. Da die Ausdehnung der Strahlungszonen variiert, befinden sich solche geosynchronen Satelliten jedoch nicht dauerhaft im Bereich der Strahlung, dennoch stellt die Strahlung der Van-Allen-Gürtel natürlich ein Risiko für diese Satelliten dar. Die Satelliten können sich durch die Strahlung elektrostatisch aufladen; erfolgt dann nach hoher statischer Aufladung eine plötzliche Entladung, so kann dies die Hardware des Satelliten beschädigen. Ein solcher Vorfall aus der näheren Vergangenheit hat z.B. im Jahre 2010 den Kommunikationssatelliten Intelsat Galaxy 15 beschädigt [42]. Zahlreiche weitere Beispiele werden im Buch „Spacecraft System Failures and Anomalies Attributed to the Natural Space Environment“ genannt, herausgegeben von der NASA [10]. Dabei ist zu beachten, dass die Ursache oft nicht direkt in den Van-Allen-Strahlungsgürteln zu finden ist, sondern häufig in Sonneneruptionen und den damit verbundenen Strahlungen, wie z.B. Röntgenstrahlung. Die von der Sonne ausgehenden Strahlungen wirken sich aber wiederum auf die Van-Allen-Strahlungsgürtel aus; beide Phänomene sind somit miteinander verwoben.

Die Regionen des inneren Gürtels mit der höchsten Strahlung würden getroffen werden, wenn man vom Äquator aus senkrecht „nach oben“ fliegen würde, wie aus Abbildung 40 ersichtlich ist. Die Apollo-Flüge haben diese Gebiete aufgrund des Winkels für die Trans-Lunar-Injection (TLI) umgangen. Dies ist z.B. ersichtlich aus den Flugplänen und Logbüchern der Apollo-Missionen (siehe hierzu z.B. das „Apollo Flight Journal“ zu Apollo 8 sowie zum Launch Window [121], [108]).

Außerdem bewegte sich Apollo so schnell, dass Raumschiff und Besatzung nur kurzfristig der erhöhten Strahlung der Van-Allen-Strahlungsgürtel unterlagen. Aus den Transkripten des „Apollo Flight Journal“ ist ersichtlich, dass die Region der Van-Allen-Strahlungsgürtel in wenigen Stunden durchflogen wurde. Robert A. Braeunig (Dayton, Ohio), der eine Website zur Geschichte der bemannten Raumfahrt betreibt in der er insbesondere die technischen Aspekte der Apollo-Missionen darstellt, hat für Apollo 11 sowohl die Flugbahn durch die Van-Allen-Strahlungsgürtel als auch die wahrscheinliche Höhe der dabei von den Astronauten aufgenommen Strahlung berechnet und kommt zu dem Schluss, dass dabei keine besonderen Risiken für die Astronauten bestanden [18].

Die Gefahr von Strahlenschäden während der Apollo-Missionen wurde von den Planern auch sehr ernst genommen. Im Apollo-Raumschiff befanden sich mehrere Messinstrumente, die Aufschluss über die jeweils vorhandene Strahlung gaben. In Bezug auf die Van-Allen-Strahlungsgürtel gab es den „Van Allen Belt Dosimeter“; dies war jedoch nicht das einzige Gerät, auch ein portabler Strahlenmesser wurde eingesetzt. Im Falle erhöhter Strahlungswerte konnten die Astronauten damit die Bereiche im Raumschiff ermitteln, in denen die Strahlung den geringsten Wert hatte, um sich dann vorübergehend möglichst dort aufzuhalten. Tabelle 6 listet die eingesetzten Geräte auf.

Instrument	Measurement	Location
Nuclear particle detection system	Alpha-proton spectrometer (4 channels proton, 15 m 1 50 MeV; 3 channels alpha, 40 m 300 MeV); telemetered	Service Module
Van Allen belt dosimeter	Skin and depth dose rates; telemetered	CM
Radiation survey meter	Portable, hand-held ratemeter: 4 linear ranges, 0 to 0.1 to 0 to 100 rad/hr; visual readout	CM (portable)
Personal radiation dosimeter	1/crewmen; accumulated radiation dose: 0.01 to 1000 rad: visual readout	Suit
Passive radiation dosimeter	3/crewmen; emulsion/thermoluminescent dosimeters; postflight analysis	constant wear garment

Tabelle 6. Onboard Radiation Instrumentation, aus: [146, S. 111]

Nicht nur die Van-Allen-Strahlungsgürtel stellten eine potentielle Strahlengefahr dar, sondern auch die kosmische Strahlung generell und Sonneneruptionen. Dies ist sicher leicht nachvollziehbar, doch weniger bekannt sein dürfte, dass es an Bord des Apollo-Raumschiffs auch menschengemachte Strahlungsquellen gab. Dabei handelte es sich z.B. um radiolumineszente Displays. Um deren Strahlung zu reduzieren, wurden diese Displays mit Kunststoffschichten überzogen [146, S. 113]. Dieses Beispiel zeigt aber auch, wie die Gefahr einer Verstrahlung bei

zukünftigen bemannten Missionen reduziert werden kann: Durch den Verzicht auf radioaktive Materialien, sei es nun für den Antrieb, für die Stromversorgung oder für die sonstige Ausstattung.

Während der Apollo-Missionen war aber auch die Gesamtstrahlenbelastung für die Astronauten nicht gefährlich hoch. Tabelle 7 zeigt die durchschnittlichen Werte der einzelnen Missionen. Die Strahlendosis, die die Besatzung während der Apollo-Flüge aufnahm, war wesentlich geringer als der zu dieser Zeit geltende jährliche Durchschnitt, der von der U.S. Atomic Energy Commission für Personen, die mit radioaktiven Materialien umgehen, festgelegt worden war [146, S. 112].

Apollo Mission	Skin Dose, rads
7	0.16
8	.16
9	.20
10	.48
11	.18
12	.58
13	.24
14	1.14
15	.30
16	.51
17	.55

Tabelle 7. Average Radiation Doses of the Flight Crews for the Apollo Missions, aus: [146, S. 112]

Doch beeinflusste diese Strahlung die Bordcomputer und die Kommunikation zur Erde? Was die Kommunikation betrifft, so helfen auch hier die Gesprächstranskripte des „Apollo Flight Journal“ weiter [100]. Sieht man sich z.B. die Transkripte für die ersten Stunden nach TLI von Apollo 11 an, also der Zeit, in der das Raumschiff den Bereich der Van-Allen-Strahlungsgürtel durchflog, so sieht man, dass es kaum Kommunikationsschwierigkeiten gab. Und die, die doch auftraten, wurden z.B. dadurch verursacht, dass aufgrund der jeweiligen Lage des Raumschiffs in Bezug zu den Bodenstationen auf eine andere Empfangsstation umgeschaltet werden musste [95], [97].

Was war nun aber mit dem Computer? Der Speicher des Apollo Guidance Computers bestand, wie im Abschnitt 7.3 dargelegt, aus Core Rope Memory, also im Wesentlichen aus Eisenringen und Drähten. Dies machte den Speicher sehr robust gegenüber Strahlung. Doch traf dies auch auf die integrierten Schaltkreise zu?

Der AGC verwendete Resistor–Transistor Logic [53, Kapitel 12]. Die Verbindungsstellen der Bauteile sind dabei größer als bei moderneren Techniken wie z.B. CMOS. Dieses „mehr an Größe“ bot in Bezug auf die Strahlung den Vorteil, dass mehr Energie nötig war um ein Bauteil zu schädigen, als es bei stärker miniaturisierten Teilen der Fall gewesen wäre.

11 Analysen der Software des AGC

Bis heute ranken sich immer noch Verschwörungstheorien um die Mondlandungen. Kritiker stellen dabei auch die Leistungsfähigkeit der Navigationscomputer in Frage und vergleichen den AGC z.B. mit modernen Mobiltelefonen. Solche Vergleiche sind schon deshalb hinkend, weil der AGC zwar ein multifunktionales Gerät war, dennoch aber nicht auf ein so breites Funktionsspektrum ausgelegt war wie moderne Allzweckrechner (oder eben heutige Mobiltelefone, sogenannte „Smartphones“). Der AGC hatte sein Anwendungsgebiet im Bereich der Weltraumnavigation; er musste keine Filme abspielen oder CDs brennen können.

Es bringt daher nicht viel, die Rechengeschwindigkeit oder Speichergröße des AGC mit denjenigen Größen moderner Allzweckrechner zu vergleichen. Man sollte deshalb berücksichtigen, *was* man eigentlich vergleichen will. Der AGC lässt sich eher mit modernen Steuercomputern vergleichen, wie sie z.B. in industriellen Fertigungsanlagen eingesetzt werden. Ebenso lassen sich Vergleiche mit Navigationsgeräten aus dem Automobilbereich anstellen.

Was lässt sich nun durch Vergleiche und moderne Methoden der Softwareanalytik über den Apollo Guidance Computer ermitteln?

11.1 Vergleichsmöglichkeiten

Ein modernes GPS-Navigationsgerät verfügt meistens bereits über wesentlich mehr Speicher als der AGC. So hat z.B. bereits das Gerät „TomTom GO 61“, das die Herstellerfirma als Einsteigermodell bezeichnet, einen internen Speicher von 8 GB und einen Slot für eine Micro-SD-Karte [20]. Dieser umfangreiche Speicher dient zur Aufnahme von Kartenmaterial, in dem tausende Orte mit zahlreichen Details enthalten sind. Demgegenüber kam der AGC mit 41 Positionsmarken, der Star List, aus (siehe Anhang, Abschnitt 16.1). Hier fällt wieder der Unterschied auf: Die TomTom-Geräte sind zur allgemeinen Navigation von einem beliebigen Ort zu einem anderen beliebigen Ort gedacht, der AGC war für die Navigation von der Erde zum Mond und zurück gedacht. Wenn auch die Navigation im Weltall komplex ist, so gab es doch festgelegte Ziele. Die Entwickler bei TomTom wissen hingegen nicht, wohin morgen ein Kunde, der heute ein Gerät erworben hat, fahren wird. Darüber hinaus haben zahlreiche moderne Navigationsgeräte sowohl Sprachausgabefunktionen als auch eine Sprachsteuerung. Ist eine Sprachsteuerung hierbei nützlich? Ja, natürlich. Ist sie aber auch notwendig für die Navigation? Nein, natürlich nicht.

Hier soll keineswegs für Minimalismus geworben werden, auch den Astronauten hätte es sicher die Arbeit erleichtert, hätten sie mit dem AGC reden können wie die Astronauten in dem Film „2001: A Space Odyssey“ mit HAL 900. Aber mit den technischen Möglichkeiten des Apollo-Programms wäre ein solcher Computer weder zu bauen gewesen, noch hätte er in ein Raumschiff von der Größe des Apollo Command Modules gepasst. (Und er würde auch heute noch nicht hineinpassen. Der einzige Computer, der zumindest in Richtung HAL 9000 geht, dürfte der IBM Watson sein. Und selbst von diesem sagte Chefentwickler David Ferrucci in Bezug auf HAL 9000: „We’re not even close to that.“[34]). Doch es muss ja eben auch kein solcher Supercomputer sein. Noch nicht einmal ein TomTom mit seinem umfangreichen Kartenmaterial ist nötig, wenn Strecke und Reiseverlauf gut geplant sind. Auch nicht bei einer Reise zum Mond.

Geht man von der Hardware- auf die Softwareebene, so wird das Spektrum möglicher Vergleichskandidaten und Analysemethoden noch größer, da sich hier Aspekte wie Programmstruktur, Größe, Anzahl an Verzweigungen, etc. untersuchen lassen. Zwar ist auch die Größe oder Komplexität eines Programms vom geplanten Verwendungszweck des jeweiligen Programms abhängig, jedoch gibt es grundlegende Richtlinien bzw. Orientierungen, die bei der Softwareentwicklung berücksichtigt werden sollten um z.B. die Wartbarkeit der Programme zu erleichtern. So sollten Programme z.B. vernünftig kommentiert sein und Variablen mit aussagekräftigen Namen versehen sein. Dies sind Aspekte, die in modernen Lehrgängen zum Thema Softwareentwicklung zum „Grundgerüst“ gehören.

Der Bereich der Informatik, der sich mit der Analyse von Programmen in Bezug auf solche Kriterien befasst, ist die Software-Metrik (auch Softwaremetrie genannt).

Die wissenschaftliche Vermessung von Software begann etwa Ende der 1960er/Anfang der 1970er-Jahre [168, Kapitel 3], ein Zeitraum, der noch zur Apollo-Ära gehört. Und im Rahmen des Apollo-Programms wurden in der Tat auch Test- und Verifikationsverfahren für Software entwickelt. Denn in der bemannten Raumfahrt können schließlich Menschenleben von der Zuverlässigkeit der Computerprogramme abhängen. So wurde z.B. zu Beginn der 1970er-Jahre für den Program Development Branch der Mission Planning and Analysis Division des Manned Spacecraft Center das AUTOMATED VERIFICATION SYSTEM (AVS) entwickelt. AVS diente der Qualitätssicherung von Fortran-Programmen und hatte u.a. eine Komponente zur Analyse von Fortran-Code [63].

Doch auch außerhalb des Raumfahrtprogramms nahm in den 1960er- und 1970er-Jahren die Komplexität von Programmen stark zu. Daraus resultierte der Wunsch nach Methoden um diese zunehmende Komplexität in den Griff zu bekommen. Dies führte u.a. zu neuen Programmierkonzepten, zu neuen Ideen der Aufwandsabschätzung für Software-Projekte und zu Code-Messverfahren. In den Jahrzehnten nach Apollo gab es auf dem Gebiet der Software-Metrik dann auch zahlreiche neue Entwicklungen. Diese Vermessung von Programmen sollte auch dazu dienen, den Aufwand und die Fehleranfälligkeit von Änderungen bestehender Projekte näher spezifizieren zu können. Bedenkt man, wie umfangreich der weltweite Bestand von Legacy-Anwendungen ist, so erhält dieser Aspekt eine außerordentliche Bedeutung. Noch heute wird ein Großteil der Weltwirtschaft mit COBOL-Programmen ge-

steuert [155]. Sollen diese Programme angepasst oder geändert werden, so hilft hier die Software-Metrik um Aufwand und Umfang besser einschätzen zu können. Denken wir nur an die enormen Anstrengungen, die in den späten 1990er-Jahren unternommen wurden, um das „Jahr-2000-Problem“ in den Griff zu bekommen; es wird schnell klar, dass solche Änderungen kein abstraktes Szenario darstellen.

Wie lassen sich nun aber moderne Methoden der Software-Analyse auf den AGC-Quellcode anwenden? Dazu wird im Folgenden ein Überblick über Software-Messverfahren gegeben und einige Hauptverfahren der modernen Software-Metrik näher dargestellt. Außerdem werden jeweils Möglichkeiten erörtert, solche Methoden auf den AGC-Quellcode anzuwenden. Doch zunächst soll erst einmal klar gestellt werden, was überhaupt gemeint ist, wenn von „dem AGC-Quellcode“ die Rede ist.

11.2 AGC-Quellcode

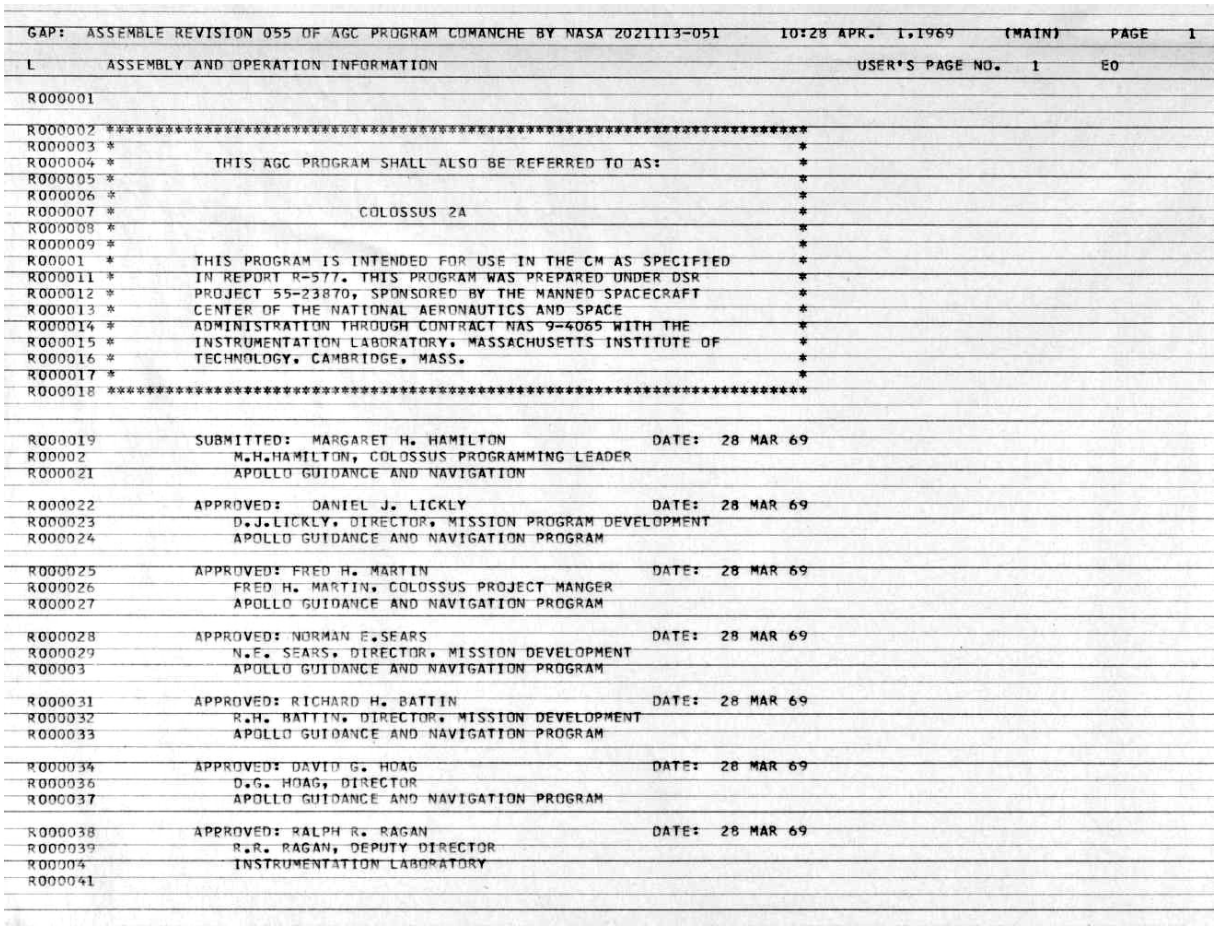
Nicht nur die Hardware des AGC durchlief mehrere Entwicklungsphasen (siehe Tabelle 1), auch die Software wurde ergänzt und verbessert. Anders als die Hardware wurde die Software des AGC während der gesamten Laufzeit des Apollo-Programms weiterentwickelt. Die dabei verwendeten Namen für die einzelnen Programmversionen wirken auf den ersten Blick möglicherweise etwas verwirrend, da es zahlreiche Unterversionen gab, die teilweise eigene Namen hatten. Die frühen Phasen der im Block I eingesetzten Software hatte Namen wie Sundisk, Solarium und Sunspot [162, S. 45].

Ab den bemannten Missionen wurden die Software-Versionen für den AGC im Command Module (CGC - Command Module Guidance Computer) dann Colossus 1 (Apollo 8-10), Colossus 2 bzw. Comanche (Apollo 11-14) und Colossus 3 (ab Apollo 15) genannt. In der Apollo-Sojus-Mission wurde die Version Skylark eingesetzt [58]. Die Software im LGC (Lunar Module Guidance Computer) wurde Luminary genannt [162, S. 45]. Natürlich gab es auch innerhalb einer Hauptversion zahlreiche Unterversionen. Im CM von Apollo 11 wurde z.B. die Revision 55 von Comanche (Colossus 2A) eingesetzt [54, S. 153]. Bei den Mondmissionen gab es also als grundlegendes Unterscheidungskriterium Namen mit „L“ am Anfang für die Software im Lunar Module und Namen mit „C“ am Anfang für die Software im Command Module. Abbildung 43 zeigt die Seite 1 des Listings der Colossus-Version aus Apollo 11. Der Name Comanche und die Revisionsnummer wurden auf sämtlichen Seiten wiederholt, die zu diesem Programm gehörten. Dabei sieht man auch, dass „Comanche“ kein Spitzname war, sondern auf den ausgedruckten Listings als offizielle Bezeichnung verwendet wird. Auf Seite 1 sieht man aber auch den zusätzlichen Hinweis:

„This AGC program shall also be referred to as Colossus 2A“ [78, S. 1]

Wenn man sich also fragt, welches nun der „offizielle“ Name war, so lautet die Antwort ganz einfach: Beide.

Am 20. Juli 2009, also 40 Jahre nachdem Neil Armstrong und Buzz Aldrin auf dem Mond gelandet waren, wurde der Quellcode des LGC und des CGC bei Google-Code veröffentlicht [83]. Später kamen nicht nur Software-Versionen weiterer Missionen hinzu sondern auch ein Assembler und eine virtuelle Maschine, in der die assemblierten Dateien ausgeführt werden können. Das Projekt, dass den Namen VirtualAGC trägt, wird inzwischen auf GitHub gehostet und enthält derzeit (Mai 2016) den transkribierten Code von Colossus 1 Build 237, Colossus 1 Build 249, Colossus 2A, Colossus 3, Luninary 1A (Apollo 11) und Luminary 1C (Apollo 13 und 14) [144]. Der Code wurde aus eingescannten Ausdrucken der AGC-Entwickler erstellt und in einzelne Module aufgeteilt. Der ursprüngliche Quellcode der AGC-Programme ist ein umfangreiches, wenig übersichtliches Listing, dennoch ist auch darin eine modulare Aufteilung zu finden; einzelne Programme sind als solche gekennzeichnet und von den jeweiligen anderen Programmen abgegrenzt. Den Entwicklern stand jedoch kein Linker zur Verfügung, daher wurde der gesamte Code (bzw. die entsprechenden Lochkarten) komplett assembliert.



```
GAP:  ASSEMBLE REVISION 055 OF AGC PROGRAM COMANCHE BY NASA 2021113-051      10:28 APR.  1,1969     (MAIN)     PAGE    1

L       ASSEMBLY AND OPERATION INFORMATION                                    USER'S PAGE NO.  1      EO

R000001

R000002 ************************************************************************
R000003 *                                                                      *
R000004 *          THIS AGC PROGRAM SHALL ALSO BE REFERRED TO AS:              *
R000005 *                                                                      *
R000006 *                                                                      *
R000007 *                          COLOSSUS 2A                                 *
R000008 *                                                                      *
R000009 *                                                                      *
R00001  *        THIS PROGRAM IS INTENDED FOR USE IN THE CM AS SPECIFIED       *
R000011 *        IN REPORT R-577. THIS PROGRAM WAS PREPARED UNDER DSR          *
R000012 *        PROJECT 55-23870, SPONSORED BY THE MANNED SPACECRAFT          *
R000013 *        CENTER OF THE NATIONAL AERONAUTICS AND SPACE                  *
R000014 *        ADMINISTRATION THROUGH CONTRACT NAS 9-4065 WITH THE           *
R000015 *        INSTRUMENTATION LABORATORY, MASSACHUSETTS INSTITUTE OF        *
R000016 *        TECHNOLOGY, CAMBRIDGE, MASS.                                  *
R000017 *                                                                      *
R000018 ************************************************************************

R000019          SUBMITTED:  MARGARET H. HAMILTON                DATE:  28 MAR 69
R00002               M.H.HAMILTON, COLOSSUS PROGRAMMING LEADER
R000021              APOLLO GUIDANCE AND NAVIGATION

R000022          APPROVED:   DANIEL J. LICKLY                    DATE:  28 MAR 69
R000023              D.J.LICKLY, DIRECTOR, MISSION PROGRAM DEVELOPMENT
R000024              APOLLO GUIDANCE AND NAVIGATION PROGRAM

R000025          APPROVED: FRED H. MARTIN                        DATE:  28 MAR 69
R000026              FRED H. MARTIN, COLOSSUS PROJECT MANGER
R000027              APOLLO GUIDANCE AND NAVIGATION PROGRAM

R000028          APPROVED: NORMAN E.SEARS                        DATE:  28 MAR 69
R000029              N.E. SEARS, DIRECTOR, MISSION DEVELOPMENT
R00003               APOLLO GUIDANCE AND NAVIGATION PROGRAM

R000031          APPROVED: RICHARD H. BATTIN                     DATE:  28 MAR 69
R000032              R.H. BATTIN, DIRECTOR, MISSION DEVELOPMENT
R000033              APOLLO GUIDANCE AND NAVIGATION PROGRAM

R000034          APPROVED: DAVID G. HOAG                         DATE:  28 MAR 69
R000036              D.G. HOAG, DIRECTOR
R000037              APOLLO GUIDANCE AND NAVIGATION PROGRAM

R000038          APPROVED: RALPH R. RAGAN                        DATE:  28 MAR 69
R000039              R.R. RAGAN, DEPUTY DIRECTOR
R00004               INSTRUMENTATION LABORATORY
R000041
```

Abbildung 43. Seite 1 des Colossus-Listings (nicht zu verwechseln mit den ebenfalls Colossus genannten Computern des Zweiten Weltkriegs) [78, S. 1]

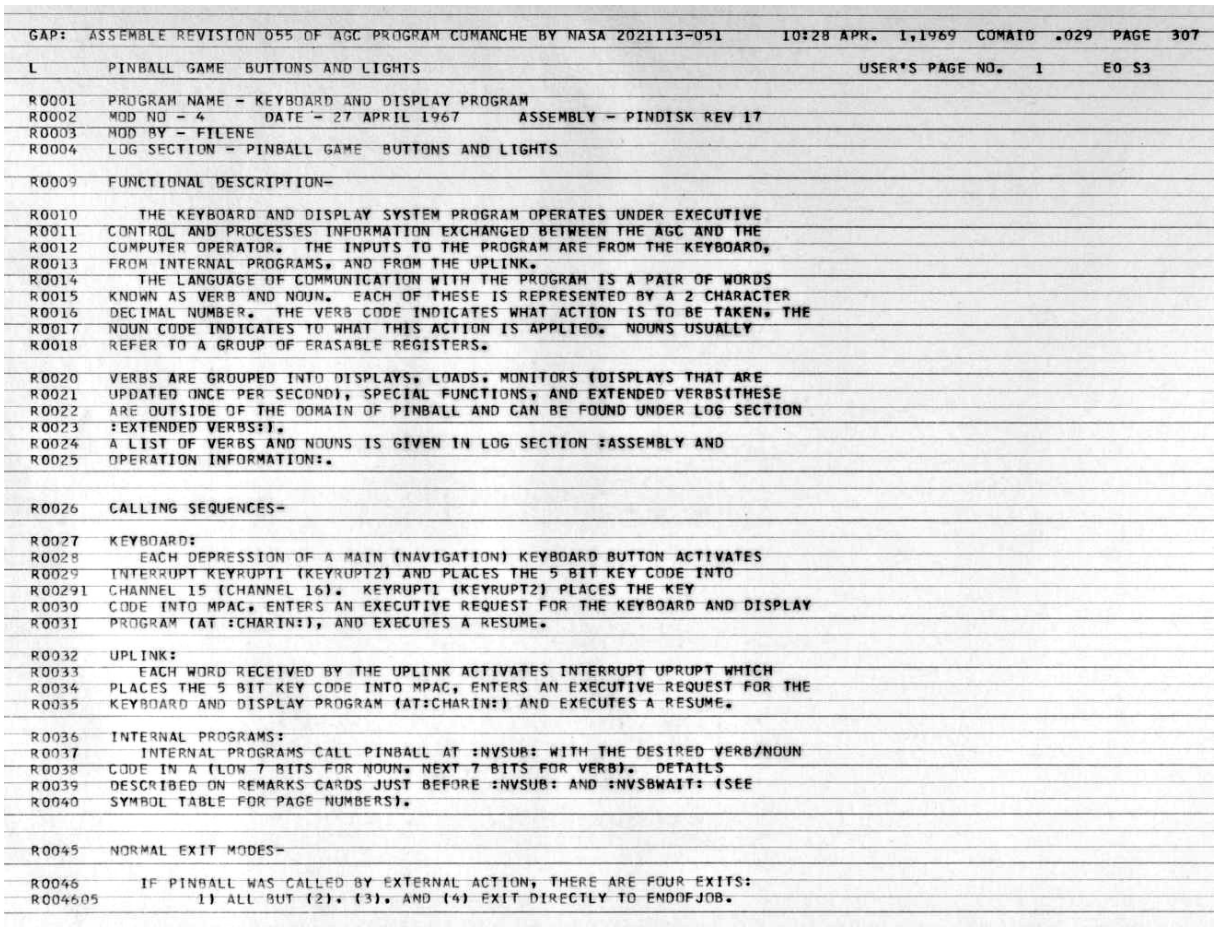
Die eingescannten Ausdrucke des Original-Codes stehen noch von mehreren AGC-Entwicklern zur Verfügung und werden in den folgenden Abschnitten daher auch für Überprüfungen herangezogen. Zwar müssen die aufbereiteten Dateien aus dem VirtualAGC-Projekt nicht nur überprüft, sondern ggf. auch überarbeitet werden, um den Originalen möglichst exakt zu entsprechen, die vorhandenen Dateien sind aber dennoch eine große Hilfe bei der Analyse und finden daher im Folgenden auch Verwendung.

11.3 Programme und Routinen im AGC-Quellcode

Der Quellcode von Colossus bzw. Luminary bildet zwar, wie gesagt, einen zusammenhängenden Block, dennoch ist der Code in einzelne, abgrenzbare Programme unterteilt. Die einzelnen Seiten des Listings enthalten in der Überschrift jeweils den Namen des Programms, zu dem die jeweilige Seite gehört. Abbildung 44 zeigt als Beispiel den Beginn des Programms PINBALL GAME BUTTONS AND LIGHTS. Der Programmname wird auf sämtlichen Seiten wiederholt, die zu diesem Programm gehören. Ebenso verhält es sich mit den weiteren Programmen, die das Listing bilden.

Im Anhang, in den Abschnitten 16.4 und 16.5, sind die Programme und Routinen aufgelistet, die bei Apollo 11 eingesetzt wurden. Dabei handelt es sich um Colossus Version 2A in der Revision 055 (also um Comanche 055) und um Luminary 1A in der Revision 099. Die Programme sind in der Reihenfolge wiedergegeben, wie sie im

Quellcode auftreten, mit Angabe der entsprechenden Seiten der Listings. Die Programme, die das Betriebssystem ausmachen, sind in beiden Versionen vorhanden, ebenso die Funktionen zur Ansteuerung des DSKY. Unterschiede sind z.B. die folgenden: Nur im Luminary-Code gibt es die Routinen zur Radarsteuerung und nur im Colossus-Code existiert das Modul „REENTRY CONTROL“ für den Wiedereintritt in die Erdatmosphäre.



```
GAP:  ASSEMBLE REVISION 055 OF AGC PROGRAM COMANCHE BY NASA 2021113-051      10:28 APR.  1,1969  COMAID  .029  PAGE  307

L       PINBALL GAME  BUTTONS AND LIGHTS                                    USER'S PAGE NO.   1      E0 S3

R0001   PROGRAM NAME - KEYBOARD AND DISPLAY PROGRAM
R0002   MOD NO - 4       DATE - 27 APRIL 1967       ASSEMBLY - PINDISK REV 17
R0003   MOD BY - FILENE
R0004   LOG SECTION - PINBALL GAME  BUTTONS AND LIGHTS

R0009   FUNCTIONAL DESCRIPTION-

R0010      THE KEYBOARD AND DISPLAY SYSTEM PROGRAM OPERATES UNDER EXECUTIVE
R0011   CONTROL AND PROCESSES INFORMATION EXCHANGED BETWEEN THE AGC AND THE
R0012   COMPUTER OPERATOR.  THE INPUTS TO THE PROGRAM ARE FROM THE KEYBOARD,
R0013   FROM INTERNAL PROGRAMS, AND FROM THE UPLINK.
R0014      THE LANGUAGE OF COMMUNICATION WITH THE PROGRAM IS A PAIR OF WORDS
R0015   KNOWN AS VERB AND NOUN.  EACH OF THESE IS REPRESENTED BY A 2 CHARACTER
R0016   DECIMAL NUMBER.  THE VERB CODE INDICATES WHAT ACTION IS TO BE TAKEN. THE
R0017   NOUN CODE INDICATES TO WHAT THIS ACTION IS APPLIED.  NOUNS USUALLY
R0018   REFER TO A GROUP OF ERASABLE REGISTERS.

R0020   VERBS ARE GROUPED INTO DISPLAYS, LOADS, MONITORS (DISPLAYS THAT ARE
R0021   UPDATED ONCE PER SECOND), SPECIAL FUNCTIONS, AND EXTENDED VERBS(THESE
R0022   ARE OUTSIDE OF THE DOMAIN OF PINBALL AND CAN BE FOUND UNDER LOG SECTION
R0023   :EXTENDED VERBS:).
R0024   A LIST OF VERBS AND NOUNS IS GIVEN IN LOG SECTION :ASSEMBLY AND
R0025   OPERATION INFORMATION:.

R0026   CALLING SEQUENCES-

R0027   KEYBOARD:
R0028      EACH DEPRESSION OF A MAIN (NAVIGATION) KEYBOARD BUTTON ACTIVATES
R0029   INTERRUPT KEYRUPT1 (KEYRUPT2) AND PLACES THE 5 BIT KEY CODE INTO
R00291  CHANNEL 15 (CHANNEL 16).  KEYRUPT1 (KEYRUPT2) PLACES THE KEY
R0030   CODE INTO MPAC, ENTERS AN EXECUTIVE REQUEST FOR THE KEYBOARD AND DISPLAY
R0031   PROGRAM (AT :CHARIN:), AND EXECUTES A RESUME.

R0032   UPLINK:
R0033      EACH WORD RECEIVED BY THE UPLINK ACTIVATES INTERRUPT UPRUPT WHICH
R0034   PLACES THE 5 BIT KEY CODE INTO MPAC, ENTERS AN EXECUTIVE REQUEST FOR THE
R0035   KEYBOARD AND DISPLAY PROGRAM (AT:CHARIN:) AND EXECUTES A RESUME.

R0036   INTERNAL PROGRAMS:
R0037      INTERNAL PROGRAMS CALL PINBALL AT :NVSUB: WITH THE DESIRED VERB/NOUN
R0038   CODE IN A (LOW 7 BITS FOR NOUN, NEXT 7 BITS FOR VERB).  DETAILS
R0039   DESCRIBED ON REMARKS CARDS JUST BEFORE :NVSUB: AND :NVSBWAIT: (SEE
R0040   SYMBOL TABLE FOR PAGE NUMBERS).

R0045   NORMAL EXIT MODES-

R0046      IF PINBALL WAS CALLED BY EXTERNAL ACTION, THERE ARE FOUR EXITS:
R004605          1) ALL BUT (2), (3), AND (4) EXIT DIRECTLY TO ENDOFJOB.
```

Abbildung 44. Seite 307 des Colossus-Listings [78, S. 307]

Die beiden AGC konnten sich durchaus gegenseitig ersetzen, wenn es darum ging, grundlegende Funktionalitäten aufrecht zu erhalten, was z.B. bei Apollo 13 wichtig wurde. Aber eine Mondlandung mit Colossus wäre nicht möglich gewesen. Ebenso war Luminary nicht für einen Wiedereintritt in die Erdatmosphäre gedacht; es ist allerdings anzunehmen, dass die NASA-Ingenieure im Falle einer nahenden Katastrophe hier auch noch Möglichkeiten gefunden hätten. Bei Apollo 13 gelang es ja zum Glück, den Computer des Command Modules für den Wiedereintritt zu reaktivieren.

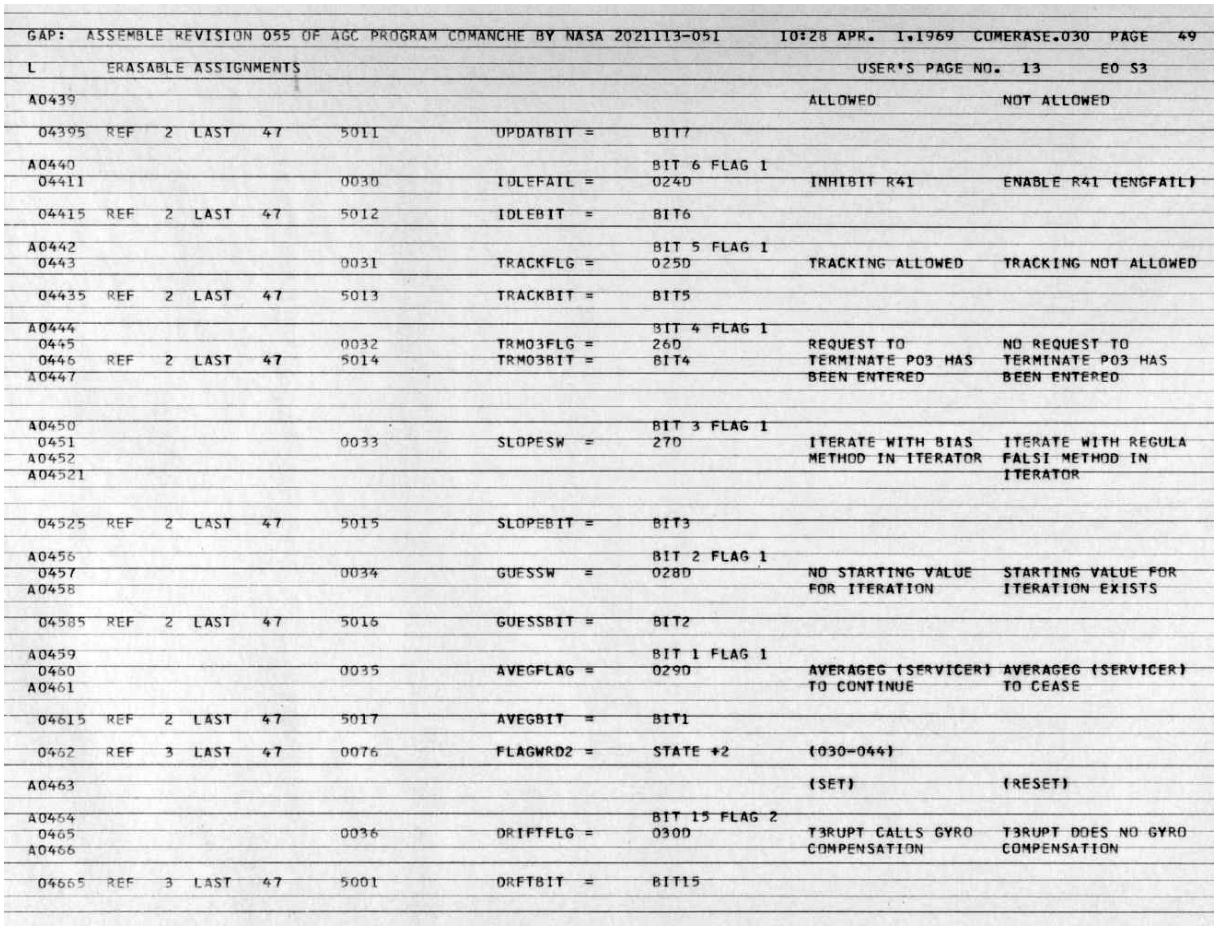
11.4 Die Sprache des AGC-Quellcodes

Die AGC-Programme sind in zwei unterschiedlichen Programmiersprachen geschrieben worden, einem Basis-Assembler (YUL- bzw. GAP) und einer höheren Sprache, dem Interpreter.

11.4.1 YUL-Assembler

Die erste Version des Assemblers, der später für den AGC eingesetzt wurde, wurde zu Weihnachten 1959 fertiggestellt. Aufgrund des Weihnachtsdatums wurde der Assembler YUL genannt. Der YUL-Assembler wurde in den Entwicklungsversionen und den ersten Flugversionen eingesetzt, noch vor der ersten bemannten Mondlandung jedoch durch eine erweiterte Version namens GAP (General Assembly Program) ersetzt.

Abbildung 45 zeigt ein weiteres Quellcode-Beispiel aus Comanche 055, wie er in Apollo 11 eingesetzt worden ist.



```
GAP:  ASSEMBLE REVISION 055 OF AGC PROGRAM COMANCHE BY NASA 2021113-051     10:28 APR.  1,1969  COMERASE.030  PAGE   49

L       ERASABLE ASSIGNMENTS                                                      USER'S PAGE NO.  13      EO S3

A0439                                                                     ALLOWED              NOT ALLOWED

  04395  REF   2  LAST   47      5011          UPDATBIT =      BIT7

A0440                                                          BIT 6 FLAG 1
  04411                        0030          IDLEFAIL =      024D          INHIBIT R41          ENABLE R41 (ENGFAIL)

  04415  REF   2  LAST   47      5012          IDLEBIT  =      BIT6

A0442                                                          BIT 5 FLAG 1
  0443                         0031          TRACKFLG =      025D          TRACKING ALLOWED     TRACKING NOT ALLOWED

  04435  REF   2  LAST   47      5013          TRACKBIT =      BIT5

A0444                                                          BIT 4 FLAG 1
  0445                         0032          TRM03FLG =      26D
  0446   REF   2  LAST   47      5014          TRM03BIT =      BIT4          REQUEST TO           NO REQUEST TO
A0447                                                        TERMINATE P03 HAS    TERMINATE P03 HAS
                                                             BEEN ENTERED         BEEN ENTERED

A0450                                                          BIT 3 FLAG 1
  0451                         0033          SLOPESW  =      27D           ITERATE WITH BIAS    ITERATE WITH REGULA
A0452                                                        METHOD IN ITERATOR   FALSI METHOD IN
A04521                                                                            ITERATOR

  04525  REF   2  LAST   47      5015          SLOPEBIT =      BIT3

A0456                                                          BIT 2 FLAG 1
  0457                         0034          GUESSW   =      028D          NO STARTING VALUE    STARTING VALUE FOR
A0458                                                        FOR ITERATION        ITERATION EXISTS

  04585  REF   2  LAST   47      5016          GUESSBIT =      BIT2

A0459                                                          BIT 1 FLAG 1
  0460                         0035          AVEGFLAG =      029D          AVERAGEG (SERVICER)  AVERAGEG (SERVICER)
A0461                                                        TO CONTINUE          TO CEASE

  04615  REF   2  LAST   47      5017          AVEGBIT  =      BIT1

  0462   REF   3  LAST   47      0076          FLAGWRD2 =      STATE +2      (030-044)

A0463                                                                     (SET)                (RESET)

A0464                                                          BIT 15 FLAG 2
  0465                         0036          DRIFTFLG =      030D          T3RUPT CALLS GYRO    T3RUPT DOES NO GYRO
A0466                                                        COMPENSATION         COMPENSATION

  04665  REF   3  LAST   47      5001          DRFTBIT  =      BIT15
```

Abbildung 45. Seite 49 des Colossus-Listings [78, S. 49]

11.4.2 Interpreter

Zusätzlich zu den Befehlen des YUL- bzw. GAP-Assemblers existierte noch ein weiterer Befehlssatz, der Interpreter. Dabei handelte es sich um Befehle, die durch eine Art virtuelle Maschine realisiert wurden. Die Interpreter-Instruktionen sind im AGC-Assembler geschrieben und bieten komplexere Funktionen als der AGC-Basisbefehlssatz an, z.B. trigonometrische- und Vektoroperationen. Bei den Interpreter-Befehlen zeigt sich deutlich die Herangehensweise, sich nicht daran zu orientieren, was eine bestimmte Maschine vermag, sondern daran, welche Aufgaben die Maschine übernehmen soll. Ingenieure stellten mathematische Modelle für die Navigation im Weltraum auf und leiteten daraus ab, welche mathematischen Funktionen vom Computer während

des Fluges zu berechnen waren. Für die Implementierung dieser Funktionen wurden nun bestimmte Befehle benötigt, wie z.B. ein Sinus-Befehl. Diese Befehle wurden dann schließlich in AGC-Assembler realisiert. Somit stellt der Interpreter eine stark für ihr Einsatzgebiet optimierte Sprachschicht dar, eine Domain Specific Language (DSL).

11.5 Software-Messverfahren

Wie in den Abschnitten 2.1.1 und 2.2 dargestellt, wurden in den 1960er-Jahren neue Betriebssysteme entwickelt, die den Umgang mit dem Computer gegenüber früheren Systemen erheblich erleichterten und neue Möglichkeiten, wie das Time-Sharing, boten. Diese Systeme waren komplex und umfangreich; selbst IBM hat das für den IBM 360 geplante Time-Sharing-System TSS/360 nicht in der ursprünglich vorgesehenen Zeit fertiggestellt. Erst im IBM 370 kam das Time-Sharing-System zum Einsatz. Um die zunehmende Komplexität der Software besser beherrschen zu können, wurden dann auch bereits Ende der 1960er-/Anfang der 1970er-Jahre verschiedene Konzepte zur Qualitätssicherung von Software-Projekten entwickelt. Hier ist insbesondere der Ansatz hervorzuheben, Ergebnisse der Messtheorie auf Software anzuwenden: die Softwaremetrie. Softwaremetrie bezieht sich auf die quantitative Erfassung von Eigenschaften von Software-Projekten und deren Interpretation. Bei einer solchen Definition lässt sich zwar einwenden, dass eine Vermessung auch ohne Interpretation gültige Resultate liefert, doch ohne Interpretation (die sowohl auf Grundlage eines Theoriegebäudes als auch auf Erfahrungswerten basierend erfolgen kann) sind Messergebnisse zunächst nur Zahlenwerte wie „13 km“, „15 Stunden“ oder „3,14159265“. Sind 15 Stunden nun viel oder wenig? Natürlich hängt das vom Kontext und von möglichen Vergleichswerten ab. Ebenso benötigt man in der Softwaremetrie nicht nur Zahlen, sondern auch Möglichkeiten der Einordnung und Vergleichswerte. Darüber hinaus müssen die Messwerte selbst auch auf Grundlage eines entsprechenden theoretischen Unterbaus zustande kommen. Misst man z.B. die Frequenz, mit der ein Softwareentwickler seine Hemden wechselt, so dürfte es schwierig sein, daraus Aussagen zur Qualität des Softwareprodukts zu erschließen, denn hier fehlt es am theoretischen Unterbau, der beide Aspekte vereint.

Wie aber lassen sich nun Software, bzw. Software-Projekte, vermessen? Für eine grundlegende Einteilung, in welcher „Richtung“ gemessen werden soll, lassen sich folgende Hauptbereiche spezifizieren:

- Quantität
- Komplexität
- Qualität

Wie lassen sich nun diese Kriterien messen? Insbesondere Komplexität und Qualität sind recht allgemeine Begriffe, wenn keine exakte Definition vorliegt. Es gibt daher verschiedene Definitionsmöglichkeiten und unterschiedliche Möglichkeiten, die Komplexität und die Qualität von Software zu messen, siehe hierzu z.B. [82] für einen Überblick und [170] für eine tiefer gehende Einführung in die Thematik. Die Quantität lässt sich zwar leichter messen, im einfachsten Fall als Anzahl Programmzeilen (LOC, „Lines of Code“). Dies kann bereits einen ersten Eindruck des möglichen Implementierungs- oder auch des Änderungsaufwands geben. Doch auch hier lässt sich detaillierter vorgehen, als einfach nur die Anzahl der gedruckten Zeilen zu zählen. Man kann z.B. unterscheiden zwischen Anweisungs- und Kommentarzeilen, ebenso kann man Zeilen pro Modul zählen und dergleichen mehr.

Ebenso wie für die zu messenden Kriterien, benötigt man auch Spezifizierungen für den zu vermessenden Gegenstand, damit Vergleiche überhaupt Sinn machen. Horst Zuse definiert in seinem Werk „A Framework of Software Measurement“ daher auch die Begriffe Programm und Modul [168, Kapitel 2]. Sowohl für Programm als auch für Modul lassen sich natürlich unterschiedliche Beschreibungen aufstellen. Für die nachfolgenden Darlegungen genügen folgende recht allgemeine Definitionen, um eine klare Vorstellung der beiden Begriffe zu haben:

- Modul: Eine eigenständige funktionale Software-Einheit
- Programm: Eine Abfolge von Deklarationen und Instruktionen zur Erreichung eines festgelegten Ziels; die Abfolge von Deklarationen und Instruktionen muss dabei den Regeln einer Programmiersprache folgen.

Hat man nun Definitionen für Kriterien und für die zu vermessenden Objekte, so kann man mit geeigneten Messverfahren den jeweiligen Code bzw. das jeweilige Software-Projekt untersuchen und erhält Zahlenwerte als Resultat.

Doch solche Werte ergeben natürlich noch kein aussagekräftiges Ergebnis; die ermittelten Zahlen müssen auch interpretiert werden. Horst Zuse weist darauf hin, dass es eines der größten Probleme auf dem Gebiet der Software-Metriken ist, dass nur wenig über die *Bedeutung* der ermittelten Zahlen gesprochen wird:

„One of the major problems in the area of software measurement is the lack of discussing the meaning of the numbers produced by measures." [168, S. 42]

Daher ist die Interpretation ermittelter Werte von erheblicher Bedeutung, will man aussagekräftige Resultate erzielen oder Vergleiche mit anderen Software-Projekten durchführen.

Und was sind nun „geeignete" Messverfahren? Um dies zu beantworten, werden zunächst einige der weit verbreitetsten Code-Metriken vorgestellt. Dabei wird auch jeweils auf die Besonderheiten von Messungen in Bezug auf den AGC-Code eingegangen.

11.5.1 Komplexität nach Halstead

Maurice Halstead hat 1977 Metriken zur Vermessung von Programmen aufgestellt, die auf der Unterscheidung von Operatoren (Befehlen) und Operanden (Argumenten) beruhen [82, S. 163] [47]. Da keine weitergehenden Kriterien als diese Unterscheidbarkeit (und die Zählbarkeit der auftretenden Operanden/Operatoren) erforderlich sind, lassen sich die Halstead-Metriken auf nahezu jede Programmiersprache anwenden.

Gemäß Baumgartner und Sneed errechnet sich die Halstead-Komplexität folgendermaßen [82, S. 163]:

$$\text{Komplexität nach Halstead} = \frac{Operatoren}{Operatorenvorkommnisse} \cdot \frac{Operanden}{Operandenvorkommnisse}$$

Die Unterscheidbarkeit und Zählbarkeit von Operatoren und Operanden sind dabei auch die einzigen Bedingungen, die erfüllt sein müssen, um die Halstead-Metriken anwenden zu können. Dabei sind die Elemente der Metriken die folgenden:

$$\begin{aligned}
\eta_1 &= \text{Anzahl unterschiedlicher Operatoren (Befehle)} \\
\eta_2 &= \text{Anzahl unterschiedlicher Operanden (Argumente)} \\
N_1 &= \text{Gesamtanzahl Operatoren (Befehle)} \\
N_2 &= \text{Gesamtanzahl Operanden (Argumente)}
\end{aligned}$$

Aus diesen Elementen lassen sich nun folgende Halstead-Metriken aufstellen:

$$\begin{aligned}
\text{Programmvokabular } \eta &= \eta_1 + \eta_2 \\
\text{Programmlänge } N &= N_1 + N_2 \\
\text{Volumen } V &= N \cdot log_2 \eta \\
\text{Schwierigkeit } D &= \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2} \\
\text{Aufwand } E &= D \cdot V
\end{aligned}$$

Die Halstead-Metriken wurden von verschiedenen Wissenschaftlern dahingehend kritisiert, dass sie keine realistische Abschätzung für Aufwand oder Fehler liefern würden, siehe z.B. [150, Kap. 11]

Wie aussagekräftig sind nun also die Werte, die sich nach Halstead ergeben? Die Halstead-Metriken lassen sich durchaus gut einsetzen für Aufwandsabschätzungen oder auch für Vergleiche. Allerdings muss man schon genau überlegen, *was womit* verglichen werden soll. Die Werte für N steigen linear mit der Größe des zu untersuchenden

Programms an, aber nicht die Werte für η. Horst Zuse hat gezeigt, dass letztere zu einer Konstanten konvergieren [169], was leicht nachvollziehbar ist: Nehmen wir ein einzeiliges Programm und fügen sukzessive weitere Programmzeilen hinzu, so wird irgendwann der Punkt erreicht sein, an dem nahezu alle Befehlswörter und Argumente bereits verwendet worden sind und somit keine weitere Erhöhung von η mehr stattfindet (oder zumindest nur noch eine äußerst geringe Erhöhung im Verhältnis zum linear anwachsenden N), die Programmgröße hat also sozusagen eine „kritische Masse“. Daher ist es für Vergleiche wichtig, die Programmgröße zu berücksichtigen und entsprechend ausreichend große Programme zu betrachten. Zwar ist „ausreichend groß“ keine sehr konkrete Festlegung, da jedoch nach obiger Feststellung die „kritische Größe“ dann erreicht ist, wenn (nahezu) alle Befehlswörter und Argumente Verwendung finden, ist dieses Größenkriterium beim AGC-Code erfüllt, denn wenn man den kompletten Code aus dem AGC im Lunar Module (LGC) oder auch aus dem Command Module (CGC) betrachtet, so treten darin jeweils nahezu alle Befehle auf. Die Einschränkung „nahezu“ kommt daher, dass einige Befehle nur im Lunar Module bzw. nur im Command Module auftreten. Bei den Interpreter-Instruktionen ist die Erfüllung des Größenkriteriums dabei noch deutlicher erkennbar als bei den Befehlen des YUL- bzw. GAP-Assemblers, da der Interpreter-Befehlssatz ja auf die Lösung bestimmter Navigationsprobleme hin entworfen worden ist. Der AGC-Quellcode ist auch sehr umfangreich; die Ausdrucke des Colossus- und des Luminary-Codes umfassen jeweils mehr als 1700 Seiten. Bei einem Vergleich mit anderen Programmen mittels der Halstead-Metriken muss aber eben auch der Umfang dieser anderen Programme entsprechend berücksichtigt werden, um sinnvolle Aussagen treffen zu können.

11.5.2 Zyklomatische Komplexität nach McCabe

Bereits 1976 hat Thomas J. McCabe eine Software-Metrik beschrieben, die die Anzahl der Verzweigungen eines Programmes oder Moduls misst [82, S. 164] [84] [170, Kap. 4].

Bei Anwendung der McCabe-Metrik auf den AGC-Assembler ist jedoch zu beachten, dass dieser eine hohe Anzahl an Sprunganweisungen enthält (s. a. 11.6.1), was bei Vergleichen mit McCabe-Analysen von Hochsprachen zu berücksichtigen ist. Der Interpreter-Code lässt sich schon eher mit Sprachen wie C Vergleichen, da dieser eine Sprache darstellt, die bereits zwischen Assembler und Hochsprache einzustufen ist. Vergleicht man mittels der McCabe-Metrik direkt Assemblerprogramme mit C-Programmen, so wird man meist eine höhere Komplexität bei den Assemblerprogrammen erhalten. Das ist auch naheliegend, denn einer der Gründe, weshalb Hochsprachen überhaupt entwickelt worden sind, ist es ja, die zunehmende Komplexität umfangreicher Systeme handhabbar zu machen. Und dass ein Programm in C oder in Fortran bereits bei oberflächlicher Betrachtung übersichtlicher aussieht als ein Assemblerprogramm gleicher Funktion, dem dürften wohl die meisten Programmierer zustimmen (zumindest wenn grundlegende Regeln zu Programmstruktur, Quellcodeformatierung und Variablennamen beachtet werden).

11.5.3 Verständlichkeit

Die Verständlichkeit lässt sich folgendermaßen ermitteln [82, S. 174]

$$\text{Verständlichkeit} = \frac{Kommentarzeilen}{alleZeilen} \cdot \frac{Bausteine}{Anweisungen} \cdot \frac{sprechendeNamen}{alleNamen}$$

Der AGC-Code ist durchaus gut kommentiert, wirkt aber auf den ersten Blick dennoch schwer verständlich. In einer Zeit ohne Versionsverwaltungssysteme und E-Mail dienten die Kommentare natürlich auch dem Austausch unter den Entwicklern, und für diese waren die Kommentare auch verständlich. Und befasst man sich etwas näher mit den Kommentaren, so merkt man schnell, dass diese auch aus heutiger Sicht gute Erklärungen des jeweiligen Programms bzw. Programmabschnitts bieten. Sprechende Namen sind schon schwieriger in der Assemblersprache. Es wurden kurze, aber durchaus treffende Bezeichnungen verwendet. Auch hier muss berücksichtigt werden, dass auch diese Bezeichnungen nicht nur aus dem Bereich Mathematik und Programmierung sondern auch aus der Raumfahrt stammen. Daher finden wir z.B. Bezeichnungen wie „IGNITION“, „DURATION“, „ROLLFIRE“ aber auch „DSPLAY“„COSINE“ oder „SINE“.

11.5.4 Sprungweiten

Mit einer Analyse der Sprunganweisungen lässt sich die Weite (z.B. in Codezeilen) ermitteln, über die ein Sprung erfolgt. Sprünge über große Entfernungen können den Code unübersichtlicher machen. Dabei ist jedoch anzumerken, dass der Aufruf eines Unterprogrammes mittels Sprungbefehl in einer Assemblersprache durchaus zu hohen Sprungweiten führen kann, der Code jedoch durch die Aufteilung in Unterprogramme dennoch gut strukturiert sein kann.

11.5.5 Geeignete Code-Metriken für den AGC-Code

Von den dargestellten Metriken lassen sich einige durchaus gut auf den AGC-Code anwenden, z.B. die Komplexität nach Halstead, da diese ja zum einen sehr universell einsetzbar ist und zum anderen auch aussagekräftige Resultate liefert, wenn das zu untersuchende Programm umfangreich genug ist.

Eine Untersuchung auf Portierbarkeit macht wenig Sinn, da das System ja gar nicht auf Portierbarkeit ausgelegt war.
Qualitätsmessungen sind problemlos möglich, z.B. Messungen der Verständlichkeit. Bei der Verständlichkeitsmesung des AGC-Codes müssen allerdings bereits Einschränkungen in Kauf genommen werden, da z.B. die begrenzte Länge der Bezeichner die Verwendung aussagekräftiger Namen einschränkt.
Komplexitätsmessungen, die sich auf die Verschachtelungstiefe beziehen (Verschachtelungskomplexität), sind ebenfalls geeignet, allerdings muss hierbei wiederum berücksichtigt werden, dass bei Assemblersprachen Sprunganweisungen häufiger zu erwarten sind als bei Hochsprachen.

Wichtig ist festzuhalten, dass aufgrund der Unterschiedlichkeit des AGC-Codes in Bezug auf moderne Projekte und der Verwendung unterschiedlicher Messverfahren keine einzelne Zahl als Ergebnis zu erwarten ist, sondern vielmehr ein Vergleich einzelner Aspekte/Kriterien sowie Einschätzungen aufgrund der vorhandenen Dokumentationen zum AGC.

11.6 Werkzeuge zur Anwendung von Software-Metriken

Die Berechnung der vorgestellten Metriken lässt sich durchaus „von Hand“ durchführen, in dem z.B. in den Code-Listings die einzelnen Instruktionsarten gezählt werden. Allerdings stößt eine manuelle Vorgehensweise schnell an ihre Grenzen, sowohl was Umfang als auch Fehleranfälligkeit betrifft: Bei größeren Programmen ist es zum einen ein langwieriges Unterfangen, den Code zu analysieren, und zum anderen besteht die Gefahr nachlassender Konzentration, was z.B. zu Verwechslungen bei ähnlich benannten Anweisungen führen kann. Da Software-Analysen außerdem meist aus dem wiederholten Anwenden gewisser Grundregeln bestehen, bietet sich eine Automatisierung an. Dementsprechend gab es auch – wie bereits in Abschnitt 11.1 erwähnt – bereits Anfang der 1970er-Jahre Software-Systeme zur automatisierten Programmverifikation und Qualitätskontrolle. Heutzutage existiert ein breiter Markt an Produkten zur Software-Qualitätskontrolle. In den folgenden Abschnitten werden einige davon – vorzugsweise Open-Source-Projekte – vorgestellt und deren Anwendbarkeit auf den AGC-Code beschrieben.

11.6.1 Quantitätsanalysen

Zur Messung der Menge dient nicht einfach nur das Zählen der Zeilen oder Bytes. Mehr Informationen erhält man bereits, wenn man die Anzahl an Befehlszeilen nimmt. Diese kann man auch mit der Anzahl an Kommentaren und Leerzeilen vergleichen. Für solche Messungen existieren zahlreiche Tools, von denen die meisten zwar auf modernere Programmiersprachen zugeschnitten sind, jedoch auch in Bezug auf den AGC-Code brauchbare Ergebnisse liefern können. Das Programm OHCount z.B. unterstützt zahlreiche Programmiersprachen, auch verschiedene Assemblerversionen [165]. OHCount stammt ursprünglich von den Ohloh Labs, einem 2004 gegründetem Unternehmen, das Analyseprodukte für Open-Source-Projekte herstellt und heute unter „Black Duck Open Hub“

weiterhin umfangreiche Analysen von Open-Source-Projekten anbietet [12] – Informationen über das ursprüngliche Unternehmen Ohloh lassen sich über das Internetarchiv aufrufen, siehe z.B. [8]. OHCount eignet sich auch für eine quantitative Analyse des AGC-Codes. Hierfür ist allerdings zu beachten, dass der Originalcode des AGC kein spezielles Zeichen zur Signalisierung von Kommentaren vorsieht; dementsprechend müsste also entweder eines definiert und gesetzt werden oder auf Informationen zu Kommentaren verzichtet werden. Ein Verzicht auf Kommentare würde aber einen Verlust eines Großteils der Informationen, die OHCount liefern kann, bedeuten. Daher ist es sinnvoller, Kommentarzeichen einzufügen.

11.6.2 Häufigkeitsanalysen

Eine erste Häufigkeitsanalyse einzelner Anweisungen lässt sich mit bereits mit UNIX/Linux-Tools wie grep (globally search a regular expression and print) und wc (word count) durchführen. Dafür kann ein Shell-Skript z.B. folgendermaßen aufgebaut sein:

```
#! /bin/bash
count=0
linecount=0
if [ -z ${2+x} ] || [ -z ${1+x} ]
then
  echo "usage: $0 [FILENAME] [WORDLIST]"
  exit
fi
while read line
do
  array+=($line)
done < $2
echo
echo "Statistics for $1:"
echo -e "Instruction \tCount "
echo     "----------------------------"
for i in "${array[@]}"
do
  searchword=$'[[:cntrl:]]{1,10}'$i$'[[:cntrl:]]{1,10}'
  foundword=$(egrep $searchword $1)
  count=$(egrep $searchword $1 | wc -l);
  echo -e "$i\t\t $count\t"
done
echo     "----------------------------"
linecount=$(cat $1 | wc -l)
echo " Line count is: $linecount";
echo     "----------------------------"
```

Hierbei ist [FILENAME] der Name der zu untersuchenden Datei und [WORDLIST] die Liste mit den AGC-Instruktionen, nach denen gesucht werden soll.

11.6.3 Komplexitätsanalysen

Es existieren zahlreiche kommerzielle Programmpakte zur Durchführung von Komplexitätsanalysen. Unternehmen wie Verisoft [164] oder Testwell [160] haben sich auf die Software-Qualitätssicherung spezialisiert und bieten dementsprechend auch verschiedene Programme zur Komplexitätsanalyse an.

Doch auch in der Open-Source-Community ist das Thema Software-Qualitätssicherung (sowohl mit als auch ohne kommerzielle Interessen) keineswegs unbedeutend. Hier gibt es ebenfalls zahlreiche Programme zur Durchführung von Komplexitätsanalysen, so z.B. „pmccabe“, ein C-Projekt [9], „metrix++“, ein Python-Projekt für C/C++/C# und Java [76], oder auch das „Halstead Metrics Tool“, ein Java-Projekt [142]. Das Python-Projekt „Commented Code Detector“ [17] zur Ermittlung von auskommentiertem Code eignet sich ebenfalls zur Analyse, da es den auskommentierten Code mittels Halstead-Metriken ermittelt.

Anders als bei Quantitätsmessungen ist es bei Komplexitätsmessungen jedoch nötig, die Struktur des zu untersuchenden Programms zu erfassen. Daher sind Werkzeuge zur Analyse von Komplexitäten zumeist auf bestimmte Programmiersprachen ausgelegt, wie obige Aufzählung einiger Werkzeuge bereits erkennen lässt. Auch Analyseprogramme, die mit unterschiedlichen Sprachen umgehen können, enthalten entsprechende Struktur- oder Parserinformationen über einzelne Programmiersprachen. Für den AGC-Code scheint keines der verbreiteten Analysewerkzeuge per se geeignet zu sein. Um den AGC-Code analysieren zu können, ergeben sich daher folgende Möglichkeiten:

1. Ein eigenes Programm schreiben.
2. Den AGC-Code auf andere Sprachen zu übertragen.
3. Ein vorhandenes Werkzeug derart zu konfigurieren bzw. zu modifizieren, so dass sich damit der AGC-Code analysieren lässt.

Für die folgenden Analysen werden alle drei Möglichkeiten genutzt. In Abschnitt 11.8.4 wird ein eigenes Programm zur Analyse des AGC-Codes nach Halstead verwendet. Zur Anwendung der McCabe-Metriken, die ja die Komplexität des Kontrollflusses im Code anzeigen, eignet sich die Übertragung des Codes auf andere Programmiersprachen, da hier lediglich der Kontrollfluss der Programme abgebildet werden muss. Zur Ermittlung der McCabe-Metriken wird daher neben einem selbst erstellten Programm auch pmccabe eingesetzt. Das Programm pmccabe zählt die Anzahl der Funktionen und die bedingten Sprung- bzw. Verzweigungsanweisungen innerhalb der Funktionen. Für die Berechnung der McCabe-Werte erhält eine Funktion zunächst den Wert 1 (für die Funktion selbst); dann wird die Anzahl der Entscheidungen in der jeweilige Funktion aufaddiert. Das Programm pmccabe liefert zwei verschiedene Ergebnisse für die McCabe-Komplexität: „Modified McCabe“ und „Traditional McCabe“. Bei „Modified McCabe“ werden die einzelnen Case-Anweisungen in einer Switch-Anweisung nicht für die Berechnung der Komplexität aufaddiert, erhöhen diese daher also nicht. Bei „Traditional McCabe“ werden demgegenüber die einzelnen case-Anweisungen aufaddiert und führen somit zu einer Erhöhung der Komplexität [9]. Darüber hinaus liefert pmccabe die Anzahl an Zeilen für jede Funktion.

Das Prinzip, nach dem pmccabe arbeitet, wird auch in anderen Projekten verwendet. Ein Beispiel hierfür ist der „Project Analyzer“ des finnischen Unternehmens Aivosto [4]. Dieses Prinzip lässt sich leicht auf den AGC-Code übertragen. Im Anhang, Abschnitt 16.8, ist ein entsprechendes Programm wiedergegeben. Dieses Programm, McCabe_AGC, geht folgendermaßen vor: Eine AGC-Quellcodedatei wird eingelesen und die darin enthaltenen Funktionen werden gezählt. Für jede dieser Funktionen wird der Code auf das Vorhandensein der folgenden Sprunganweisungen untersucht:

TC	Transfer Control
TCF	Transfer Control to Fixed
BZMF	Branch on Zero or Minus to Fixed
CCS	Count, Compare and Skip

Die CCS-Anweisung bietet dabei eine vierfache Fallunterscheidung: >0, <0, =-0, =+0 (Zwei Werte für 0 sind vorhanden, da der AGC im Einerkomplement rechnet). Aufgrund der Fallunterscheidungen wird daher die Anzahl der CCS-Anweisungen für die Berechnung der McCabe-Werte mit 4 multipliziert. TC und TCF werden zwar gezählt und ausgegeben, fließen aber nicht in die Berechnung der McCabe-Werte ein, da es sich bei diesen Anweisungen um unbedingte Sprünge handelt. Die Sprungmarken wiederum fließen mit 1 in die Berechnung ein, da es sich hierbei um Einsprungpunkte handelt, die somit als Beginn einer funktionalen Einheit bzw. Funktion angesehen werden

können. Die übrigen Sprunganweisungen fließen jeweils mit 1 in die Berechnung ein. Anhand der Sprungmarken werden die ermittelten Werte den jeweiligen Funktionen zugeordnet.

Das gleiche Verfahren lässt sich natürlich auch auf modernen Assembler-Code anwenden. Hier sind anstelle der AGC-Sprunganweisungen die Sprunganweisungen des x68-Assemblers (bzw. des zu analysierenden Assemblers) zu untersuchen. Ebenfalls im Anhang, in Abschnitt 16.9, ist der Quelltext des Programms McCabe_Assembler wiedergegeben. Dieses Programm ermittelt die McCabe-Werte entsprechend der Vorgehensweise von pmccabe und McCabe_AGC, jedoch für (x86-) Assembler. Die untersuchten Sprunganweisungen sind hierbei:

call	Sprung mit Rücksprungadresse
jmp	unbedingter Sprung
js	Jump if sign
jns	Jump if not sign
je	Jump if equal
jz	Jump if zero
jne	Jump if not equal
jnz	Jump if not zero
jb	Jump if below
jc	Jump if carry
jnae	Jump if above or equal
jnb	Jump if not below
jae	Jump if above or equal
jnc	Jump if not carry
jbe	Jump if below or equal
jna	Jump if not above
ja	Jump if above
jnbe	Jump if not below or equal
jl	Jump if less
jnge	Jump if not greater or equal
jge	Jump if greater or equal
jnl	Jump if not less
jle	Jump if less or equal
jng	Jump if not greater
jg	Jump if greater
jnle	Jump if not less or equal
jecxz	Jump if %ECX register is 0
jo	Jump if overflow
jp	Jump if parity
jpe	Jump if parity even
jpo	Jump if parity odd
jcxz	Jump if %CX register is 0

Die Jump-Anweisungen werden dabei auch in ihren Varianten „jmpl“, „jmpw“, etc berücksichtigt. Auch hier werden die unbedingten Sprünge für Vergleiche gezählt und ausgegeben, fließen jedoch nicht in die Berechnung der McCabe-Werte ein.

Als modernes Vergleichsprojekt wird in dieser Arbeit der Linux-Kernel herangezogen (Kernel 3.16.7-29, siehe Abschnitt 11.11). Die Linux-Kernelquellen bestehen nicht nur aus C-Dateien, sondern auch Assembler-Dateien (x86, m68k, arm, etc.). Mit McCabe_Assembler lassen sich die (x86-) Assembler-Quelltexte nach dem gleichen Algorithmus analysieren, mit dem auch die AGC-Quellcodedateien analysiert werden. Daher erhält man hier Ergebnisse, die Vergleiche ermöglichen. Nicht alle der oben aufgeführten bedingten Sprunganweisungen kommen in den x86-Assemblerdateien unter arch/x86 des hier untersuchten Linux-Kernels vor, werden von McCa-

be_Assembler jedoch dennoch behandelt, um auch andere x86-Assemblerprojekte für Vergleiche heranziehen zu können. Zu diesen Anweisungen gehört z.B. jp „Jump if parity“ oder auch jcxz „Jump if %CX register is 0“.

Metrix++ wird ebenfalls für die Analyse hinsichtlich der McCabe-Metriken eingesetzt. Doch obwohl sowohl pmccabe als auch Metrix++ gemäß deren Dokumentationen McCabe-Analysen durchführen [76],[9], liefern beide Programme höchst unterschiedliche Ergebnisse bei Anwendung auf den gleichen Quellcode. Daher würde es wenig Sinn machen, die Ergebnisse von Metrix++ mit denen von pmccabe zu vergleichen.

Die Möglichkeit der Konfiguration eines vorhandenen Werkzeugs wiederum lässt sich in Bezug auf den AGC-Code gut mit dem Commented Code Detector durchführen, da dieses im Quellcode eine Liste der Schlüsselwörter mitführt, die sich austauschen lässt. Der Commented Code Detector verwendet für seine Analysen die Halstead-Metriken.

Das selbstgeschriebene Tool „Halstead_AGC“ geht zwar ähnlich vor, ist jedoch völlig unabhängig entstanden und unterscheidet auch zwischen verschiedenen Arten von Operatoren- bzw. Operandenwörtern, da es speziell für den AGC-Code entwickelt wurde. Außerdem ist die Laufzeit des in C geschriebenen „Halstead_AGC“ wesentlich geringer als die des als Python-Skript realisierten Commented Code Detectors.

11.6.4 Qualitätsanalysen

Wie lässt sich die Qualität einer Software messen? Wie lässt sich Qualität in Verbindung mit Softwareprodukten überhaupt definieren? Die Nachrichtentechnische Gesellschaft (NTG – heute: Informationstechnische Gesellschaft (ITG)) des Verbands der Elektrotechnik, Elektronik und Informationstechnik (VDE) hat gemeinsam mit der Deutschen Gesellschaft für Qualität (DGQ) bereits 1986 einen Leitfaden zur Qualitätssicherung von Software herausgebracht [31].

In dem Buch der NTG werden nicht nur konkrete Definitionen, sondern auch praktisch anwendbare Vorgehensweisen beschrieben. Qualität wird darin folgendermaßen definiert:

> „Beschaffenheit einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Anforderungen zu erfüllen“. [31, S. 11]

Software wiederum umfasst hier sowohl den Code als auch die Dokumentation [31, 27ff]. Die Qualitätsmerkmale für Software werden dementsprechend eingeteilt in Merkmale für Programme und Merkmale für die Dokumentation von Software.

Bei der Entwicklung der Software für den AGC hatte die Dokumentation einen sehr hohen Stellenwert; diese diente neben der Dokumentation der Arbeit auch der Kommunikation, der Fehlerbereinigung, der Anpassung an einzelne Missionen und weiterer Aspekte, wie z.B. der Öffentlichkeitsarbeit. Da sowohl der AGC-Quellcode als auch zahlreiche Originaldokumente verfügbar sind, eignen sich die Vorgehensweisen der VDE-Publikation sehr gut für eine qualitative Analyse der AGC-Software. Einzuwenden ist jedoch, dass von der gesamten AGC-Dokumentation bis heute nur ein – vermutlich geringer – Teil veröffentlicht wurde. So sind z.B. Dokumentationen speziell zum GAP-Assembler nur schwer zu finden, was natürlich weder besagt, dass es keine Dokumentation gab, noch dass es sehr umfangreiche Dokumentationen gab. Da jedoch die verfügbaren Dokumente auch zu einzelnen Fragestellungen sehr detailliert sind, ist anzunehmen, dass auch der GAP-Assembler gut dokumentiert war bzw. ist. Bleibt man sich der Einschränkung bewusst, möglicherweise nur einen Teil der Dokumentation zur Verfügung zu haben, so lässt sich jedoch eine Qualitätsanalyse mit dem verfügbaren Material gut durchführen.

Wesentliche Aspekte zur Beurteilung der Softwarequalität sind gemäß [31] die folgenden:

Programme	**Dokumentation**
Anpassbarkeit	Änderbarkeit
Benutzbarkeit	Aktualität
Effizienz	Eindeutigkeit
Funktionsabdeckung	Identifizierbarkeit
Korrektheit	Normenkonformität
Instandsetzbarkeit	Verständlichkeit
Portabilität	Vollständigkeit
Robustheit	Widerspruchsfreiheit
Sicherheit	
Verknüpfbarkeit	
Wiederverwendbarkeit	
Zuverlässigkeit	

Zur Beurteilung dieser Kriterien werden in [31] hauptsächlich Checklisten vorgeschlagen. Dies ist ein sinnvolles Vorgehen für die Qualitätssicherung, liefert jedoch keine konkreten Zahlenwerte als Ergebnis, sondern gibt vielmehr Hinweise, welche Stellen des Entwicklungsprozesses überarbeitet werden sollten.

In Bezug auf die AGC-Software kann man also untersuchen, inwieweit solche Qualitätskriterien bereits angewendet wurden, unabhängig davon, ob diese Kriterien bereits schriftlich niedergelegt waren oder nicht.

11.7 Durchführung der Analysen

Eine Auswertung der Dokumentation lässt sich manuell durchführen. Auch die Anwendung von Softwaremetriken auf den Quellcode lässt sich prinzipiell – wie bereits angesprochen – von Hand durchführen. Da hier jedoch zum einen umfangreiche Mengen an Quellcode zu bearbeiten sind und zum anderen die Analysen aus sich wiederholenden Vorgängen bestehen (siehe auch Anmerkungen zur Fehleranfälligkeit manueller Methoden in Abschnitt 11.6), sind automatisierte Verfahren durchaus empfehlenswert und werden auch für die Analyse verwendet.

Um ein vorhandenes Werkzeug für den AGC-Code anpassen zu können, muss dieses nach Auswahl eines geeigneten Werkzeugs entsprechend modifiziert bzw. konfiguriert werden. Für die Halstead-Analysen wird sowohl das Tool „Halstead_AGC“ eingesetzt als auch der Commented Code Detectors. Im Commented Code Detectors sind hierfür die (vorgegebenen) Arrays mit C-Operatoren und -Operanden durch Arrays mit AGC-Operatoren/Operanden ersetzt worden. Für Halstead_AGC ist keine Anpassung nötig, da es ja speziell für den AGC-Code entwickelt wurde.

Um auch Tools wie pmccabe, metrix++ und andere verwenden zu können, ist der Weg beschritten worden, Teile des AGC-Codes auf moderne Sprachen (C und Fortran) umzusetzen. Dabei war es nicht das Ziel, die komplette Funktionalität zu transferieren, sondern nur die Ablauf- bzw. Verschachtelungsstruktur, dies ermöglicht bereits Analysen.

Für eine solche Konvertierung werden zunächst die Kommentare entfernt, dies lässt sich mit sed durchführen:

```
#! /bin/bash
filename=$1
extension="${filename##*.}"
filename_wo_extension="${filename%.*}"
newfile=$filename_wo_extension.wo_c.$extension
sed 's/#.*$//g' $1 > $newfile
sed -i '/^\s*$/d' $newfile
```

Als nächstes werden Bezeichner ersetzt, da in den Labeln des AGC-Quellcodes Symbole vorkommen, die oftmals nicht nur mit C, sondern auch mit anderen modernen Programmiersprachen unverträglich sind. Diese sind:

Der Schrägstrich „/“

Der Bindestrich „-“
Der Punkt „.“
Das Komma „,“
Das Plus „+“
Ziffer am Anfang des Labels

In C und Fortran dürfen z.B. keine Ziffern am Beginn eines Funktionsnamens stehen. Ebenso lassen sich in diesen Programmiersprachen keine „+“, „-“, „/“ und „,“ in den Funktions- bzw. Subroutine-Namen verwenden.

Ähnlich verhält es sich mit Skriptsprachen wie Python. Auch dort dürfen zu Beginn eines Funktionsnamens keine Ziffern stehen (auch nicht „+“, „-“, „/“ und „,“).

Es gibt durchaus Programmiersprachen, in denen nahezu alle Zeichenkombinationen als Bezeichner verwendet werden können. In LISP ist z.B. „9/vol+7“ ein gültiger Funktionsname, doch ein Komma im Funktionsnamen wäre auch in LISP nicht gültig.

Um eine Konformität mit Sprachen wie C, Fortran, Java, Python, etc. zu erreichen sind daher für die Komplexitätsanalyse einige AGC-Bezeichner umbenannt worden.

Zeichen	**Ersetzung**
Schrägstrich „/“	SLASH
Bindestrich -	HYPHEN
Punkt „.“	POINT
Komma „,“	COMMA
Plus „+“	PLUS
Ziffer am Anfang	Ausgeschriebene Ziffer, bzw. Zahl (US-Amerikanisch)

Tabelle 8. Zeichenersetzungen in AGC-Bezeichnern

Nach diesen Umbenennungen lassen sich nun die Aufrufe der einzelnen Unterprogramme leicht in C-Syntax übertragen, um die Komplexität der Verschachtlungen ermitteln zu können. Bei einer solchen Übertragung ist jedoch auch zu beachten, dass Programme für Flugmanöver häufig in Interaktion mit der Besatzung abliefen. Ein- und Ausgaben sind daher ebenfalls angemessen zu übertragen bzw. zu simulieren.

Die Konvertierung in ein verschachteltes C-Programm lässt sich mit Programmen wie dem Stream-Editor sed zu großen Teilen automatisieren. Dies kann z.B. mit folgendem Script durchgeführt werden:

```
#! /bin/bash
# Sicherungskopie anlegen und TC XXXX durch Funktionsaufruf ersetzen,
# wobei Funktionsbezeichnungen ersetzt werden, die in c nicht gültig sind.
sed -r -i.orig 's/[[:cntrl:]]+(TC)[[:cntrl:]](\+)(.*)/\t\1 PLUS\3();/g' $1
sed -r -i 's/[[:cntrl:]]+(TC)[[:cntrl:]](.*)\/(.*)/\t\1 \2SLASH\3();/g' $1
sed -r -i 's/[[:cntrl:]]+(TC)[[:cntrl:]](1)(.*)/\t\1 ONE\3();/g' $1
sed -r -i 's/[[:cntrl:]]+(TC)[[:cntrl:]](2)(.*)/\t\1 TWO\3();/g' $1
sed -r -i 's/[[:cntrl:]]+(TC)[[:cntrl:]](8)(.*)/\t\1 EIGHT\3();/g' $1
sed -r -i 's/^\+([[:alnum:]]+)/PLUS\1\n/g' $1
sed -r -i 's/^\-([[:alnum:]]+)/HYPHEN\1\n/g' $1
# Bezeichner durch Funktionen ersetzen
sed -r -i 's/^([[:alnum:]]+)/int \1(){\n/g' $1
sed -r -i 's/[[:cntrl:]]+TC[[:cntrl:]](.*)/\1();/g' $1
sed -r -i 's/(^int )/return 0;\n}\n\1/g' $1
# Weitere Anweisungen mittels printf ausgeben
sed -r -i '/^[[:cntrl:]]+TC.*/!s/^[[:cntrl:]]+(.*)/\tprintf("%s\\n","\1");/g' $1
# Restliche TC entfernen
sed -r -i 's/(^[[:cntrl:]]+)(TC.)(.*)/\1 \3/g' $1
```

```
# Außerhalb von printf Sonderzeichen und Ziffern säubern
sed -r -i '/printf/!s/\//SLASH/g' $1
sed -r -i '/printf/!s/\+/PLUS/g' $1
sed -r -i '/printf/!s/-/HYPHEN/g' $1
# Verbleibende Bezeichner ausgeben
sed -r -i 's/^(SLASH.*)/printf("%s\n",\1);/g' $1
sed -r -i 's/^(HYPHEN.*)/printf("%s\n",\1);/g' $1
sed -r -i '/^[[:cntrl:]]+printf.*/!s/^[[:cntrl:]]+(.*)/\tprintf("%s\\n","\1");/g' $1
sed -r -i 's/^([[:alnum:]])\(\).*/\tprintf("%s\\n","TC\t\1");/g' $1
# letzte schließende Klammer ergänzen
echo  "return␣0;␣}" >> $1
```

Auch für eine Aufteilung in Befehle/Argumente lässt sich der Stream-Editor sed einsetzen. Im Anhang, Abschnitt 16.12 ist das Beispiel eines Skripts hierfür wiedergegeben. Sed-Skripte werden jedoch schnell unübersichtlich, wie die Listings zeigen. Besser geeignet ist Perl, eine Skriptsprache, die ebenso wie sed reguläre Ausdrücke versteht und darüber hinaus den Komfort einer kompletten Programmiersprache bietet. Im Anhang, in den Abschnitten 16.10 und 16.11, sind Perl-Skripte wiedergegeben, die die Umwandlung eines AGC-Quelltextes in ein analysierbares C-Programm und zurück durchführen. Die beiden im Anhang dargestellten Skripte arbeiten auch mit kommentiertem AGC-Code. Beim Zurückübersetzen erhält man also wieder die Originalstruktur inklusive der Kommentare.

Zur Untersuchung wurden verschiedene Bereiche des Quellcodes herangezogen. Sowohl (YUL- bzw. GAP-) Assembler-Module als auch Interpreter-Module. Als Vertreter für Letztere wurden Navigations- bzw. Navigationshilfsfunktionen, wie z.B. ANGLFIND, herangezogen. Das Programm PINBALL GAME BUTTONS AND LIGHTS wiederum wurde als primärer Vertreter der Basisassembler-Programme herangezogen, da es zum einen eines der umfangreichsten Module bzw. Programme im gesamten AGC-Code darstellt, zum anderen sowohl im Luminary- als auch im Colossus-Code verwendet wird. Außerdem wurde dieses Programm zwischen den frühen Versionen der Block-II-Software und den späteren Versionen kaum verändert.

11.8 Ergebnisse der Analysen

Was ergeben nun die durchgeführten Analysen und wie lassen sich diese interpretieren?

11.8.1 Quantität

Im Folgenden wird zur Analyse der Quellcode von Comanche055 herangezogen. Die transkribierten Seiten des entsprechenden Listings sind dafür so aufbereitet, dass sie den Original-Ausdrucken möglichst exakt entsprechen. Hierzu wurden die original Header am jeweiligen Beginn einer neuen Seite eingefügt. Damit jedoch auch Kommentare als solche von OHCount (oder anderen Tools) erkannt werden, müssen entsprechende Kommentarsymbole in den Quelltext eingefügt werden. Durch Markieren der Kommentare mittels „;“ lässt sich dann mit OHCount eine quantitative Analyse durchführen.

Das Programm PINBALL GAME BUTTONS AND LIGHTS hat folgende Verhältnisse von Code- zu Kommentar- bzw. Leerzeilen:

Language	Code	Comment	Comment %	Blank	Total
assembler	2461	888	26.5%	643	3992
Total	2461	888	26.5%	643	3992

Ohne die Seiten-Header verschieben sich diese Verhältnisse geringfügig:

```
Language    Code     Comment   Comment %       Blank       Total
---------------  ---------  ---------  ---------  ---------
assembler   2461         953       27.9%         367        3781
---------------  ---------  ---------  ---------  ---------
Total       2461         953       27.9%         367        3781
```

11.8.2 Häufigkeit

Tabelle 9 zeigt, wie oft die 20 häufigsten Instruktionen des Programms PINBALL GAME BUTTONS AND LIGHTS in diesem vorkommen.

Wie lassen sich diese Daten nun interpretieren? Nun, es überrascht sicher nicht, dass TC (Transfer Control), also der Aufruf eines (Unter-) Programms, an erster Stelle ist, da es sich hier um eine Assembler-Sprache handelt. In solchen Sprachen machen Sprunganweisungen häufig einen Großteil des Codes aus. Interessanter ist schon die große Häufigkeit von CCS (Count Compare and Skip), einer Anweisung, die Konstrukte ähnlich einer For-Schleife ermöglichen. In Kombination mit der INDEX-Anweisung kann damit eine Tabellenstruktur durchlaufen werden. Tabelle 9 hat natürlich nur statistische Aussagekraft, aber die nahe beieinanderliegenden Häufigkeiten der beiden Anweisungen CCS und INDEX deuten bereits auf eine häufige gemeinsame Verwendung hin. Im Code von PGBL wird diese Kombination jedoch nur einmal direkt verwendet, in der Funktion TESTBIT:

Instruktion	Anzahl
TC	697
TS	285
CS	124
INDEX	101
CCS	93
XCH	88
AD	80
MASK	62
CA	57
DXCH	39
EXTEND	31
TCF	28
DCA	22
BZF	21
LXCH	19
ADS	14
INCR	10
WAND	8
DAS	6
WOR	6

Tabelle 9. Häufigste 20 Anweisungen im Programm PINBALL GAME BUTTONS AND LIGHTS aus Comanche 055 mit ihren jeweiligen Häufigkeiten

```
TESTBIT MASK    NVTEMP      NVTEMP CONTAINS BLANKING CODE
        CCS     A
        TC      Q       IF CURRENT BIT = 1, RETURN TO L+1.
        INDEX   Q       IF CURRENT BIT = 0, RETURN TO L+3.
        TC      2
```

Die Häufigkeitsanalyse gibt hier also Hinweise, jedoch noch keine Belege für die gemeinsame Verwendung mehrerer Instruktionen. Aufgrund der Hinweise lässt sich der Code nun aber gezielter durchsuchen, was zu Erkenntnissen wie derjenigen zur o.g. Funktion TESTBIT führen kann, die in der Tat eine zusammenhängende Verwendung der Instruktionen CCS und INDEX beinhaltet.

Eine Liste mit den einzelnen AGC-Anweisungen ist im Anhang, in Abschnitt 16.3, aufgelistet. Darin befinden sich auch die sogenannten „Nonprogrammed Sequences“ (auch als unwillkürliche „involuntary“ Instruktionen bezeichnet [53, S. 70ff]), die der Programmierer nicht direkt verwenden kann. Dabei handelt es sich um interne Anweisungen, die z.B. Timer inkrementieren, Interrupts auslösen oder bei einer Datenübertragung von Mission Control die Bits in den Eingangsregistern verschieben (Für seriellen Datenempfang) [51, S. 19],[54, S. 78ff]. Der

Befehlssatz des AGC wurde während der Entwicklung mehrfach erweitert, wobei verschiedene „Tricks“ verwendet worden sind, möglichst viele Befehle realisieren zu können. Waren alle diese Befehle aber auch nötig? Eine Analyse der Verwendung zeigt, das einige Befehle fast nirgends verwendet wurden. So wurde z.B. die EDRUPT-Anweisung (eine Art Software-Interrupt, der vom Programmierer ansprechbar ist) nur im Luminary-Code eingesetzt und auch dort nur einmal.

Sieht man sich die Verteilung der Häufigkeiten für den kompletten Colossus-Code an, so treten wiederum die Sprunganweisungen hervor. Abbildung 46 zeigt, das „TC“ etwa genauso häufig ist, wie die im Diagramm unter „Übrige“ zusammengefassten 28 Anweisungen (Nonprogrammed Seuqences nicht mitgezählt) zusammen.

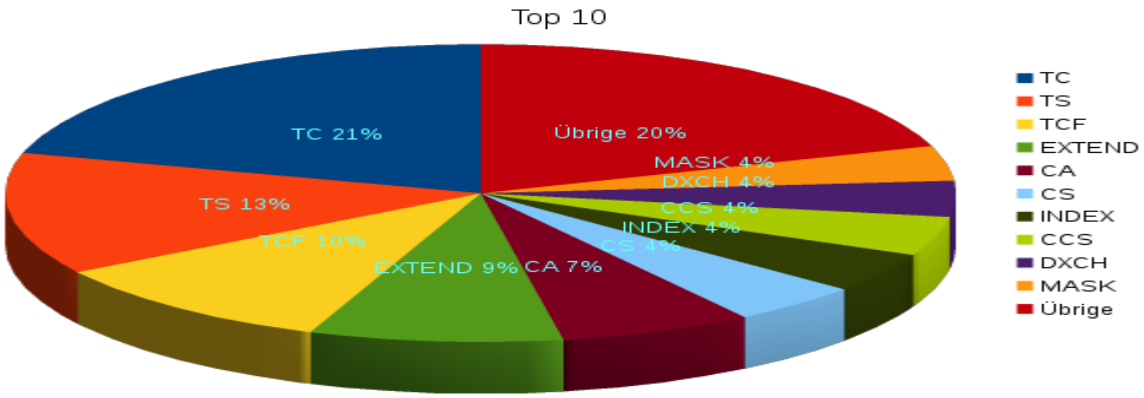


Abbildung 46. Verteilung der 10 häufigsten AGC-Instruktionen in Comanche055

Gruppiert man die Befehle in Sprungbefehle und Speicher-/Ladebefehle (Lese-/Schreiboperationen), so wird noch deutlicher, dass beide Gruppen zusammen die überwiegende Mehrheit der verwendeten AGC-Instruktionen ausmachen (Abbildung 47). Hier haben wir also ein durchaus typisches Bild einer Assembler-Programmierung.

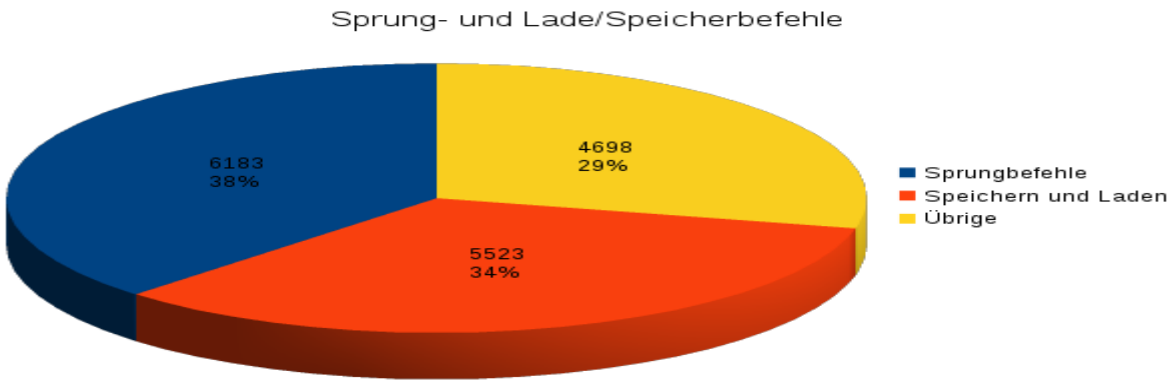


Abbildung 47. Sprunganweisungen und Lade-/Speicherbefehle aus Comanche055

Die Gruppe der Sprunganweisungen besteht aus 6 verschiedenen Befehlen, die Gruppe der Lade-/Speicherbefehle aus 9 unterschiedlichen Befehlen, die in den folgenden Tabellen gemäß [54, S. 74] aufgelistet sind:

Sprungbefehle
TC
TCF
CCS
BZF
BZMF
EDRUPT

Tabelle 10. Sprunganweisungen des Basis-AGC-Codes

Speichern und Laden
CA
CS
DCA
DCS
TS
XCH
LXCH
QXCH
DXCH

Tabelle 11. Lade- und Speicheranweisungen des Basis-AGC-Codes

Anzumerken ist, dass der EDRUPT-Befehl zwar mitgezählt wurde, dieser jedoch im Colossus-Code nicht verwendet wird, da er ja nur in Luminary eingesetzt wurde.

Wie sieht eine entsprechende Analyse der Verteilung einzelner Befehle nun beim Interpreter aus? Dabei handelt es sich ja um eine eigene Sprache bzw. Sprachschicht, dementsprechend sollten sich auch andere Analyseergebnisse zeigen, als bei der Analyse der YUL-Assembler-Befehle.

Abbildung 48 zeigt, wie oft die 10 häufigsten Interpreter-Anweisungen des Codes von Comanche 055 vorkommen. Und hier fallen bereits deutlich Unterschiede zur Analyse der „Standard“-Assemblerbefehle auf:

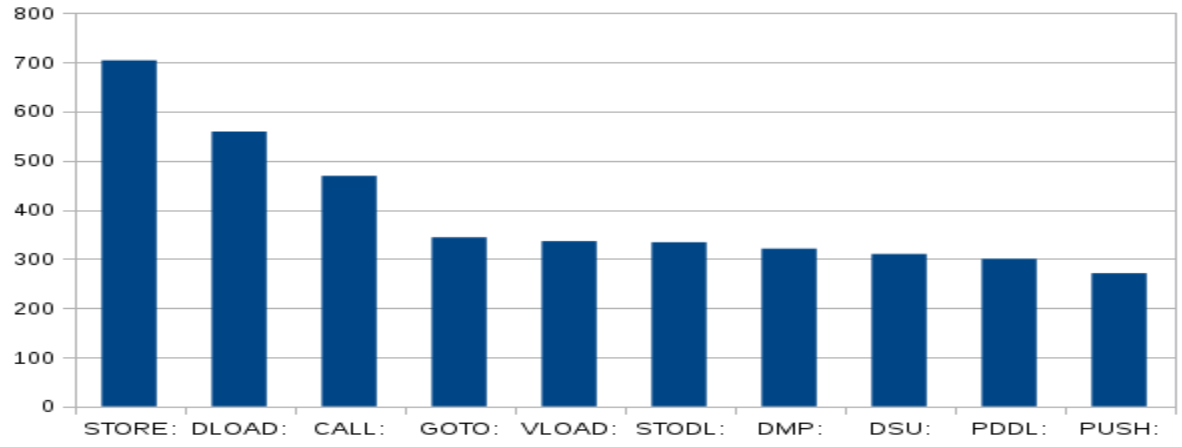


Abbildung 48. Die 10 häufigsten Interpreter-Instruktionen in Comanche055

Sieht man sich hier die Verteilung an, so erscheint bei weitem kein so klares Bild wie bei den Basis-Instruktionen. Im Diagramm (Abb. 49) wird die Verteilung der 20 häufigsten Befehle dargestellt, da die Verteilung wesentlich gleichmäßiger als bei den Basis-Instruktionen ist, und die Betrachtung von lediglich 10 Instruktionen noch nicht sehr aussagekräftig ist. Außerdem gibt es wesentlich mehr Interpreter-Instruktionen als Basis-Instruktionen.

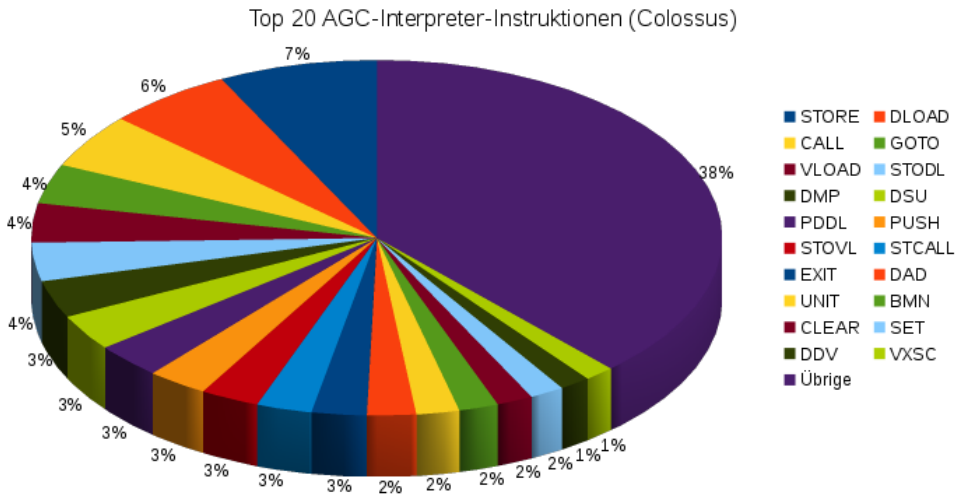


Abbildung 49. Verteilung der 20 häufigsten Interpreter-Instruktionen in Comanche055

Doch die Menge an Instruktionen allein erklärt die Verteilung noch nicht. In Abbildung 50 wird die Verteilung sämtlicher Interpreter-Instruktionen in Comanche 055 dargestellt, sie bleibt relativ gleichmäßig. Das liegt sicher u.A. daran, dass man nicht ständig Sprunganweisungen zu schreiben braucht. Aber auch daran, dass der Instruktionssatz nicht ohne Grund diese Befehle enthält; die einzelnen Befehle sind zweckdienlich, deshalb werden die meisten auch relativ häufig eingesetzt. Diese Art der Verwendung und der umfangreiche Befehlssatz lassen den Interpreter-Instruktionssatz schon fast wie ein Hochsprache erscheinen, wobei die nahe Verwandtschaft zum Assembler-Code weiterhin deutlich bleibt. Sprunganweisungen dominieren auch hier, z.B. GOTO oder CALL (Unterprogrammaufruf), wenn auch nicht so deutlich, wie bei den Anweisungen des Basis-Befehlssatzes.

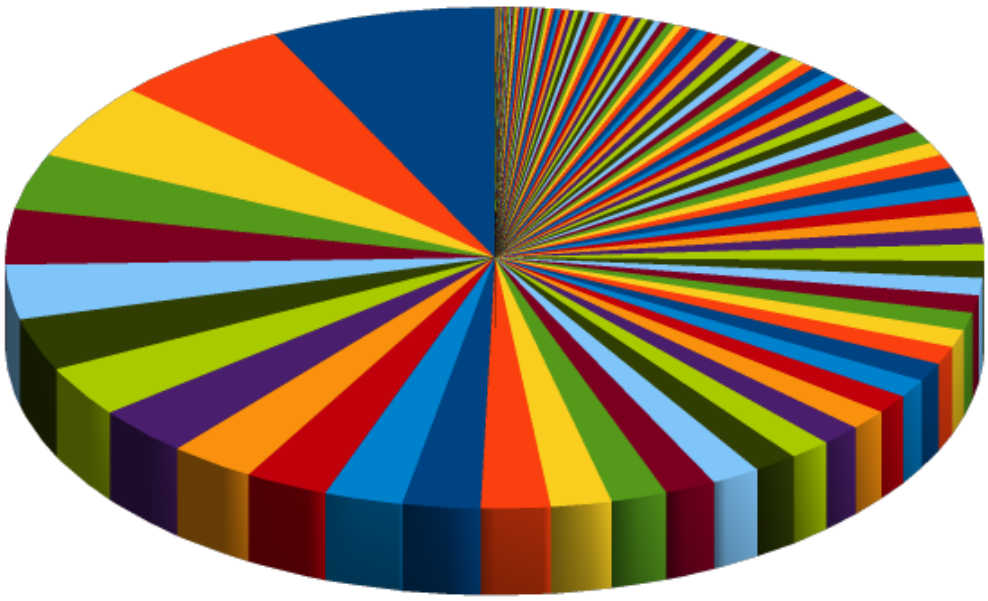


Abbildung 50. Verteilung sämtlicher Interpreter-Instruktionen in Comanche055

11.8.3 Funktionsgrößen

Wie groß sind nun die einzelnen Funktionen in den Programmen? Abbildung 51 zeigt die Größen der längsten 20 Funktionen in PINBALL GAME BUTTONS AND LIGHTS. Die durchschnittliche Länge einer Funktion beträgt hier 6,082 Zeilen ohne Kommentarzeilen. Werden die Kommentarzeilen mit berücksichtigt, so ergibt sich eine durchschnittliche Länge von 9,88 Zeilen, da PGBL sehr gut kommentiert ist. PGBL eignet sich auch hier als eines der größten Programme im AGC-Code gut zur Untersuchung.

Bei Interpreter-Programmen sind die Größen einzelner Funktionen schon höher. Untersucht man das Programm ANGLFIND aus Colossus, so findet man eine durchschnittliche Funktionslänge von 12,2 Zeilen ohne Kommentarzeilen bzw. 14,8 Zeilen unter Einbeziehung der Kommentarzeilen. Dies sieht für sich noch nicht bemerkenswert aus, doch sieht man sich auch hier die Längen der längsten Funktionen an, so fallen schon deutliche Unterschiede zu PGBL auf. Die längsten drei Funktionen in ANGLFIND sind DELCOMP mit 85 Zeilen, LOOPSIN mit 76 Zeilen sowie SECAD mit 59 Zeilen (jeweils ohne Kommentarzeilen), wohingegen die – mit Abstand – längste Funktion in PGBL HMSIN ist, die eine Länge von 50 Zeilen hat. ANGLFIND ist mit 488 Zeilen (605 Zeilen inkl. Kommentarzeilen) wesentlich kürzer als PGBL und enthält auch kurze Funktionen, aber eben auch die bereits erwähnten längeren. Hier wird wieder der Unterschied des Basis-Assemblers (YUL, GAP) zum Interpreter deutlich. Die im Basis-Assembler geschriebenen Programme enthalten zahlreiche, überwiegend kürzere Funktionen, die Interpreter-Programme enthalten meist weniger, dafür aber längere Funktionen.

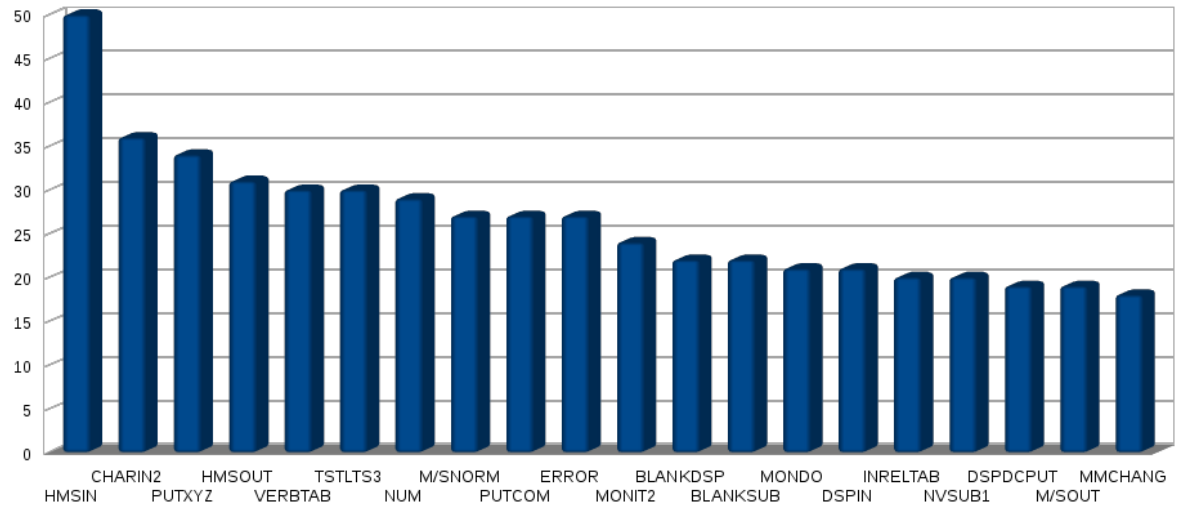


Abbildung 51. Die längsten 20 Funktionen in PGBL (ohne Kommentarzeilen)

11.8.4 Metrik-Berechnungen nach Halstead

Um über Häufigkeitsanalysen hinaus zu gehen, benötigt man nun speziellere Vorgehensweisen bzw. speziellere Programme. Je nach der zu verwendenden Metrik ist die Entwicklung eines Analyseprogramms für den AGC-Code unterschiedlich aufwändig. Wie bereits im Abschnitt 11.5 dargelegt, liefern nicht alle Software-Metriken sinnvolle Ergebnisse bei Anwendung auf den AGC-Code. Die Halstead-Metriken (sowie einige andere) sind, wie in Abschnitt 11.8.4 dargelegt, jedoch sowohl allgemein als auch aussagekräftig genug, um auch auf den AGC-Code angewandt zu werden. Da sich die Halstead-Metriken außerdem relativ leicht berechnen lassen, muss ein solches Analyseprogramm für den AGC-Code auch nicht selbst schon unnötig komplex aufgebaut sein. Im Anhang (Abschnitt 16.7) ist ein einfaches Programm zur Berechnung der Halstead-Metriken des AGC-Codes dargestellt. Schwierig ist dabei eher die Unterscheidung von Operanden und Operatoren, doch diese lässt sich dem

vorhandenen AGC-Code entnehmen und entsprechend einsetzen. Und hat man diese erst einmal, so lassen sich die Halstead-Werte leicht berechnen.

Eine Analyse des Colossus-Codes mit dem im Anhang dargestellten Programm ergibt in Bezug auf den Basis-AGC-Befehlssatz (YUL-Assembler) folgende Werte:

Anzahl Befehle insgesamt (N_1):	24654
Anzahl Argumente insgesamt (N_2):	8917
Anzahl Kommentare:	29659
Anzahl unterschiedlicher Befehle (η_1):	52
Anzahl unterschiedlicher Argumente (η_2):	575
Komplexität:	0.00014
Programmlänge (Implementierungslänge N):	33571
Programmvokabular:	627
Berechnete Programmlänge (Halstead-Länge):	5567.69
Programmvolumen (V):	311952.50
Schwierigkeit (D):	403.20
Aufwand (E):	125780328.00
Niveau:	0.00248
Programmierzeit (Sekunden):	6987796.00 Sekunden
Programmierzeit (Tage und Stunden):	80 Tage, 21:03:00 Stunden
Ausgelieferte Fehler nach Halstead	83.68

Tabelle 12. Halstead-Metriken für Comanche055, Basis-Befehlssatz ohne Interpreter

Sind diese Werte brauchbar? Nun, der hohe Wert für Aufwand und das sehr niedrige Niveau fallen sicher sofort auf, allerdings muss hier natürlich berücksichtigt werden, dass es sich um Assembler-Code handelt und hier außerdem die Ergebnisse des kompletten Colossus-Codes dargestellt sind.

Daher werden im Folgenden wiederum einzelne Module (Programme) des AGC untersucht. Außerdem wird der AGC auch in Bezug auf den Interpreter mit den Halstead-Metriken untersucht.

Metriken für Colossus komplett (Interpreter):

```
----------------------------------------
Halstead-Metriken für Colossus.agc
Untersucht auf 173 Operatoren (Befehle)
und 149 Operanden (Argumente)
----------------------------------------
Anzahl Befehle gesamt (N1):  8512
Anzahl Argumente gesamt (N2): 11323
Anzahl Kommentare: 29659
Anzahl unterschiedlicher Operatoren (n1): 154
Anzahl unterschiedlicher Operanden (n2): 149
Komplexität: 0.00024
Programmlänge (Implementierungslänge N): 19835
Programmvokabular (n): 303
Berechnete Programmlänge (Halstead-Länge): 2194.74
Programmvolumen (V): 163503.34
Schwierigkeit (D): 5851.48
Aufwand (E): 956737024.00
Niveau: 0.00017
```

```
Programmierzeit: 53152056.00 Sekunden
Programmierzeit: 615 Tage, 04:28 Stunden
Ausgelieferte Fehler nach Halstead: 323.65
----------------------------------------
```

Halstead-Metriken für R30 (Interpreter):

```
Halstead-Metriken für 026_R30.s
Untersucht auf 173 Operatoren (Befehle)
und 149 Operanden (Argumente)
----------------------------------------
Anzahl Befehle gesamt (N1):  78
Anzahl Argumente gesamt (N2): 106
Anzahl Kommentare: 278
Anzahl unterschiedlicher Operatoren (n1): 25
Anzahl unterschiedlicher Operanden (n2): 28
Komplexität: 0.08466
Programmlänge (Implementierungslänge N): 184
Programmvokabular (n): 53
Berechnete Programmlänge (Halstead-Länge): 250.70
Programmvolumen (V): 1053.94
Schwierigkeit (D): 47.32
Aufwand (E): 49873.82
Niveau: 0.02113
Programmierzeit: 2770.77 Sekunden
Ausgelieferte Fehler nach Halstead: 0.45
----------------------------------------
```

11.8.5 Metrik-Berechnungen nach McCabe

Die Module des AGC sind zwar meistens sehr umfangreich, weisen jedoch nicht zwangsläufig auch hohe Ablaufkomplexitäten auf. Abbildung 52 zeigt ein Beispiel aus Colossus. Dabei handelt es sich um eine der Navigationsroutinen, die den Interpreter aufrufen.

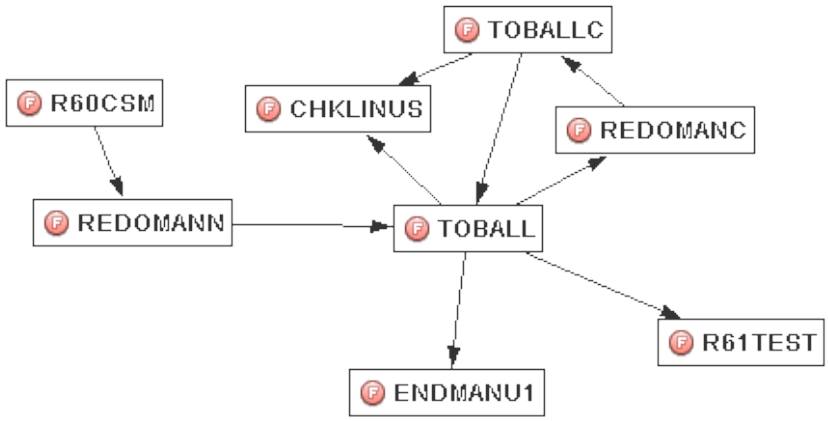


Abbildung 52. Ausschnitt aus dem Aufrufgraphen der Navigationsroutine R60 aus Colossus

Die Routine R60, die zusammen mit R61 und R62 einen zusammenhängenden Block bildet (in den Colossus-Listings bezeichnet mit „R60,R62“), ist natürlich umfangreicher als es Abbildung 52 zeigt. Da aber „R60,R62“ mit der Besatzung interagieren, wird nicht zwangsläufig jeder Teilgraph aufgerufen. Daher ist es durchaus möglich, Teilabläufe als einzelne Graphen darzustellen. Durch Konvertierung der Funktionsaufrufe im AGC-Code nach C lässt sich pmccabe zur Einschätzung der McCabe-Komplexität für die einzelnen Module einsetzen. Die Analyse von R60,R62 mittels pmccabe ergibt dabei folgende Werte:

```
Modified McCabe Cyclomatic Complexity
|   Traditional McCabe Cyclomatic Complexity
|       |    # Statements in function
|       |        |   First line of function
|       |        |       |   # lines in function
|       |        |       |       |  filename(definition line number):function
|       |        |       |       |           |
1       1       4       21      9       r60_62.c(21): R60CSM
2       2       12      31      17      r60_62.c(31): REDOMANN
1       1       12      49      15      r60_62.c(49): TOBALL
2       2       12      65      21      r60_62.c(65): REDOMANC
1       1       14      87      17      r60_62.c(87): TOBALLC
1       1       2       105     5       r60_62.c(105): STARTMNV
1       1       3       111     6       r60_62.c(111): ENDMANUV
1       1       4       118     7       r60_62.c(118): ENDMANU1
2       2       16      126     24      r60_62.c(126): CHKLINUS
1       1       16      151     19      r60_62.c(151): RELINUS
1       1       3       171     7       r60_62.c(171): GOREDO20
1       1       9       179     12      r60_62.c(179): R61TEST
1       1       2       192     5       r60_62.c(192): BIT14plus7
1       1       7       198     10      r60_62.c(198): V06N18
1       1       4       210     7       r60_62.c(210): VECPOINT
1       1       25      218     28      r60_62.c(218): VECLEAR
1       1       32      247     35      r60_62.c(247): COMPMATX
1       1       11      283     14      r60_62.c(283): IGSAMEX
1       1       4       297     7       r60_62.c(297): UequalsSCAXIS
1       1       10      305     13      r60_62.c(305): CHEKAXIS
1       1       3       319     6       r60_62.c(319): PICKANG1
1       1       10      326     13      r60_62.c(326): COMPMFSN
1       1       10      340     13      r60_62.c(340): FINDGIMB
1       1       10      354     13      r60_62.c(354): PICKAXIS
1       1       15      368     18      r60_62.c(368): ROT180
1       1       5       387     8       r60_62.c(387): XROT
1       1       8       396     11      r60_62.c(396): PICKX
1       1       2       408     5       r60_62.c(408): SINGIMLC
1       1       2       414     5       r60_62.c(414): SINVEC1
1       1       2       420     5       r60_62.c(420): SINVEC2
1       1       2       426     5       r60_62.c(426): VECANG1
1       1       2       432     5       r60_62.c(432): VECANG2
1       1       2       438     5       r60_62.c(438): ONEBITDPOCT0
1       1       10      443     13      r60_62.c(443): DPBhyphen14
1       1       12      457     15      r60_62.c(457): R62DISP
1       1       7       473     10      r60_62.c(473): GOMOVE
39      39      346     1       501     r60_62.c
```

Eine McCabe-Komplexität von 39 für das gesamte Modul ist auch aus heutiger Sicht nicht ungewöhnlich hoch. Allerdings handelt es sich bei R60_62 auch um ein relativ kleines Modul (360 Zeilen), bei den größeren Modulen sind auch höhere McCabe-Werte zu finden. Genauere Ergebnisse lassen sich mit dem im Anhang (Abschnitt 16.8) dargestellten Programm McCabe_AGC erzielen. Doch auch mit diesem Programm zeigt sich, dass die Komplexität pro Funktion relativ gering ist:

```
Werte pro Funktion (Sprungmarken (Label)):
  TC      TCF     BZMF      CCS   McCabe  Sprunganw.  Funktion Datei:016_R60_62.agc
   1        0        0        0        1           1    R60CSM
   1        1        0        1        5           6    REDOMANN
   5        1        0        0        1           6    TOBALL
   1        1        0        1        5           6    REDOMANC
   2        1        0        0        1           3    TOBALLC
   1        0        0        0        1           1    STARTMNV
   0        1        0        0        1           1    ENDMANUV
```

```
   2        0        0        0        1           2    ENDMANU1
   4        0        0        1        5           8    CHKLINUS
   4        0        0        0        1           4    RELINUS
   1        0        0        0        1           1    GOREDO20
   1        0        0        0        1           1    R61TEST
   0        0        0        0        1           0    BIT14+7
   0        0        0        0        1           0    V06N18
   0        0        0        0        1           0    VECPOINT
   0        0        0        0        1           0    VECLEAR
   0        0        0        0        1           0    COMPMATX
   0        0        0        0        1           0    IGSAMEX
   0        0        0        0        1           0    U=SCAXIS
   0        0        0        0        1           0    CHEKAXIS
   0        0        0        0        1           0    PICKANG1
   0        0        0        0        1           0    COMPMFSN
   0        0        0        0        1           0    FINDGIMB
   0        0        0        0        1           0    PICKAXIS
   0        0        0        0        1           0    ROT180
   0        0        0        0        1           0    XROT
   0        0        0        0        1           0    PICKX
   0        0        0        0        1           0    SINGIMLC
   0        0        0        0        1           0    SINVEC1
   0        0        0        0        1           0    SINVEC2
   0        0        0        0        1           0    VECANG1
   0        0        0        0        1           0    VECANG2
   0        0        0        0        1           0    1BITDP
   0        0        0        0        1           0    DPB-14
   1        3        0        0        1           4    R62DISP
   2        1        0        0        1           3    GOMOVE
Gesamtwerte für 016_R60_62.agc
TC:  26    TCF:   9   BZMF:   0    CCS:   3 McCabe:  48
Sprunganweisungen gesamt:  47   Funktionen (Label):  36
Codezeilen:   248
Verhältnis Sprunganweisungen/Codezeilen:    0.19
Entpricht 18.95 Sprunganweisungen auf 100 Codezeilen
Verhältnis Funktionen/Codezeile    0.15
Entpricht 14.52 Funktionen auf 100 Codezeilen
Durchschnittliche Funktionslänge:   6.89 Zeilen
```

Durch die Auswertung der einzelnen bedingten Sprunganweisungen in den Funktionen erhalten wir einen Wert von 48 für das gesamte Modul. Die Werte des in C konvertierten AGC-Codes weichen also nicht stark ab von den Werten, die mittels McCabe_AGC für den unveränderten Code ermittelt wurden. In R60_62 existieren drei Funktionen mit McCabe-Wert von 5, die übrigen liegen bei 1.

Pro Funktion sind die Werte meistens gering und liegen nur selten über 1. Doch hohe Werte pro Funktion sind selbst heutzutage nicht ungewöhnlich, wie Programme aus dem Linux-Kernel zeigen (siehe 11.11).

P76, ein weiteres Programm, das den Interpreter aufruft, zeigt folgenden grundlegenden Ablauf:

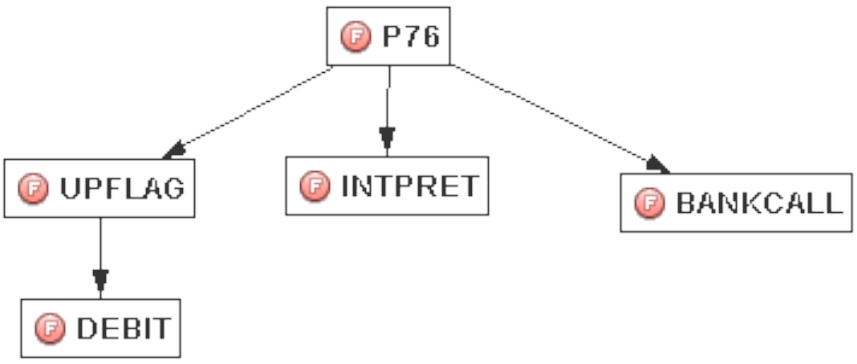


Abbildung 53. Ausschnitt aus dem Aufrufgraphen des Navigationsprogramms P76 aus Colossus

Wie sehen hier nun die Komplexitätswerte aus? Die McCabe-Maße, ermittelt mit McCabe_AGC, ergeben sich für P76 wie folgt:

```
Werte pro Funktion (Sprungmarken (Label)):
  TC      TCF     BZMF      CCS   McCabe  Sprunganw.  Funktion Datei:025_P76.agc
   7        1        0        0        1           8    P76
   0        0        0        0        1           0    COMPMAT
   3        0        0        0        1           3    INTOTHIS
   0        0        0        0        1           0    OUT
   0        1        0        0        1           1    ENDP76
   0        0        0        0        1           0    V06N84
   0        0        0        0        1           0    P76SUB1
Gesamtwerte für 025_P76.agc
TC:  10    TCF:   2   BZMF:   0    CCS:   0 McCabe:   7
Sprunganweisungen gesamt:  12   Funktionen (Label):   7
Codezeilen:   101
Verhältnis Sprunganweisungen/Codezeilen:    0.12
Entpricht 11.88 Sprunganweisungen auf 100 Codezeilen
Verhältnis Funktionen/Codezeile    0.07
Entpricht  6.93 Funktionen auf 100 Codezeilen
Durchschnittliche Funktionslänge:  14.43 Zeilen
```

Wie verteilen sich nun die McCabe-Werte auf die einzelnen Funktionen? Die meisten Funktionen haben sehr geringe McCabe-Werte, der Programmablauf dieser Funktionen ist sehr linear, die Werte liegen zumeist bei 1 bis 2. Siehe hierzu Abbildung 54, welche die Ergebnisse von Analysen mit Hilfe von McCabe_AGC darstellt.

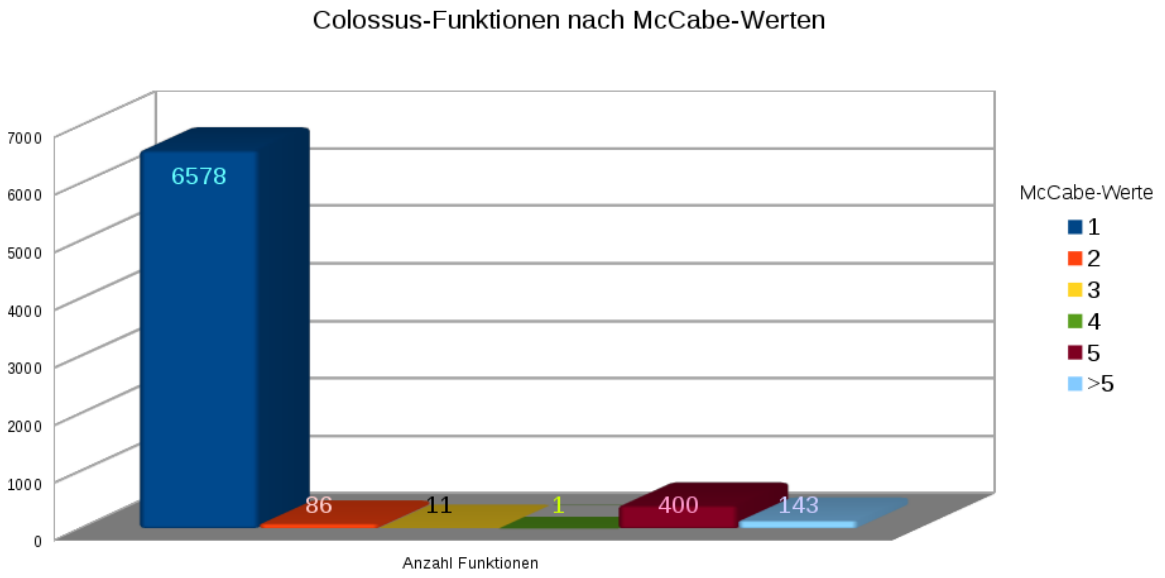


Abbildung 54. Anzahl Funktionen in Colossus nach McCabe-Werten

11.8.6 Sprungweiten

Im AGC-Code werden zahlreiche Funktionen in verschiedenen Programmen wiederholt aufgerufen. Dadurch werden häufig auch Sprünge durchgeführt, die über große Entfernungen gehen. Der Durchschnittswert für Colossus liegt bei 16.000 Zeilen. Dies ist jedoch weniger dramatisch als es auf den ersten Blick scheint, denn innerhalb der einzelnen Unterprogramme werden zumeist Funktionen des jeweiligen Unterprogramms aufgerufen. Die durchschnittliche Sprungweite liegt dabei nur bei ca. 50 Zeilen. Der hohe Gesamtdurchschnitt kommt daher, dass außer diesen Unterprogramm-eigenen Funktionen auch häufig Grundfunktionen des Systems, wie z.B. die Restart-

Funktionen, aufgerufen werden, die relativ weit vorn im Code zu finden sind (die Restart-Funktionen sind in Colossus z.B. ab Seite 181 des gedruckten Listings zu finden). Dadurch, dass auch in Programmen, die gegen Ende des Codes zu finden sind, solche Grundfunktionen aufgerufen werden, entsteht solch hoher Gesamtdurchschnitt.

Ein solcher Effekt ist auch auf modernen Systemen leicht zu finden: Wenn man ein C-Programm mittels der gcc-Option „-S“ (nur kompilieren, nicht assemblieren oder binden) in Assembler-Code übersetzen lässt, kann man darin bereits höhere Sprungweiten als im zugrundeliegenden C-Programm finden, da die Assembler-Syntax eben die komplexen Funktionen einer Hochsprache auf zahlreiche einfachere Befehle abbildet. Bedenkt man außerdem, dass den AGC-Entwicklern kein Linker zur Verfügung stand und somit der Code – trotz Aufteilung in einzelne Programme – eine Gesamteinheit bildet, so ist ein Hoher Wert der Sprungweiten nicht verwunderlich. Man sieht an solchen Ergebnissen natürlich auch, welchen Komfort heutige Entwickler – auch die von eingebetteten Systemen – haben. Ein Linker, die Verwendung höherer Sprachen und die Einbindung externer Bibliotheken hätte die AGC-Entwicklung sicher vereinfacht; wenn auch der AGC dadurch sicher nicht „besser“ geworden wäre, so wäre er doch vermutlich schneller fertig geworden.

11.8.7 Qualitätsanalysen der AGC-Software

Für die Untersuchung der Qualitätsaspekte wird zum einen der Code selbst betrachtet, zum anderen die verfügbare Dokumentation. Im Dokument „KEYBOARD AND DISPLAY PROGRAM AND OPERATION“ wird auf das Listing des DSKY-Programms als „the prime reference for the program's description“ verwiesen [7, S. 41]. Dies lässt vermuten, dass es (zumindest für PGBL) keine besonders umfangreiche Dokumentation außerhalb der Listings gab. Zwar wird in [7] auf weitere Dokumente zu einzelnen Routinen verwiesen, der Ausdruck „prime reference“ in Bezug auf das Listing ist aber schon sehr eindeutig. Das Listing ist auch in der Tat sehr umfangreich kommentiert und kann daher sowohl als Programm als auch als Dokumentation betrachtet werden. Die Beurteilung einzelner Qualitätsmerkmale erfolgt zunächst in beschreibender Form, anschließend werden anhand zweier Qualitätsmerkmale konkrete prozentuale Werte ermittelt.

Zur allgemeinen Beurteilung der Qualitätsaspekte gemäß [31] lässt sich folgendes festhalten:

Dokumentation

Änderbarkeit
Das Listing musste neu erstellt werden, wenn die Dokumentation darin geändert werden sollte. Dokumente mussten neu gedruckt werden. In einigen Dokumenten sind auch nachträgliche Änderungen erkennbar (z.B. in [26]), die allerdings auch Artefakte des Digitalisierungsprozesses sein könnten.

Aktualität
Kommentare im Quellcode sind – sofern sie bei Codeänderung nicht vergessen werden – eine äußerst aktuelle Beschreibung des Programms. Daher ist die Aktualität der Dokumentation beim AGC-Code sehr hoch.

Eindeutigkeit
Einheitliche Benennungen sind in den Dokumenten erkennbar, siehe z.B. [7]. Ebenso werden Zahlenangaben als Zahlen wiedergegeben.

Identifizierbarkeit
Jede Seite des Listings enthält die genaue Programmversion und das Datum der Erstellung, daher ist insbesondere die in-Code-Dokumentation eindeutig.

Normenkonformität
An den Dokumenten ist erkennbar, dass diese ähnlich strukturiert sind. Dies lässt feststehende Vorgabenormen als wahrscheinlich gelten. Ähnlich verhält es sich bei der Dokumentation im Code.

Verständlichkeit
Die verfügbaren Dokumentationen sind zumeist sehr verständlich geschrieben und erklären die einzelnen Funktionen präzise. Die in-Code-Dokumentation verwendet zur Beschreibung häufig Formeln aus der Weltraumnavigation und beschreibt jeweils die Zwecke; für eine fachkundige Person sollten diese Kommentare somit ebenfalls gut verständlich sein.

Vollständigkeit
Die in-Code-Dokumentation beschreibt nahezu jeden Bereich der Programme. Was die externe Dokumentation angeht, so ist es zumindest denkbar, dass nicht jedes Programmteil umfangreich dokumentiert wurde.

Widerspruchsfreiheit
Bei den hier durchgeführten Auswertungen vorliegender Dokumente sind keine Widersprüche aufgefallen. Dies bedeutet jedoch nicht die Widerspruchsfreiheit der gesamten Dokumentation, hierzu müsste eine komplette Überprüfung durchgeführt werden. Diese ist bereits aufgrund der Nichtverfügbarkeit einzelner Dokumente nicht durchführbar. Darüber hinaus würde es den Rahmen der vorliegenden Arbeit sprengen.

Programme

Anpassbarkeit
Die Software ist anpassbar und erweiterbar, siehe dazu auch die Tabelle 14.

Benutzbarkeit
Durch das neuartige und sehr intuitiv bedienbare DSKY ist die Systembenutzung für die Benutzer (Astronauten) komfortabel.

Effizienz
Der Funktionsumfang ist für die geringe Speicherkapazität des AGC sehr groß, die Effizienz sehr hoch.

Funktionsabdeckung
Es wurden nicht alle Funktionen umgesetzt, die ursprünglich angedacht waren (siehe auch Abschnitt 7.5), nach Änderung der geplanten Funktionalität wurde der entsprechend angepasste Funktionsumfang jedoch auch umgesetzt.

Korrektheit
Dieser Aspekt wäre nur durch eine Komplettüberprüfung und -analyse zu ermitteln, was den Rahmen der vorliegenden Arbeit sprengen würde.

Instandsetzbarkeit
Eine Fehlerkorrektur der Programme war möglich und wurde auch mit hohem Aufwand betrieben, siehe hierzu auch Tabelle 13.

Portabilität
Der AGC war nicht auf Portabilität ausgelegt, dieses Kriterium ist somit hier nicht anwendbar.

Robustheit
Die AGC-Software enthielt zahlreiche Methoden der Fehlerkorrektur, wie z.B. die Restart-Möglichkeit oder den Rupt-Lock-Test (siehe hierzu auch Abschnitt 7.7).

Sicherheit
Vom AGC ging keine besondere Gefährdung für die Astronauten aus. Aspekte, wie z.B. die Strahlung durch radiolumineszente Displays, lassen sich zwar durchaus als potentielle Gefährdung einstufen (siehe hierzu Abschnitt 10.1), doch auch wenn diese Displays durch die AGC-Software gesteuert werden, sind sie dennoch ein Teil der grundlegenden Technik des Apollo-Programms und keine spezielle Gefährdung, die durch die Software entsteht.

Verknüpfbarkeit
Durch den modularisierten Programmaufbau ist der Punkt Verknüpfbarkeit bereits unterstützt bzw. gegeben, jedoch

muss berücksichtigt werden, dass im AGC keine Standardprogrammiersprache verwendet wurde, sondern eine spezielle Assemblerversion. Daher lässt sich der Begriff der Verküpfbarkeit hier auch nur im Hinblick auf andere AGC-Programme sinnvoll anwenden.

Wiederverwendbarkeit
Die AGC-Software ließ sich ohne Änderungen nur im AGC selbst verwenden.

Zuverlässigkeit
Im Einsatz zeigte sich, dass die AGC-Software auch in Ausnahmesituation zuverlässig arbeitete, siehe hierzu auch Abschnitt 12.1.

Um für die Bewertung einzelner Qualitätskriterien der AGC-Software auch konkrete Zahlenwerte zu erhalten, werden die im Folgenden dargestellten Checklisten für die Qualitätsmerkmale Instandsetzung und Anpassbarkeit vorgeschlagen. Diese Checklisten basieren auf den Beispiel-Checklisten aus [31]. Die Merkmale Instandsetzung und Anpassbarkeit wurden ausgewählt, da diese mit den verfügbaren Quellen recht gut einzustufen sind. Zudem war die Anpassbarkeit sehr wichtig, um die Software des AGC auf die speziellen Ziele der einzelnen Missionen hin ausrichten zu können (siehe hierzu auch Abschnitt 7.5).

Die Anwesenheit eines Merkmals wird in dieser Analyse mit 1 bewertet, die Abwesenheit eines Merkmals mit 0. Bei den Merkmalen, die mit Werten zwischen 0 und 1 bewertet sind, ist dieser Wert ein Schätzwert und entsprechend kommentiert. Da diese Schätzwerte zwar sachlich begründet sind, dennoch aber eine subjektive Einschätzung darstellen, wird die Analyse sowohl mit diesen Werten als auch ohne diese durchgeführt: In der ersten Analyse werden diese Werte mit ihren jeweiligen Schätzwerten verwendet, in der zweiten Analyse fließen diese Werte mit dem geringstmöglichen Wert 0 ein.

Um ein prozentuales Ergebnis zu ermöglichen, wird die Summe der Einzelwerte durch die Anzahl untersuchter Kriterien dividiert, das Ergebnis wird dann mit 100 multipliziert.
Qualitätsmerkmal Instandsetzung:

Prüfung	**Wert**	**Kommentar**
Ist das Softwareprodukt im Änderungsfall eindeutig identifizierbar (Name, Version, Prüfsumme etc.)?	1	
Existiert eine systematische Software-Konfigurationsverwaltung, aus der die tatsächlich eingesetzte Softwareversion inkl. Dokumentation rekonstruierbar ist?	0	
Existiert zu jeder Softwareversion eine aktuelle und ausführliche Dokumentation?	0,5	Hauptsächlich im Quellcode dokumentiert, nicht alle (externen) Dokumente verfügbar
Gibt es Zusatzinformationen, die im Fehlerfall zur Analyse herangezogen werden können (Traces, Protokolle, Analyseoptionen)?	1	
Gibt es eine feststellbare Modularisierung nach Funktionen und/oder Daten?	1	
Ist der Entwurf dokumentiert und folgt er einer im Unternehmen bekannten Methode?	1	
Ist die Codierung in einer im Unternehmen bekannten Programmiersprache ausgeführt?	1	
Ist jedes Modul getrennt änderbar, erweiterbar und ersetzbar?	1	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Gibt es Hilfsmittel für die temporäre Fehlerbeseitigung?	1	
Gibt es Hilfsmittel, mit denen die Folgen einer Änderung auf andere Module und Datenstrukturen untersucht werden können?	1	
Gibt es Unterstützungsmittel bei der Kompilierung. die mögliche Syntax-, Datentypen- und Schnittstellenfehler prüfen?	0	
Ist die Prüfung einzelner Module möglich?	1	
Summe	9,5	

Tabelle 13. Checkliste für Qualitätsmerkmal Instandsetzung

Damit ergibt sich für das Qualitätsmerkmal Instandsetzung, nachfolgend als Q_I bezeichnet, folgendes:

$$Q_I = \frac{9,5}{12} \cdot 100$$
$$Q_I = 79,17\%$$

Qualitätsmerkmal Anpassbarkeit/Adaptabilität:

Prüfung	**Wert**	**Kommentar**
Ist das Softwareprodukt im Änderungsfall eindeutig identifizierbar (Name, Version, Prüfsumme etc.)?	1	
Existiert eine Software-Konfigurationsverwaltung?	0	
Existiert zu jeder Softwareversion eine aktuelle und ausführliche Dokumentation?	0,5	Hauptsächlich im Quellcode dokumentiert, nicht alle (externen) Dokumente verfügbar
Gibt es eine feststellbare Modularisierung nach Funktionen und/oder Daten?	1	
Ist der Entwurf dokumentiert und folgt er einer im Unternehmen bekannten Methode?	1	
Ist die Codierung in einer im Unternehmen bekannten Programmiersprache ausgeführt?	1	
Ist jeder Modul getrennt änderbar und ersetzbar?	1	
Gibt es Hilfsmittel, mit denen die Folgen einer Änderung auf andere Module und Datenstrukturen untersucht werden können?	0,5	Manuelle Prüfung
Ist konstruktiv sichergestellt, dass bei ausreichender Prüfung der geänderten Elemente keine notwendigen Änderungen in anderen Teilen des Prüfobjektes übersehen werden?	1	
Gibt es Unterstützungsmittel bei der Kompilierung, die mögliche Syntax-, Datentypen- und Schnittstellenfehler prüfen?	0	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Ist Prüfung einzelner Module möglich?	1	
Gibt es Hilfsmittel, die den Testablauf überwachen und mit denen die Anzahl der Tests minimiert werden kann?	1	
Summe	9,00	

Tabelle 14. Checkliste für Qualitätsmerkmal Anpassbarkeit

Damit ergibt sich für das Qualitätsmerkmal Anpassbarkeit, nachfolgend als Q_A bezeichnet, folgendes:

$$Q_A = \frac{9}{12} \cdot 100$$
$$Q_A = 75\%$$

Diese Ergebnisse zeigen also, dass die untersuchten Qualitätsmerkmale mindestens zu dreiviertel erfüllt sind. Hier lässt sich jedoch einwenden, dass die geschätzten Werte keine objektiven Messwerte darstellen. Dennoch lassen sich Untergrenzen angeben, indem die geschätzten Werte in den Tabellen 13 und 14 auf den geringst möglichen Wert 0 gesetzt werden. Damit erhält man als Untergrenze für Q_I:

$$Q_I = \frac{9}{12} \cdot 100$$
$$Q_I = 75\%$$

und für Q_A:

$$Q_A = \frac{8}{12} \cdot 100$$
$$Q_A = 66,67\%$$

Somit sind die Qualitätsmerkmale Instandsetzung und Anpassbarkeit auch ohne Bezugnahme auf geschätzte Werte zu dreiviertel bzw. zweidrittel erfüllt.

Die hier untersuchten Kriterien wurden erst nach der Entwicklung des AGC aufgestellt, daher sind solch hohe Werte für Aspekte der Softwarequalität bemerkenswert und ein weitere Beleg für die hohe Qualität der AGC-Software.

11.9 Programmumfang

Aufgrund der Aufteilung in einzelne Programme lassen sich sowohl Anzahl als auch Umfang der einzelnen Programme in der Quantitätsanalyse erfassen. Abbildung 55 zeigt die Gesamtlänge der umfangreichsten zehn Programme im Colossus-Code. PINBALL GAME BUTTONS AND LIGHTS ist nicht nur mit dabei, sondern sogar auf Platz 1.

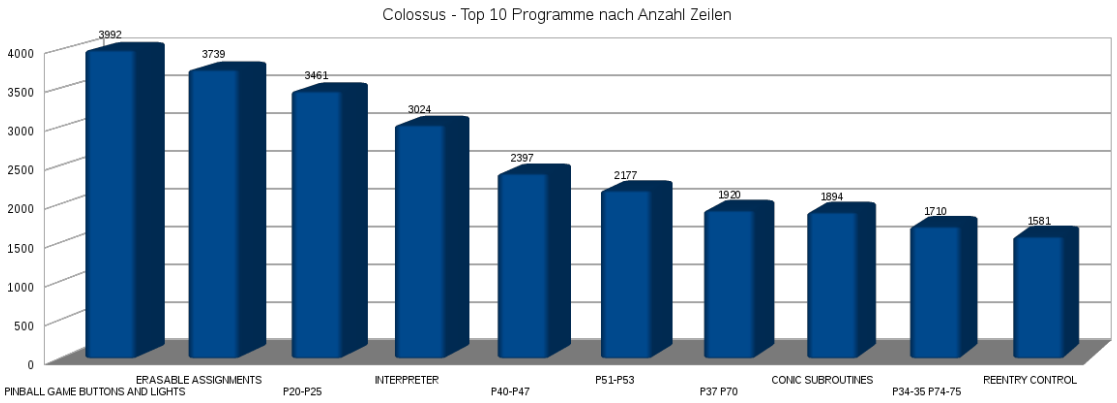


Abbildung 55. Die längsten zehn Programme in Comanche 055, nach Gesamtanzahl der Zeilen.

Allerdings sind hier alle Zeilen erfasst; sieht man sich nur die Codezeilen an, so erhält man ein etwas anderes Ergebnis. PINBALL GAME BUTTONS AND LIGHT ist dann „nur“ noch auf Platz zwei, Platz eins nimmt P20-P25 ein, welches in der Sortierung nach Gesamtzeilen auf Platz 3 liegt.

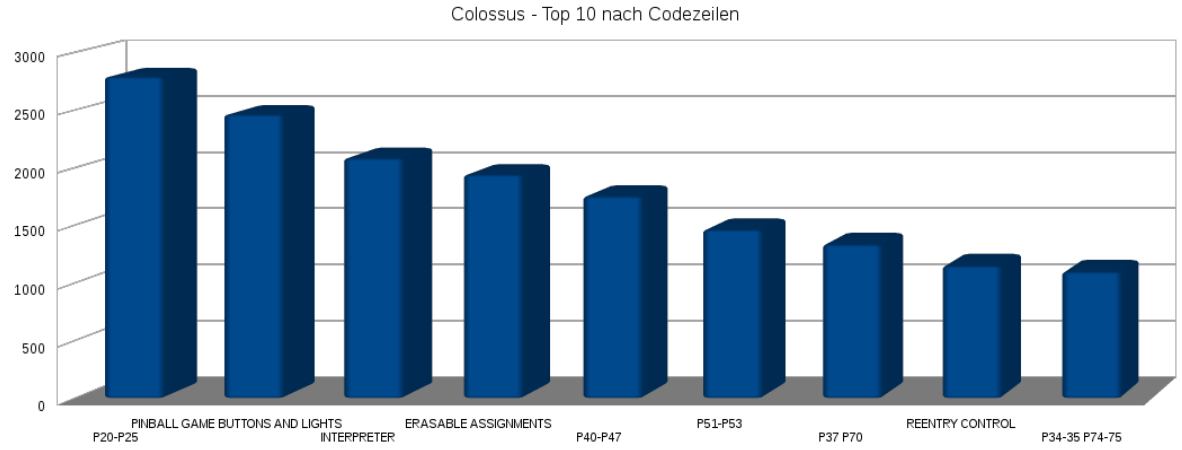


Abbildung 56. Die längsten zehn Programme in Comanche055, nach Anzahl Codezeilen.

Die Bezeichnung „P20-P25“ deutet bereits darauf hin, dass hier fünf Programme zusammengefasst sind. Zusammen bilden diese das Rendezvous-Programm und bilden im Code daher eine Einheit. Eine solche Art der Aufteilung findet sich in verschiedenen Teilen des Codes wieder, z.B. auch in den Programmen für den Wiedereintritt in die Erdatmosphäre, die aus Sicht der Besatzung zwar unterschiedliche Programme darstellen, aus Sicht des Computers jedoch eine Einheit, da sämtliche Programme für den Wiedereintritt aufeinander aufbauen und bei erfolgreichem Abschluss des jeweils vorangegangenen automatisch die nächste Phase (bzw. das nächste Programm) einleiten.

Das bereits mehrfach genannte und sehr umfangreiche Programm PINBALL GAME BUTTONS AND LIGHT ist somit bereits aufgrund seiner Größe ein guter Kandidat für weitergehende Analysen. Darüber hinaus ist dieses Programm im Luminary- und im Colossus-Code beinahe identisch, was auch Sinn macht, denn die Steuerung des DSKY ist ja in beiden Raumfahrzeugen gleich.

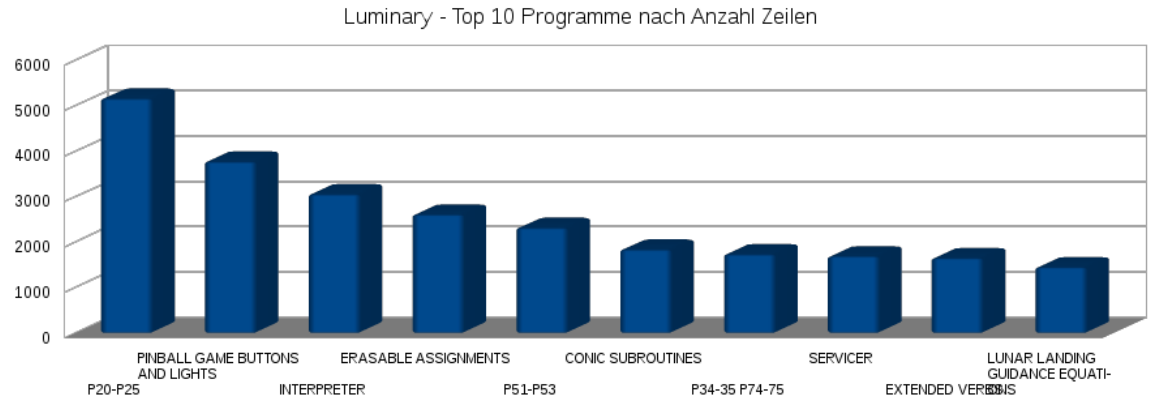


Abbildung 57. Die längsten zehn Programme in Luminary 099, nach Gesamtanzahl der Zeilen.

Ein Vergleich mit den Programmen in Luminary macht dies deutlich, auch dort findet man P20-P25 und PINBALL GAME BUTTONS AND LIGHT auf den ersten Plätzen. In Abbildung 57 sind die Programme nach der Gesamtzahl an Zeilen wiedergegeben.

11.10 Versionsvergleich

Der Code des AGC wurde während der Apollo-Missionen weiterentwickelt. Wie stark hat sich der Quellcode verändert? In Modulen wie dem DSKY-Steuerungsprogramm PINBALL GAME BUTTONS AND LIGHT wurden in Colossus 249 gegenüber Comanche055 nur wenige Zeilen geändert, was auch nachvollziehbar ist, da am DSKY selbst ja keine Änderungen vorgenommen worden sind. Die missionsspezifischen Programme wurden zwar durchaus stärker angepasst, jedoch werden auch darin die gleichen Standardmodule, z.B. für Navigation und Autopilot, wiederholt verwendet.

Welche Werte erhält man nun, wenn man den Linux-Kernel mit den gleichen Analysemethoden untersucht wie die AGC-Software?

11.11 Der Linux-Kernel

Der Linux-Kernel ist der zentrale Bestandteil (der Betriebssystemkern) von Linux-basierten Betriebssystemen, wie GNU/Linux oder auch Android. Beim Linux-Kernel handelt es sich um einen monolithischen Kernel, der preemptives Multitasking, virtuellen Speicher, shared libraries, die Internetprotokollfamilie, Memory Management und Threading unterstützt.

Der Linux-Kernel ist ein sehr umfangreiches Projekt mit zahlreichen Unterbereichen. Der Code ist frei verfügbar und beinhaltet Dateien unterschiedlichster Größe und Komplexität. Darüber hinaus existieren im Linux-Kernel Systemfunktionen, die auch funktionell mit Funktionen im AGC vergleichbar sind, wie z.B. das Watchdog-System. Diese Aspekte machen den Linux-Kernel zu einem idealen, modernen Vergleichsprojekt. Im Folgenden wird der Linux-Kernel in der Version 3.16.7-29 verwendet.

11.11.1 Quantität

Eine Messung des Umfangs mittels ohcount ergibt folgende Werte:

Language	**Files**	**Code**	**Comment**	**Comment %**	**Blank**	**Total**
c	36687	12138298	2472936	16.9%	2332446	16943680
assembler	1381	287920	65295	18.5%	47242	400457
cpp	549	205731	96579	31.9%	41926	344236
xml	195	76391	450	0.6%	5686	82527
make	1900	30847	7375	19.3%	7529	45751
perl	34	16309	2415	12.9%	3343	22067
python	23	4163	560	11.9%	733	5456
shell	85	4162	1725	29.3%	767	6654
tex	1	911	3	0.3%	108	1022
awk	7	668	85	11.3%	84	837
html	2	378	0	0.0%	58	436
scheme	1	218	0	0.0%	63	281
objective c	1	189	0	0.0%	55	244
autoconf	1	91	5	5.2%	15	111
xslt	6	70	27	27.8%	13	110
vim	1	27	12	30.8%	3	42
automake	3	20	0	0.0%	5	25
Total	40877	12766393	2647467	17.2%	2440076	17853936

Tabelle 15. OHCount-Ergebnisse für den Linux-Kernel 3.16.7-29

Das Tool CodeAnalyzer [23] kommt zu etwas anderen Werten, da es nicht mit allen Dateitypen umgehen kann, die im Linux-Kernel verwendet werden. Dennoch ist es durchaus für eine Analyse des Linux-Kernels einsetzbar, da Linux zum größten Teil in C geschrieben wurde. Und C/C++ sowie Assembler-Dateien kann der CodeAnalyzer verarbeiten. Die Ergebnisse für den Kernel sehen folgendermaßen aus:

File Name	Linux -Cumulative-
Total Files	37531
Total Lines	17319818
AVG Line Len	27
Total Code Lines	12368945
Total Comment Lines	3098273
Total WS Lines	2376153
Cd/C̆d Ratio	2,26
Cd/Cm Ratio	3,99
Cd/WS Ratio	5,21
% Code Lines	0,71
Code Lines/File	329
Comment Lines/File	82
WS Lines/File	63

Tabelle 16. CodeAnalyzer-Ergebnisse für den Linux-Kernel 3.16.7-29 (Zusammenfassung)

In der Tabelle 16 ist nur die Gesamtübersicht dargestellt, der komplette Report über alle analysierten Dateien des Kernels umfasst im Text-Format 6 MB.

11.11.2 Komplexität

Der Linux-Kernel ist aber nicht nur umfangreich, sondern auch von hoher Komplexität. Ein einfaches grep liefert bereits erste Hinweise: Im hier untersuchten Kernel sind mehr als 120.000 GOTO-Anweisungen enthalten. Genauere Ergebnisse erhält man durch Anwendung der Metriken, die in den vorherigen Abschnitten bereits auf den AGC-Code Anwendung fanden, insbesondere Halstead und McCabe. Zur Ermittlung der McCabe-Metriken lässt sich z.B. pmccabe einsetzen. Tabelle 17 zeigt die damit ermittelten Werte für die 10 Funktionen mit den höchsten (Traditional) McCabe-Werten:

Traditional McCabe	Modified McCabe	Funktion	Datei
1110	2	wm5110_readable_register	drivers/mfd/wm5110-tables.c
864	10	wm5102_readable_register	drivers/mfd/wm5102-tables.c
767	20	wm5100_readable_register	sound/soc/codecs/wm5100-tables.c
676	635	sym_getpciclock	drivers/scsi/sym53c8xx_2/sym_hipd.c
667	2	wm8997_readable_register	drivers/mfd/wm8997-tables.c
641	641	threefish_encrypt_1024	drivers/staging/skein/threefish_block.c
641	641	threefish_decrypt_1024	drivers/staging/skein/threefish_block.c
628	2	wm8962_readable_register	sound/soc/codecs/wm8962.c
559	559	build_backref_tree	fs/btrfs/relocation.c
351	7	wm2200_readable_register	sound/soc/codecs/wm2200.c

Tabelle 17. McCabe-Metriken für den Linux-Kernel 3.16.7-29 (Top-Ten nach Traditional McCabe)

In Tabelle 17 fallen nicht nur die hohen Werte auf, sondern auch, dass es sich bei den hier betrachteten Funktionen vor allem um Treiber, Kryptographiefunktionen und Codecs handelt. Bei den Kryptographiefunktionen ist es leicht nachvollziehbar, dass diese eine hohe Komplexität aufweisen. Ebenso können aber auch Gerätetreiber mit ihren oft sehr spezifischen Funktionen zum Ansprechen bestimmter Hardware hohe Komplexitäten erreichen. Für die hier an erster Stelle in der Liste auftretende Funktion „wm5110_readable_register“ trifft dies jedoch nicht zu. Diese Funktion besteht nur aus einer Anweisung! Dabei handelt es sich um eine switch-Anweisung mit 1109 case-Fällen. Dementsprechend hat auch die Modified McCabe Komplexität nur den Wert 2 (zu den Unterschieden zwischen Modified und Traditional McCabe siehe auch Abschnitt 11.6.3). Hier haben wir somit die extremste Abweichung zwischen Traditional McCabe und Modified McCabe im gesamten Linux-Kernel. Das Ergebnis zeigt aber auch, dass McCabe-Analysen allein nicht unbedingt brauchbare Aussagen zur Komplexität von Programmen liefern. Dies ist auch einer der Gründe, weshalb in dieser Arbeit ein vielschichtiger Ansatz zur Analyse des AGC-Codes gewählt wurde. Ermittelt man die 10 Funktionen mit den höchsten McCabe-Werten im Linux-Kernel nach Modified McCabe, so erhält man die in Tabelle 18 dargestellten Ergebnisse:

Modified McCabe	Traditional McCabe	Funktion	Datei
641	641	threefish_encrypt_1024	drivers/staging/skein/threefish_block.c
641	641	threefish_decrypt_1024	drivers/staging/skein/threefish_block.c
635	676	sym_getpciclock	drivers/scsi/sym53c8xx_2/sym_hipd.c
559	559	build_backref_tree	fs/btrfs/relocation.c
307	318	cifs_set_ops	fs/cifs/inode.c
289	289	threefish_encrypt_512	drivers/staging/skein/threefish_block.c
289	289	threefish_decrypt_512	drivers/staging/skein/threefish_block.c
281	290	ia_hw_type	drivers/atm/iphase.c
255	326	altera_execute	drivers/misc/altera-stapl/altera.c
253	253	MXL_TuneRF	drivers/media/tuners/mxl5005s.c

Tabelle 18. McCabe-Metriken für den Linux-Kernel 3.16.7-29 (Top-Ten nach Modified McCabe)

Hier fallen wiederum die Kryptographiefunktionen auf. Außerdem gibt es hier keine solch drastischen Abweichungen zwischen Modified McCabe und Traditional McCabe wie in Tabelle 17. Die Idee der pmccabe-Entwickler, eine modifizierte McCabe-Metrik mit anzubieten, ist also durchaus sinnvoll und ermöglicht es, bestimmte „Ausreißer“ in den analysierten Daten schnell zu finden. Tabelle 18 bietet somit eine realistischere Einschätzung der oberen Grenze an McCabe-Komplexität im Linux-Kernel, die mit Werten von über 600 jedoch immer noch sehr hoch ist.

Solche komplexen Funktionen sind aber, wenn auch sehr wichtig, sicher keine geeigneten Vertreter zur Einschätzung der Komplexität des Linux-Kernels insgesamt (wobei aufgrund der Unterschiedlichkeit der einzelnen Quellcode-Dateien die Ermittlung einer durchschnittlichen Komplexität über den gesamten Kernel-Code sicher ohnehin nur bedingt aussagefähig wäre).

Für vernünftige Aussagen ist daher ein systematischeres Vorgehen nötig. Daher wird zunächst ein Überblick über die Verzeichnisstruktur der Linux-Kernel-Quellen gegeben und anschließend einzelne Bereiche näher betrachtet, die aufgrund ihrer Größe und Struktur als potentielle Vergleichskandidaten in Frage kommen.

Die Kernel-Quellen sind auf Linux-Systemen unter /usr/src/linux zu finden (sofern die Kernel-Quellen installiert sind). Dieses Verzeichnis weist folgende Struktur auf:

Verzeichnis	**Inhalt**
arch	Architekturabhängige Dateien, z.B. für x86 oder s390
block	Treiber für blockorientierte Geräte, also Geräte, die Daten nicht in Datenströmen, sondern in Blöcken senden und empfangen
crypto	Kryptografiefunktionen
Documentation	Dokumentation der Kernel-Quellen
drivers	Treiber. Dieses Verzeichnis hat zahlreiche Unterverzeichnisse für die verschiedensten Geräteklassen, wie SCSI oder ISDN. Die Unterverzeichnisse sind oft wiederum in Unterverzeichnisse gegliedert. So gibt es z.B. direkt unter drivers keine Ordner für Tastatur- oder Maustreiber. Diese befinden sich im Unterverzeichnis input in den Ordnern keyboard bzw. mouse.
firmware	Firmware-Dateien zum Ansprechen von Geräten, die ihre Hardware mittels eigener Software, der Firmware, verwalten.
fs	Dateisysteme. Dieser Ordner enthält zahlreiche Unterordner für die einzelnen Dateisysteme, wie ext4 oder reiser.
include	Allgemeine Header-Dateien zum Einbinden in andere Kernel-Dateien
init	Dateien für den Systemstart. Die in diesem Verzeichnis zu findende Datei main.c enthält auch die Funktion start_kernel(void).
ipc	Dateien für die Interprozesskommunikation.
kernel	Dateien zur Steuerung des Kernels. Hier sind auch die Zeitfunktionen untergebracht.
lib	Kernel-Bibliotheken
mm	Speicherverwaltung (Memory Management)
net	Dateien für Netzwerkprotokolle, wie Ethernet oder auch ipv4 und ipv6
samples	Neue Module, die weiterer Entwicklung bedürfen und Programmierbeispiele
scripts	Skript-Dateien zur Kompilierung des Kernels
security	Sicherheitsrelevante Dateien
sound	Audio-Treiber
tools	Werkzeuge zur Interaktion mit dem Kernel
usr	Dateien zur Erzeugung des Kernel-Images vmlinuz

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Verzeichnis	Inhalt
virt	Dateien für die Virtualisierung

Tabelle 19. Hauptverzeichnisse der Linux-Kernel-Quellen

Aufgrund der unterschiedlichen Anwendungsgebiete der einzelnen Bereiche des Kernel-Codes finden sich auch unterschiedlichste Komplexitäten in den Dateien. Daher werden zunächst einzelne Bereiche daraufhin untersucht, inwieweit diese als potentielle Vergleichskandidaten zum AGC-Code Verwendung finden können. Kriterien hierbei sind sowohl ähnliche Funktionalitäten als auch ähnliche Größe bzw. ähnliche Anzahl an enthaltenen Funktionen.

Nach dem Kriterium der Funktionalität bieten sich hier die Kernel-Verwaltungsdateien in „kernel“ an, da sich hier zahlreiche systemrelevante Funktionen befinden, wie sie ähnlich auch im AGC zu finden sind. Hierzu zählt z.B. die Datei watchdog.c (zur Ermittlung von System-Lockups) und die Timer-Funktionen. Abbildung 58 zeigt die durchschnittlichen Werte dieser Dateien.

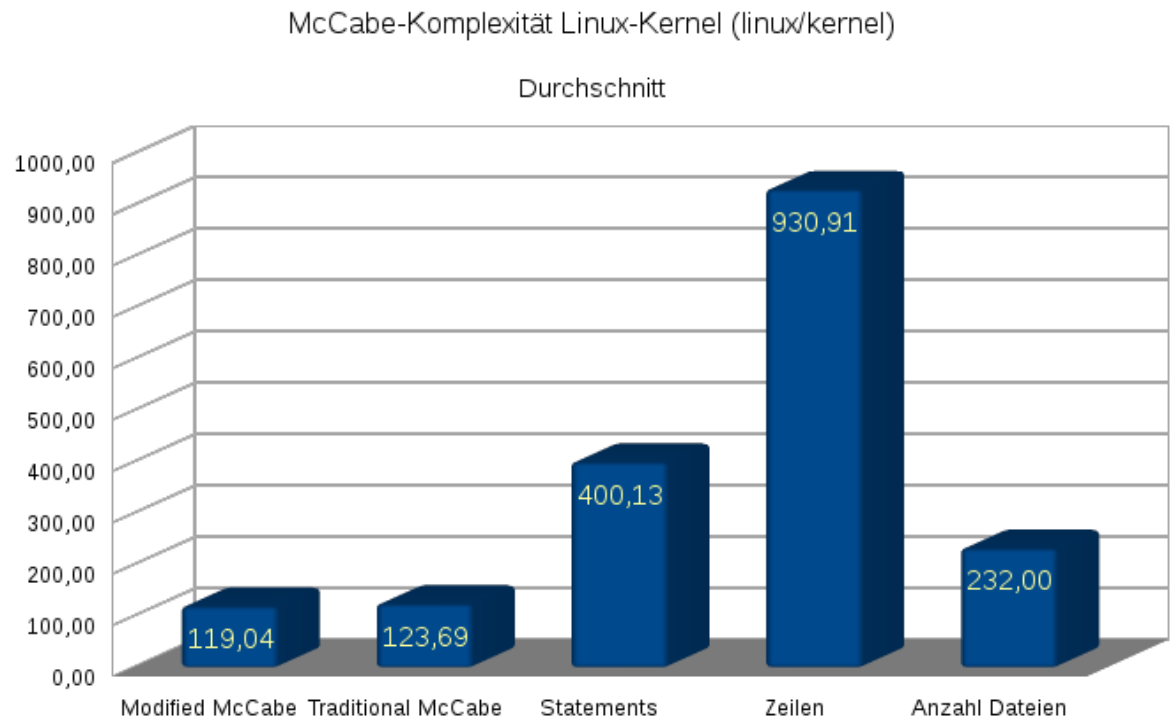


Abbildung 58. McCabe-Komplexität der Dateien unter kernel (Durchschnitt)

Welches sind nun unter „kernel“ die Funktionen mit den höchsten McCabe-Werten? Hier kommt wieder pmccabe zum Einsatz, Tabelle 20 zeigt die Ergebnisse.

Modified McCabe	Traditional McCabe	Anweisungen	Funktion	Datei (relativ zu „kernel“)
135	137	583	trace_seq_putmem_hex	trace/trace_output.c
120	126	221	unwind	unwind.c
63	63	97	__sched_setscheduler	sched/core.c
61	61	212	load_image_lzo	power/swap.c
59	100	152	audit_filter_rules	auditsc.c
54	54	241	copy_process	fork.c
50	52	103	kdb_parse	debug/kdb/kdb_main.c
49	49	136	SYSCALL_DEFINE5	events/core.c
48	62	194	audit_receive_msg	audit.c
45	45	150	vkdb_printf	debug/kdb/kdb_io.c

Tabelle 20. McCabe-Metriken für den Linux-Kernel 3.16.7-29 (Top-Ten aus „kernel“)

Das McCabe-Komplexitätsmaß misst zwar die Komplexität des Ablaufgraphen und ist somit auch nicht auf eine bestimmte Programmiersprache beschränkt, dennoch kann man sich natürlich fragen, ob hier nicht zu unterschiedliche Konzepte miteinander verglichen werden. Da der Linux-Code auch zahlreiche Assembler-Dateien enthält, die ihre grundlegenden Eigenschaften als Assembler-Programme mit den AGC-Dateien gemein haben, ist es naheliegend, Linux- und AGC-Assemblerdateien mit dem gleichen Algorithmus zu analysieren. Die beiden Programme McCabe_AGC (16.8) und McCabe_Assembler (16.9) sind zu diesem Zweck geschrieben worden. Diese Programme gehen ähnlich wie pmccabe vor, wobei McCabe_Assembler bedingte Sprünge in moderner Assembler-Syntax (genauer x86 und ähnliche Assembler) analysiert. Im Kernel befinden sich unter arch/x86 140 Assembler-Dateien. Diese weisen eine durchschnittliche McCabe-Komplexität von 18 pro Datei auf. Die Werte pro Funktion liegen in den allermeisten Fällen jedoch bei 1, siehe hierzu Abbildung 59.

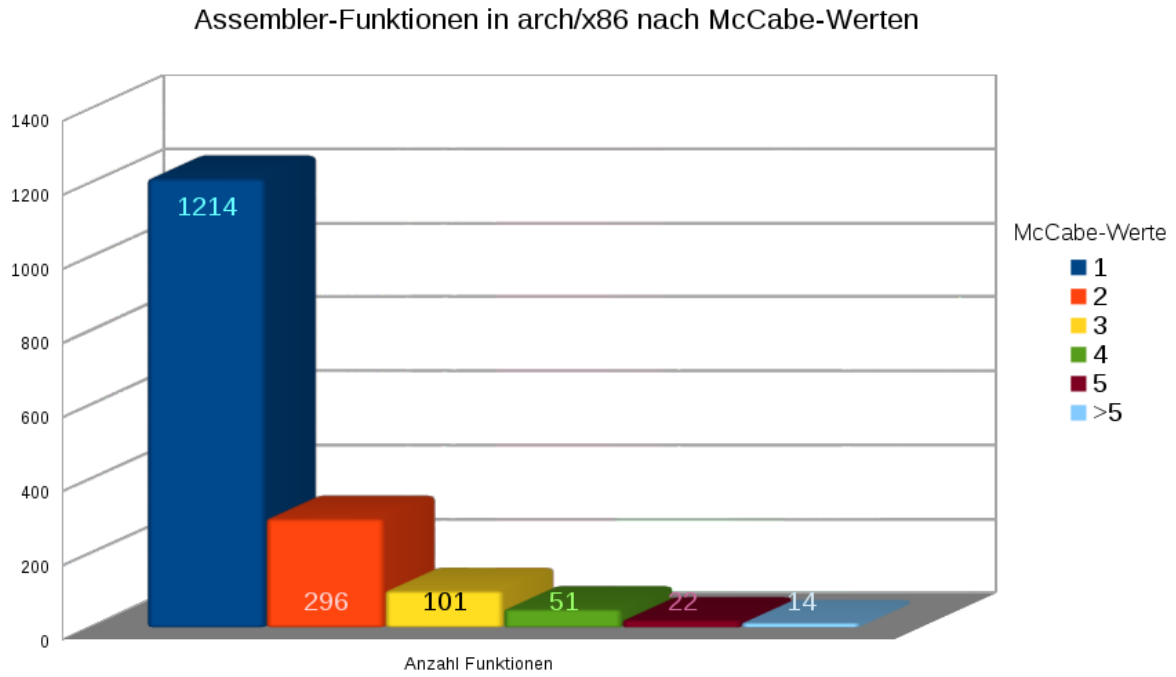


Abbildung 59. McCabe-Werte der Assembler-Dateien unter arch/x86 (Anzahl Funktionen)

Hier besteht also ebenfalls kein auffallender Unterschied zu den AGC-Dateien. Anzumerken ist noch, dass unter den Funktionen mit McCabe-Werten von über 5 zwei Funktionen enthalten sind, die Werte von 33 bzw. 49 aufweisen. Bei diesen Funktionen handelt es sich wiederum um Kryptographiefunktionen, genauer um die Implementierung des AES Algorithmus in Intel AES-NI Instruktionen und um den Galois-Counter-Mode, einem Blockverschlüsselungsmodus mit AES-Verschlüsselung [138]. Die Kryptographiefunktionen sind also auch hier wieder statistische Ausreißer; im AGC-Code spielen zumindest Ausreißer dieser Art keine Rolle, dort gibt es keine Kryptographiefunktionen.

Eine Betrachtung der Assembler-Dateien im Linux-Kernel führt also nicht zu grundlegend anderen Werten als bei den C-Dateien, was bei Anwendung der McCabe-Metrik auch nicht verwundern sollte, denn bei dieser wird ja genau dieser Ablaufgraph analysiert. Was jedoch auffällt, ist die hohe Anzahl von Funktionen mit geringer Komplexität; die Assembler-Funktionen sind größtenteils sehr linear aufgebaut, eine Eigenschaft, die auch bei den AGC-Funktionen zu finden ist (siehe auch Abschnitt 11.8.5).

Die Komplexität der Dateien des Linux-Kernels lassen sich natürlich auch nach Halsteads Methoden analysieren. In den Tabellen 21 und 22 sind die Halstead-Metriken für zwei ausgewählte Dateien dargestellt: Für initramfs.c, eine der grundlegenden Dateien des Linux-Kernels, und für unwind.c, die eine der komplexesten Funktionen unter „kernel“ beinhaltet (zur McCabe-Komplexität der Funktion unwind in unwind.c mittels pmccabe siehe Tabelle 20):

initramfs.c	
Operators count:	2086
Distinct operators:	52
Operands count:	1203
Distinct operands:	300
Program length:	3289
Program vocabulary:	352
Volume:	27823
Difficulty:	104
Effort:	2893599

Tabelle 21. Halstead-Metriken für init/initramfs.c

unwind.c	
Operators count:	6240
Distinct operators:	57
Operands count:	3676
Distinct operands:	627
Program length:	9916
Program vocabulary:	684
Volume:	93387
Difficulty:	164
Effort:	15315537

Tabelle 22. Halstead-Metriken für kernel/unwind.c

11.11.3 Funktionsgrößen

Aufgrund des Umfanges des Linux-Kernels finden sich in diesem sowohl sehr große als auch kleine Funktionen. Die Datei module.c hat mit 3963 Zeilen eine vergleichbare Größe wie PGBL (3991). In dieser Datei befinden sich 189 Funktionen mit einer durchschnittlichen Länge von 17,2 Zeilen pro Funktion, damit ähnelt module.c auch von Anzahl und Umfang der Funktionen her dem Programm PGBL (siehe auch 11.8.3).

11.12 Schlussfolgerungen aus den Analysen

Die Analysen der AGC-Software und die Vergleiche mit dem Linux-Kernel zeigen also, dass die AGC-Programme modernen Kriterien für Qualität und Komplexität standhalten. Die Herangehensweise der AGC-Entwickler war dabei jedoch an verschiedenen Stellen anders als die Herangehensweise bei modernen Projekten. So bildete z.B der AGC-Code – trotz Aufteilung in einzelne Module – ein zusammenhängendes Programm in dem nach Fehlern gesucht wurde. Die gewonnenen Erkenntnisse bei der AGC-Entwicklung flossen in die Qualitätssicherung späterer Softwareprojekte ein und sind damit ein Teil der Entwicklung, die zu den heutigen Methoden der Softwareentwicklung geführt haben. Die Sorgfalt, mit der die AGC-Entwickler vorgingen, zeigt auch, dass Vorwürfe, der AGC wäre ein wenig durchdachtes System, haltlos sind.

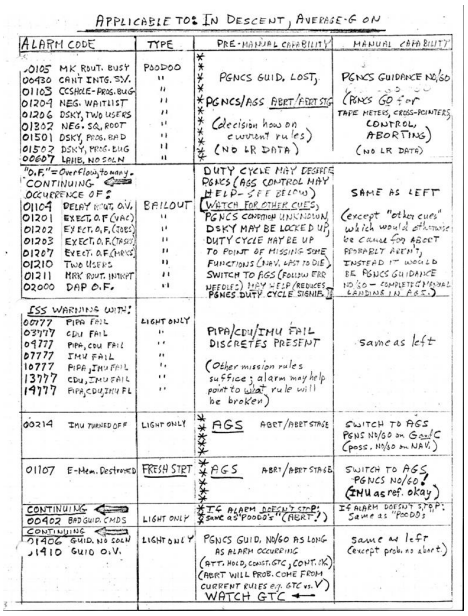
12 Ablauf der Mondlandungen

Bereits bei der ersten bemannten Mondlandung am 20. Juli 1969 wurde der AGC auf eine harte Probe gestellt. Die Art und Weise, wie der Computer dabei mit auftretenden Problemen umgegangen ist, zeigt, wie robust das Betriebssystem war.

12.1 Apollo 11

Drei Minuten vor der Mondlandung von Apollo 11 gab der AGC einen „1202"-Alarm, woraufhin die Astronauten bei Mission Control nachfragten. Der NASA-Softwareentwickler Jack Garman hatte sich alle Alarmcodes aufgeschrieben und kannte daher deren Bedeutung. „1202" bedeutete: „executive overflow – no core sets" (Nicht genug Zeit um alle anstehenden Aufgaben in Echtzeit auszuführen). In den Dokumentationen und Trainingsunterlagen zum Navigationssystem waren solche Alarmcodes natürlich zu finden (Siehe z.B. [1, Section 2-4]), doch gab es in Mission Control keine zentrale Stelle, bei der diese Informationen in den wenigen zur Verfügung stehenden Sekunden bis zu der Entscheidung, ob die Landung abgebrochen werden sollte oder stattfinden könnte, abrufbar gewesen wären.

Abbildung 60. Jack Garman [89]



APPLICABLE TO: IN DESCENT, AVERAGE-G ON

ALARM CODE	TYPE	PRE-NOMINAL CAPABILITY	MANUAL CAPABILITY
00105 MK RQUT. BUSY 00430 CANT INTG. SV. 01103 COSHKE-PROG. BUG 01204 NEG. WAITLST 01206 DSKY, TWO USERS 01302 NEG. SQ. ROOT 01501 DSKY, PROG. BAD 01502 DSKY, MNG-ENG 00607 UPLINK NO SCALN	P00D00 " " " " " " "	PGNCS GUID. LOST, PGNCS/ASS ABRT/ABRT STG (decision how on current rules) (NO LR DATA)	PGNCS GUIDANCE NO/GO (PGNCS GO + use TAPE METERS, CROSS-POINTERS, CONTROL, ABORTING) (NO LR DATA)
"O.F." = Overflow, to many - CONTINUING ⟸ OCCURENCE OF ? 01104 DELAY ROUT. OVF. 01201 EXECT. O.F. (VAC) 01202 EXECT. O.F. (CORE) 01203 EXECT. O.F. (CHAN) 01207 EXECT. O.F. (NEW) 01210 TWO USERS 01211 MRK ROUT. INTERT 02000 DAP O.F.	BAILOUT " " " " " " " "	DUTY CYCLE MAY DESIRE PGNCS (ASS CONTROL MAY HELP - SEE BELOW) (WATCH FOR OTHER CUES) PGNCS COMPUTR UNCERTAIN, DSKY MAY BE LOCKED UP, DUTY CYCLE MAY BE UP TO POINT OF MISSING SOME FUNCTIONS (NAV, ASS TO DIE), SWITCH TO ASS (FOLLOW PRO UPDATES) NOW HELP PRESERVE PGNCS DUTY CYCLE SOMEWHAT	SAME AS LEFT (except "other cues" which would ultimately be CRAWL FOR ABORT PROBABLY ABRT), INSTEAD IT WOULD BE PGNCS GUIDANCE NO/GO - COMPLETE/VOKAL LANDING IN P66)
ESS WARNING WITH: 00777 PIPA FAIL 03777 CDU FAIL 09777 PIPA, CDU FAIL 07777 IMU FAIL 10777 PIPA, IMU FAIL 13777 CDU, IMU FAIL 19777 PIPA, CDU, IMU FL	LIGHT ONLY " " " " " "	PIPA/CDU/IMU FAIL DISCRETES PRESENT (Other mission rules suffice; alarm may help point to what rule will be broken)	same as left
00214 IMU TURNED OFF	LIGHT ONLY	AGS ABRT/ABRT STAGE	SWITCH TO AGS PGNS NO/GO on G and C (POSS. NO/GO on NAV.)
01107 E-Mem. Destroyed	FRESH STRT	AGS ABRT/ABRT STAGE	SWITCH TO AGS PGNCS NO/GO (ENU as ref. okay)
CONTINUING ⟸ 00402 BAD GUID. CMDS	LIGHT ONLY	TC ALARM DOESN'T STOP; same as "P00D0S" (ABRT?)	IF ALARM DOESN'T STOP: Same as "P00D0"
CONTINUING ⟸ 01406 GUID. NO COLN 01410 GUID O.V.	LIGHT ONLY	PGNCS GUID. NO/GO AS LONG AS ALARM OCCURRING (ATT. HOLD, CONST. GTC, CONT. IK) (ABRT WILL PROB. COME FROM CURRENT RULES e.g. GTC vs. V) WATCH GTC ⟵	same as left (except prob. no abort)

Abbildung 61. Jack Garmans Checkliste mit den Fehlercodes [43]

Kurz nach den „1202"-Alarmmeldungen meldete der AGC auch noch „1201"-Alarme („executive overflow - no VAC areas", Keine Vector-Accumulator-Bereiche verfügbar). Jack Garman konnte aufgrund seiner Liste seinen Vorgesetzten, Steve Bales, sehr schnell informieren, dass die Alarme keine missionsgefährdenden Probleme darstellen würden, woraufhin Bales seinerseits das OK an Flight Director Gene Kranz gab. Daraufhin konnte die Landung erfolgen.

Die Alarme waren dadurch ausgelöst worden, dass Buzz Aldrin während des Landeanflugs das „RENDEZVOUS NAVIGATION PROGRAM 20" startete. Dieses Programm steuerte das Rendezvous-Radar und aktualisierte laufend die Daten für das Wiederandocken am CSM. Der Start des Programms P20 war somit eine zusätzliche Sicherheitsmaßnahme für einen eventuellen Missionsabbruch vor oder unmittelbar nach der Landung. Doch der Computer musste nun zusätzlich zu den Daten für den Landeanflug auch noch die Daten für das Rendezvousmanöver mit der Columbia berechnen, dies konnte der AGC nicht mehr in Echtzeit leisten. Dank Hal Lanings

Idee des prioritätengesteuerten Scheduling schob der AGC daher die Verarbeitung der Radardaten auf, so dass der Computer die Berechnungen für die eigentliche Landung problemlos durchführen konnte.

Es handelte sich bei den „1202“/„1201“-Alarmen daher nicht um Fehler, sondern um Hinweise, mit denen der AGC signalisierte, dass er Arbeiten, die derzeit nicht die höchste Priorität besaßen, zugunsten wichtigerer Arbeiten zurückstellte. Der Vorfall zeigt daher, dass die die Architektur des AGC-Betriebssystems gut durchdacht war.

12.2 Apollo 13

Apollo 13 startete am 11. April 1970 und sollte die dritte bemannte Mondlandung durchführen. Doch am 14. April explodierte ein Sauerstofftank im Service Module, eine Landung auf dem Mond war nun nicht mehr möglich und die Mission von Apollo 13 wurde nun plötzlich zu einer Rettungsmission, bei der Astronauten, Techniker und Computer gefordert waren, wie bei keiner Apollo-Mission zuvor. Das in Abbildung 24 gezeigte DSKY ist aus dem Command Module *Odyssey* von Apollo 13; gut zu erkennen ist die Abnutzung einzelner Tasten, insbesondere der Verb-Taste; der AGC wurde wirklich intensiv genutzt.

13 Das Erbe des Apollo-Programms

Die Auswirkungen der rasanten technischen Entwicklung während des Apollo-Programms sind bis heute zu spüren. Innerhalb weniger Jahre wurden Probleme gelöst, die vorher wenig bis gar keine Beachtung gefunden hatten. Dies hat auch auf dem Gebiet der Computertechnik für einen starken Entwicklungsschub gesorgt. In vielen Bereichen wurden neue Techniken eingeführt oder bereits vorhandene Techniken stärker verbreitet. So wurden z.B. die ersten integrierten Schaltkreise erst 1958 entwickelt [75], von der NASA dann aber bereits einige Jahre später in großem Umfang eingesetzt. Aufgrund dieses hohen Bedarfs an integrierten Schaltkreisen waren die Hersteller stärker gefordert, zuverlässige Produktionsverfahren zu entwickeln, die auch bei großen Stückzahlen problemlos mithielten. Neuere Verfahren und große Stückzahlen wiederum wirkten sich auch auf die Stückpreise aus; die ICs wurden kostengünstiger, was die Verbreitung weiter förderte.

Die Verbreitung der IC-Technologie ist damit eine direkte Auswirkung der *Entwicklung* des AGC, und auch der AGC selbst hat zahlreiche spätere Systeme direkt beeinflusst, wie z.B. das Fly-by-Wire und die Steuerung von Rettungs-U-Booten.

13.1 Fly-by-Wire

Bei der manuellen Steuerung von Flugzeugen werden die Steuerbewegungen des Piloten durch Hydrauliksysteme, Stahlseile oder andere mechanische Systeme auf die Steuerflächen des Flugzeugs übertragen. Im Gegensatz dazu werden beim Fly-by-Wire die Steuerbewegungen mittels elektrischer Signale übertragen, die wiederum Elektromotoren, Hydrauliksysteme oder andere Aktoren zur Bewegung der Steuerflächen steuern. Werden die Steuersignale vor der Weitergabe an die Aktoren noch durch einen Computer geleitet, kann dieser die Steuerbewegungen des Piloten daraufhin prüfen, ob die gewünschten Manöver das Flugzeug gefährden könnten (z.B. zu viel oder zu wenig Schub). Des Weiteren kann ein computergestütztes Fly-by-Wire-System wesentlich schneller als ein Mensch auf äußere Einflüsse, wie z.B. auf Luftturbulenzen, reagieren. Fly-by-Wire mit Computerunterstützung trägt damit sowohl zur Flugsicherheit als auch zur Entlastung des Piloten bei. Heutzutage ist Fly-by-wire zu einer Art Industriestandard für Passagierflugzeuge geworden, nahezu alle moderneren Verkehrsflugzeuge sind damit ausgestattet, wie z.B. die Boeing 777 [16], die Boeing 787 [15] oder die Airbus A320-, A330- und A340-Familien [3]. Doch es ist noch nicht sehr lange her, dass solche Fly-by-wire-Systeme mit digitaler Computerunterstützung in die Passagierluftfahrt Einzug hielten, dies war erst 1987 der Fall[5]. Das erste Passagierflugzeug mit vollständig digitalem, computergestützten Fly-by-wire-System war der Airbus A320, der im Februar 1987 vorgestellt wurde [2]. Doch

[5] Die Concorde hatte zwar bereits vorher ein Fly-by-wire-System, jedoch ein analoges [46].

bevor ein System zuverlässig genug ist, um in Passagierflugzeugen eingesetzt zu werden, ist natürlich umfangreiche Entwicklungsarbeit nötig; so war der Airbus A320 nicht das erste Flugzeug mit digitalem Fly-by-wire. Das war ein Testflugzeug der NASA, eine F-8, die am 25. Mai 1972 von der Edwards Air Force Base in Kalifornien startete [72, S. 73]. Der verwendete Computer war der Apollo Guidance Computer [72, S. 59ff]. Dies ist ein Beispiel dafür, wie der Apollo Guidance Computer auch außerhalb des Apollo-Programms großen Einfluss hatte; die heutigen Fly-by-wire-Systeme sind ein Erbe von Apollo, von dem bis heute (und vermutlich auch in der Zukunft) jeder Fluggast profitieren kann.

13.2 Deep-Submergence Rescue Vehicle

Bereits 1964 wurde für die US Navy ein Unterwasserrettungssystem entwickelt, das Deep-Submergence Rescue Vehicle (DSRV). Zur Steuerung bekam dieses System einen Navigationscomputer, der von Charles Drapers MIT Instrumentation Laboratory entwickelt werden sollte. Man kann sich denken, was geschah: Der AGC kam zum Einsatz.

Aufgrund der komplexen Berechnungen, die für die Unterwassernavigation nötig sind, wurden jedoch gleich zwei AGC zusammen im DSRV eingesetzt [29, S. 119ff]. Hier zeigt sich zum einen, dass der AGC auch auf anderen Gebieten als der Weltraum- bzw. Luftfahrtnavigation einsetzbar war, zum anderen aber auch, dass die Unterwassernavigation durchaus höhere Anforderungen an ein Navigationssystem stellen kann als die Weltraumnavigation.

13.3 Eingebettete Systeme

Der Apollo Guidance Computer war außerdem das erste moderne eingebettete System; Apollo hat dementsprechend auch auf den Gebieten der eingebetteten Systeme und der mobilen Computer den Weg für großangelegte Weiterentwicklungen geebnet. Daraus resultieren nicht nur mobile Computer, wie sie etwa moderne Smartphones darstellen, sondern auch die „Wearable Computers“, die in Kleidungsstücken und anderen Gebrauchsgegenständen integriert sind.

14 Technik aktueller NASA-Missionen

Wie sieht es bei den NASA-Missionen des 21. Jahrhunderts aus? In den 1980er-, 1990er- und 2000er-Jahren dominierte das Space Shuttle die bemannte Raumfahrt der NASA, so dass leicht der Eindruck entstehen konnte, das Space Shuttle würde die Zukunft menschlicher Exkursionen in den Weltraum darstellen. Das Shuttle konnte Menschen und Material jedoch nur in einen niedrigen Erdorbit bringen. Damit kein Missverständnis entsteht: Das ist kein Makel des Shuttles, denn es wurde von vornherein als Zubringer (eben als Shuttle) für den Erdorbit entwickelt. Aber das bedeutete auch, dass mit dem Shuttle keine ferneren Ziele als die niedrige Erdumlaufbahn erreicht werden konnten. Wie bereits in Abschnitt 8.1 angesprochen, bot Apollo-Saturn nicht nur die Möglichkeit, Menschen wesentlich weiter in den Weltraum zu befördern, sondern verfügte außerdem über mehr Sicherheitsfeatures als das Space Shuttle.

Abbildung 62. Orion Splashdown, 5. Dezember 2014 [116]

Die NASA entwickelt derzeit das Orion-Raumschiff für bemannte Missionen außerhalb des niedrigen Erdorbits. Abbildung 62 stammt nicht aus der Apollo-Ära, sondern wurde am 5. Dezember 2014 aufgenommen und zeigt eine Testwasserung der Orion-Raumkapsel. Die Verwandtschaft zu Apollo ist nicht zu übersehen und wird in Abbildung 63 vermutlich noch deutlicher. Diese zeigt die Orion-Raumkapsel nach dem Testflug.

Diese Verwandtschaft ist natürlich kein Zufall, In Orion fließen die erfolgreichen Techniken der Apollo-Missionen wieder ein. Die Apollo-Technik ist aus heutiger Sicht alt, hat sich aber als sehr erfolgreich erwiesen und bietet zahlreiche Aspekte (z.B. ein Launch-Escape-System, eine kompakte Raumkapsel, etc.), die zukünftige Missionen für sich nutzen können. Das Orion-System zeigt hier, wie sehr ein Blick in die Geschichte die zukünftige Entwicklung beeinflussen kann.

Und welchen Computer verwendet Orion? Zu der Zeit, zu der die bemannten Mondlandungen begannen, stellte der AGC längst nicht mehr den aktuellsten Stand der Computertechnik dar [53, S. 136]. Doch der AGC war

zuverlässig und sehr robust, was bei einem solch komplexen Unternehmen wie Apollo wesentlich bedeutender war, als die Verwendung der jeweils neusten Technik.

Im Orion-Raumschiff arbeitet ein IBM PowerPC 750 FX Prozessor, der 2002 veröffentlicht wurde und z.B. in Appples iMac eingesetzt wurde. Dessen Leistung ist mit einer Taktfrequenz von 800 MHz [67] gegenüber dem AGC zwar beeindruckend, doch seit dem Erscheinen des PowerPC 750 FX sind bereits mehr als 10 Jahre vergangen und die Entwicklung neuer Prozessoren ist nicht stehengeblieben. Heutige Mobiltelefone verfügen häufig bereits über Prozessoren mit ähnlicher oder sogar höherer Leistung.

Ebenso wie bei Apollo ist es auch bei Orion nicht das Ziel, eine möglichst neue, sondern eine möglichst zuverlässige Technik zu verwenden, wie Matt Lemke, Avionics Manager im Orion-Projekt [132], in einem Interview mit „Computerworld“ feststellt [45]. Der Orion-Computer ist auf Aspekte wie Strahlen- und Erschütterungsresistenz ausgelegt: Er verfügt über ein stabiles Gehäuse, eine dickere Leiterplatte und über einen Prozessor, der sehr strahlenresistent ist [45]. Somit sind auch in den Orion-Computer Aspekte der AGC-Entwicklung eingeflossen.

Zur Redundanz existiert der Orion-Computer gleich dreimal im Raumschiff. Dies ist ein weiterer Aspekt, der zeigt, wie ernst die Zuverlässigkeit von Computern in der bemannten Raumfahrt genommen wird.

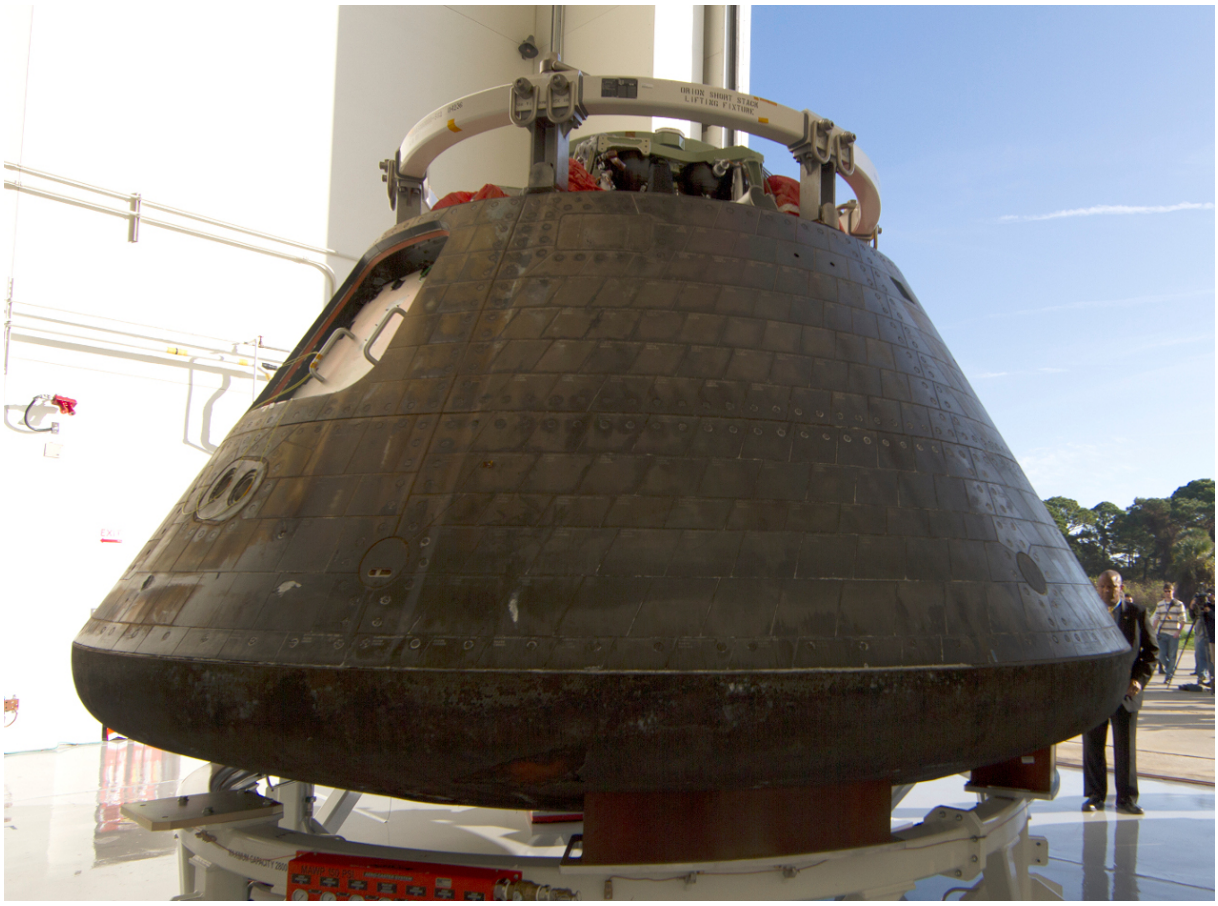


Abbildung 63. Orion nach Testflug, Aufnahme vom 6. Januar 2015 [116]

Noch etwas fällt beim Vergleich mit Apollo auf: Der Artikel in „Computerworld“ trägt den Titel „The Orion spacecraft is no smarter than your phone“. Personen, die die Echtheit der Apollo-Missionen anzweifeln, bringen u.a. das Argument, der AGC wäre nicht leistungsfähiger als ein Homecomputer der 1980er-Jahre und somit völlig ungeeignet für ein derart ambitioniertes Unternehmen wie eine bemannte Mondlandung. Bei Orion haben wir

die Situation, dass es bisher noch nicht einmal bemannte Flüge gab, das Projekt befindet sich ja noch im Entwicklungsstadium. Dennoch behauptet auch Matt Lemke nicht, man würde irgendeinen neuartigen Supercomputer verwenden oder einen solchen noch entwickeln. Er sagt ganz offen, dass der Orion-Computer nicht schneller als ein Mobiltelefon ist. Es geht darum, dass der Computer seine Aufgaben zuverlässig ausführt, nicht darum, dass er besonders schnell ist.

Wenn also nach einer bemannten Marslandung wiederum Zweifler an der Echtheit einer solchen Mission mit Vergleichen kommen und den Orion-Computer dann möglicherweise mit einer Armbanduhr oder einer elektronischen Krawattennadel vergleichen, kann man ihnen zurecht entgegenhalten, dass sie sich von Anfang an hätten besser informieren können.

15 Zusammenfassung

Das Ziel, Menschen zum Mond und sicher wieder zurück zu bringen, erforderte den Einsatz modernster Techniken in Verbindung mit hohen Zuverlässigkeitsanforderungen. Daher flossen in die Computer des Apollo-Programms sowohl völlig neuartige Konzepte als auch erprobte Technologien ein. Die Rechner des RTCC waren Hochleistungsmainframes – industrielle Großrechner – bei denen die wichtigsten Kriterien die Zuverlässigkeit und die Rechenleistung waren. Der RTCC bestand deshalb auch, wie dargelegt, gleich aus 5 gleichartigen Mainframes.

Für die Onboard-Navigation gab es noch keine Rechner, die leistungsfähig genug für die zu bewerkstelligende Aufgabe und dabei kompakt genug für den Einbau in ein Raumfahrzeug waren. Hier wurde daher eine komplette Neuentwicklung erforderlich, der Apollo Guidance Computer.

Der AGC war von vornherein auf Robustheit und Zuverlässigkeit ausgelegt, das zeigt sich auf der Hardwareseite z.B. durch die Verwendung von Core Rope Memory, auf der Softwareseite durch umfangreiches Testen und Kommentieren sowie durch die Aufteilung in Einzelprogramme und auch durch die Verwendung von Fehlertoleranztechniken. Von Beginn an verlief die Entwicklung des AGC dabei äußerst solide und war durch rigoroses Testen gekennzeichnet.

Zu den innovativen Neuheiten zählte der umfangreiche Einsatz von integrierten Schaltkreisen, die ja erst 1958 entwickelt worden waren, also nur wenige Jahre vor Beginn der AGC-Entwicklung. Obwohl die IC-Technologie noch neu und damit relativ unerprobt war, hat sie sich als zuverlässig erwiesen. Auf die Frage, ob der Einsatz einer solch neuen Technologie nicht den Zuverlässigkeitsforderungen widerspricht, lässt sich sagen, dass ohne ICs der AGC in der realisierten Form gar nicht hätte gebaut werden können. Alternativen wie z.B. Elektronenröhren hätten zu viel Raum und Gewicht gekostet, um einen flugtauglichen Computer von der Leitungsfähigkeit des AGC zu erstellen. Elektronenröhren hätten außerdem durch ihre hohe Wärmeentwicklung Raumfahrzeug und Besatzung gefährden können. Die Verwendung diskreter Transistoren wäre zwar auch möglich gewesen (und wurde auch bis 1962 verfolgt [53, S. 81ff]), doch boten ICs höhere Leistung bei geringerem Volumen [53, S. 82ff], was für den Einsatzzweck der Weltraumnavigation äußerst erstrebenswert war.

Eine weitere bedeutende Innovation war die Benutzersteuerung des AGC. Der AGC war einer der ersten Computer, der auf eine direkte Interaktion mit dem Benutzer ausgelegt war. Multics von Prof. Corbató wurde, wie dargelegt, etwa zur gleichen Zeit wie der AGC entwickelt. Multics war eine revolutionäre Neuerung, doch es zielte als Betriebssystem auf die Verwendung von Großrechnern ab, die bereits vorhanden waren und für Stapelverarbeitung genutzt wurden. Demgegenüber war der AGC von vornherein als interaktiv nutzbarer Computer entworfen worden. Ebenfalls völlig neuartig war das prioritätenbasierte Scheduling, das heute Standardverfahren zahlreicher Betriebssysteme ist. Die Prioritätensteuerung trug wesentlich zur Zuverlässigkeit des AGC bei.

Die vorgestellten Analysen des Apollo Guidance Computers zeigen außerdem, dass die eingesetzten Programme – Navigationssoftware – in ihren grundlegenden Werten keine deutlichen Abweichungen von „modernen“ Projekten aufweisen. Und das, obwohl die meisten der vorgestellten Analyse- und Entwicklungstechniken zu Beginn des Apollo-Programms noch gar nicht existierten. So stand den Entwicklern u.A. kein Linker zur Verfügung. Es zeigt sich somit, dass die Entwickler nicht nur sehr sorgfältig vorgegangen sind, sondern auch Aspekte berücksichtigt haben (z.B. Modularisierung), die später von erheblicher Bedeutung für die Computerentwicklung werden sollten. Die sorgfältige, pragmatische und empirische Vorgehensweise der Softwareentwickler des AGC reflektiert dabei das Vorgehen im gesamten Apollo-Programm. Der Vergleich mit modernen Methoden der Softwareentwicklung zeigt dadurch auch, dass solche Methoden kein Selbstzweck sind, sondern auf soliden und rationalen Grundlagen basieren und dem Entwickler helfen können, die zunehmende Komplexität beherrschbar zu halten.

Zuverlässigkeit wurde bei den Computern des Apollo-Programms auch durch moderne Methoden der Fehlertoleranz erreicht: Dreifach redundante Gatter im LVDC und strahlenresistente Hardware im AGC. Der AGC selbst war außerdem doppelt vorhanden, als LGC im Lunar Module und als CGC im Command Module. Das Betriebssystem des AGC verfügte über Prioritätensteuerung, die „Night Watchman“-Fehlererkennung, Restart Tables und den Rupt-Lock-Test als fehlerkorrigierende Bestandteile. Zur weiteren Ausfallsicherung existierte außerdem das Abort Guidance System. Navigationsberechnungen wurden ebenfalls sowohl in den Computern der Bodenstationen als auch an Bord des Raumfahrzeugs durchgeführt. Somit war auch hier Redundanz gegeben.

Ob sich die Entwicklungen der Computertechnik in den 1960er-Jahren gegenseitig beeinflusst haben, lässt sich jedoch nicht mit Bestimmtheit sagen. Aussagen, wie die von Prof. Corbató scheinen dieser These zu widersprechen. Dennoch sind Ähnlichkeiten erkennbar, z.B. in Bezug auf die Entwicklung neuer Mensch-Maschine-Interaktionsmöglichkeiten oder auch bei der Verwaltung von Computerressourcen mittels Betriebssystemen. In diesem Bereich ist daher weitere Forschung nötig.

Die Fly-By-Wire-Systeme sind demgegenüber jedoch eindeutig direkte Nachfolgeentwicklungen des AGC. Das erste Testflugzeug mit Fly-By-Wire verwendete ja, wie dargestellt, sogar den AGC.

Bereits zu Apollo-Zeiten erstellte die NASA Hilfsmittel zur Qualitätssicherung von Software-Projekten. Spätere Entwicklungen bauten auf solchen Projekten auf und führten zu zahlreichen Werkzeugen für die Softwareentwicklung und deren Qualitätssicherung.

Die Steuerung des DSKY wiederum ähnelt der Menüsteuerung moderner Systeme, dies kann jedoch auch darauf zurückzuführen sein, das sowohl die Entwicklung der DSKY-Steuerung als auch die Menüsteuerung auf Aspekte menschlicher Sprache, genauer auf die Unterteilung in Verben und Nomen, basieren. Daher sind hier parallele Entwicklungen durchaus möglich. Eine gemeinsame Grundlage könnten hier die Arbeiten von Noam Chomsky darstellen.

Die Zuverlässigkeit der im Apollo-Programm eingesetzten Techniken war, wie im Abschnitt über das Space-Shuttle beschrieben, in der Tat höher als die der im Space-Shuttle eingesetzten. Unfälle, wie der der Columbia im Jahre 2003, wären durch Anordnung des Besatzungsbereichs oberhalb des Trägerfahrzeugs zu verhindern gewesen.

Viele Aspekte der Navigationscomputer des Apollo-Programms flossen später zunächst in die Luftfahrttechnik, dann in die allgemeine Computerentwicklung ein, wie z.B. das prioritätenbasierte Scheduling. Dies zeigt auch, dass Apollo kein isoliertes Projekt war, sondern seinen Einfluss weit über den Bereich der bemannten Raumfahrt hinweg ausgedehnt hat. Auch dies lässt sich als Beleg für die Zuverlässigkeit der eingesetzten Technik werten, denn unzureichend funktionierende Komponenten hätten sich wohl kaum so weit verbreitet, wie es die Techniken aus dem Apollo-Programm taten (z.B. das beschriebene Fly-By-Wire).

Zusammenfassend lässt sich daher feststellen, dass das Apollo-Programm die Computertechnik in den unterschiedlichsten Bereichen vorangebracht hat, nicht nur auf dem Gebiet der Flug- oder Weltraumcomputer:

- Miniaturisierung und eingebettete Systeme
- Verbreitung der IC-Technologie
- Computergestützte Navigationssysteme - Fly-by-Wire
- Drahtlose und drahtgebundene Netzwerkkommunikation
- Prioritätengesteuertes Scheduling
- Softwareentwicklungsverfahren
- Sowie viele weitere Aspekte, die hier nicht behandelt worden sind (Bild- / Tonverarbeitung, ...)

Das Apollo-Projekt ist und bleibt eine technische Meisterleistung, die den Vergleich mit anderen technischen Großleistungen der Menschheit, wie dem Bau der Pyramiden (sowohl der ägyptischen als auch der mesoamerikanischen), nicht zu scheuen braucht und die viel dazu beigetragen hat, dass wir heute über eine enorme Anzahl der unterschiedlichsten Computer verfügen, über hochentwickelte Flugsicherungssysteme, über weltweite Kommunikationssysteme und über vieles mehr.

Damit ist das Apollo-Programm keineswegs nur ein spannender Abschnitt der Technikgeschichte, sondern erstreckt seinen Einfluss bis in die heutige Zeit und darüber hinaus!

16 Anhang

16.1 Liste der Sterne im AGC

Octal Star Code	Star / Planet	Visual Magnitude
1	Alpha Andromedae (Alpheratz)	2.1
2	Beta Ceti (Diphda)	2.2
3	Gamma Cassiopeiae (Navi)	2.2
4	Alpha Eridani (Achernar)	0.6
5	Alpha Ursae Minoris (Polaris)	2.1
6	Theta Eridani (Acamar)	3.4
7	Alpha Ceti (Menkar)	2.8
10	Alpha Persei (Mirfak)	1.9
11	Alpha Tauri (Aldebaran)	1.1
12	Beta Orionis (Rigel)	0.3
13	Alpha Aurigae (Capella)	0.2
14	Alpha Carinae (Canopus)	-0.9
15	Alpha Canis Maioris (Sirius)	-1.6
16	Alpha Canis Minoris (Procyon)	0.5
17	Gamma Velorum (Regor)	1.9
20	Iota Ursae Maioris (Dnoces)	3.1
21	Alpha Hydrae (Alphard)	2.2
22	Alpha Leonis (Regulus)	1.3
23	Beta Leonis (Denebola)	2.2
24	Gamma Corvi (Gienah)	2.8
25	Alpha Crucis (Acrux)	1.6
26	Alpha Virginis (Spica)	1.2
27	Eta Ursae Majoris (Alkaid)	1.9
30	Theta Centauri (Menkent)	2.3
31	Alpha Bootis (Arcturus)	0.2
32	Alpha Corona Boreal. (Alphecca)	2.3
33	Alpha Scorpii (Antares)	1.2
34	Alpha Trianguli Austr. (Atria)	1.9
35	Alpha Ophiuchi (Rasalhague)	2.1
36	Alpha Lyrae (Vega)	0.1
37	Sigma Sagittarii (Nunki)	2.1
40	Alpha Aquilae (Altair)	0.9
41	Beta Capricorni (Dabih)	3.2
42	Alpha Pavonis (Peacock)	2.1
43	Alpha Cygni (Deneb)	1.3
44	Epsilon Pegasi (Enip)	2.5
45	Alpha Piscis Austr. (Fomalhaut)	1.3
46	Sun	
47	Earth	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Octal Star Code	Star / Planet	Visual Magnitude
50	Moon	
00	Planet (Apollo 12 LM: Pollux)	

Tabelle 23. Liste der Sterne im AGC [92]

16.2 Wichtige Verbs und Nouns

Verb	Beschreibung
V25	N36E

Tabelle 24. Liste wichtiger Verbs und Nouns

16.3 Befehlssatz des AGC

Instruction	Beschreibung
AD	Add
ADS	Add to Storage
AUG	Augment
BZF	Branch Zero to Fixed
BZMF	Branch Zero or Minus to Fixed
CA	Clear and Add
CCS	Count Compare and Skip
CS	Clear and Subtract
DAS	Double Add to Storage
DCA	Double Clear and Add
DCS	Double Clear and Subtract
DIM	Diminish
DINC [6]	Diminish Absolute Value by 1
DV	Divide
DXCH	Double Exchange
EDRUPT	Ed Smally's Interrupt
EXTEND	Set Extracode Flag
INCR	Increment
INDEX	Index Next Instruction
INDEXE	Index Next Instruction Extended
INHINT	Inhibit Interrupts
LXCH	Exchange L and K
MASK	Mask A by K
MCDU [6]	CDU Decrement
MINC [6]	Counter Decrement
MP	Multiply
MSU	Modular Subtract

Fortsetzung auf nächster Seite

[6] Nicht vom Programmierer verwendbar („Nonprogrammed Sequence“)

Fortsetzung von vorheriger Seite

Instruction	**Beschreibung**
PCDU [6]	CDU Increment
PINC [6]	Counter Increment
QXCH	Exchange Q and K
RAND	Read and Mask
READ	Read KC
RELINT	Enable Interrupts
RESUME	Resume Interrupted Program
RETURN	Return From Subroutine
ROR	Read and Superimpose
RUPT [6]	Interrupt
RXOR	Read and Invert
SHANC [6]	Counter Shift and Add 1
SHINC [6]	Counter Shift
SU	Subtract
TC	Transfer Control
TCF	Transfer Control to Fixed
TS	Transfer to Storage
WAND	Write and Mask
WOR	Write and Superimpose
WRITE	Write Channel KC
XCH	Exchange A and K
XLQ	Execute Using L and Q
XXALQ	Execute Extracode Using A, L and Q

Tabelle 25. Assembler-Instruktionssatz des Apollo Guidance Computer

16.4 Liste der Programme in Colossus 2A Rev. 055 (Comanche 055)

Abschnitt	**Seiten**
CONTRACT AND APPROVALS	1
ASSEMBLY AND OPERATION INFORMATION	2-26
TAGS FOR RELATIVE SETLOC	27-35
SUBROUTINE CALLS	36
ERASABLE ASSIGNMENTS	37-130
INTERRUPT LEAD INS	131-132
T4RUPT PROGRAM	133-169
DOWNLINK LISTS	170-180
FRESH START AND RESTART	181-210
RESTART TABLES	211-221
SXTMARK	222-235
EXTENDED VERBS	236-267
PINBALL NOUN TABLES	268-284
CSM GEOMETRY	285-296

Fortsetzung auf nächster Seite

[6] Nicht vom Programmierer verwendbar („Nonprogrammed Sequence“)

Fortsetzung von vorheriger Seite

Abschnitt	Seiten
IMU COMPENSATION PACKAGE	297-306
PINBALL GAME BUTTONS AND LIGHTS	307-389
R60 62	390-398
ANGLFIND	399-411
GIMBAL LOCK AVOIDANCE	412-413
KALCMANU STEERING	414-419
SYSTEM TEST STANDARD LEAD INS	420-422
IMU CALIBRATION AND ALIGNMENT	423-455
GROUND TRACKING DETERMINATION PROGRAM	456-459
P34-35 P74-75	460-504
R31	505-510
P76	511-513
R30	514-524
STABLE ORBIT	525-532
P11	533-550
TPI SEARCH	551-561
P20-P25	562-634
P30-P37	635-648
P32-P33 P72-P73	649-683
P40-P47	684-736
P51-P53	737-784
LUNAR AND SOLAR EPHEMERIDES SUBROUTINES	785-788
P61-P67	789-818
SERVICER207	819-836
ENTRY LEXICON	837-843
REENTRY CONTROL	844-882
CM BODY ATTITUDE	883-889
P37 P70	890-933
S-BAND ANTENNA FOR CM	934-935
LUNAR LANDMARK SELECTION FOR CM	936
TVCINITIALIZE	937-944
TVCEXECUTIVE	945-950
TVCMASSPROP	951-955
TVCRESTARTS	956-960
TVCDAPS	961-978
TVCSTROKETEST	979-983
TVCROLLDAP	984-998
MYSUBS	999-1001
RCS-CSM DIGITAL AUTOPILOT	1002-1024
AUTOMATIC MANEUVERS	1025-1036
RCS-CSM DAP EXECUTIVE PROGRAMS	1037-1038
JET SELECTION LOGIC	1039-1062
CM ENTRY DIGITAL AUTOPILOT	1063-1092

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Abschnitt	**Seiten**
DOWN-TELEMETRY PROGRAM	1093-1102
INTER-BANK COMMUNICATION	1103-1106
INTERPRETER	1107-1199
FIXED FIXED CONSTANT POOL	1200-1204
INTERPRETIVE CONSTANTS	1205-1206
SINGLE PRECISION SUBROUTINES	1207
EXECUTIVE	1208-1220
WAITLIST	1221-1235
LATITUDE LONGITUDE SUBROUTINES	1236-1242
PLANETARY INERTIAL ORIENTATION	1243-1251
MEASUREMENT INCORPORATION	1252-1261
CONIC SUBROUTINES	1262-1308
INTEGRATION INITIALIZATION	1309-1333
ORBITAL INTEGRATION	1334-1354
INFLIGHT ALIGNMENT ROUTINES	1355-1364
POWERED FLIGHT SUBROUTINES	1365-1372
TIME OF FREE FALL	1373-1388
STAR TABLES	1389-1393
AGC BLOCK TWO SELF-CHECK	1394-1403
PHASE TABLE MAINTENANCE	1404-1413
RESTARTS ROUTINE	1414-1419
IMU MODE SWITCHING ROUTINES	1420-1448
KEYRUPT UPRUPT	1449-1451
DISPLAY INTERFACE ROUTINES	1452-1484
SERVICE ROUTINES	1485-1492
ALARM AND ABORT	1493-1496
UPDATE PROGRAM	1497-1507
RT8 OP CODES	1508-1516
GAP-generierte Tabellen	1517-1751

Tabelle 26. Programme und Routinen von Colossus 2A Rev. 055 (Comanche 055 - Apollo 11)

16.5 Liste der Programme in Luminary 1A Rev. 099

Abschnitt	**Seiten**
CONTRACT AND APPROVALS	1
ASSEMBLY AND OPERATION INFORMATION	2-27
TAGS FOR RELATIVE SETLOC	28-37
CONTROLLED CONSTANTS	38-53
INPUT OUTPUT CHANNEL BIT DESCRIPTIONS	54-60
FLAGWORD ASSIGNMENTS	61-88
GAP-generierte Tabelle	89
ERASABLE ASSIGNMENTS	90-152

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Abschnitt	**Seiten**
FINDCDUW–GUIDAP INTERFACE	908-925
P51-P53	926-983
LUNAR AND SOLAR EPHEMERIDES SUBROUTINES	984-987
DOWN TELEMETRY PROGRAM	988-997
INTER-BANK COMMUNICATION	998-1001
INTERPRETER	1002-1094
FIXED FIXED CONSTANT POOL	1095-1099
INTERPRETIVE CONSTANT	1100-1101
SINGLE PRECISION SUBROUTINES	1102
EXECUTIVE	1103-1116
WAITLIST	1117-1132
LATITUDE LONGITUDE SUBROUTINES	1133-1139
PLANETARY INERTIAL ORIENTATION	1140-1148
MEASUREMENT INCORPORATION	1149-1158
CONIC SUBROUTINES	1159-1204
INTEGRATION INITIALIZATION	1205-1226
ORBITAL INTEGRATION	1227-1248
INFLIGHT ALIGNMENT ROUTINES	1249-1258
POWERED FLIGHT SUBROUTINES	1259-1267
TIME OF FREE FALL	1268-1283
AGC BLOCK TWO SELF CHECK	1284-1293
PHASE TABLE MAINTENANCE	1294-1302
RESTARTS ROUTINE	1303-1308
IMU MODE SWITCHING ROUTINES	1309-1337
KEYRUPT UPRUPT	1338-1340
DISPLAY INTERFACE ROUTINES	1341-1373
SERVICE ROUTINES	1374-1380
ALARM AND ABORT	1381-1385
UPDATE PROGRAM	1386-1396
RTB OP CODES	1397-1402
T6-RUPT PROGRAMS	1403-1405
DAP INTERFACE SUBROUTINES	1406-1409
DAPIDLER PROGRAM	1410-1420
P-AXIS RCS AUTOPILOT	1421-1441
Q R-AXIS RCS AUTOPILOT	1442-1459
TJET LAW	1460-1469
KALMAN FILTER	1470-1471
TRIM GIMBAL CNTROL SYSTEM	1472-1484
AOSTASK AND AOSJOB	1485-1506
SPS BACK-UP RCS CONTROL	1507-1510
GAP-generierte Tabellen	1511-1743

Tabelle 27. Programme und Routinen von Luminary 1A Rev. 099 (Apollo 11)

16.6 Abkürzungen

Abkürzung	Beschreibung
AFJ	Apollo Flight Journal
AGC	Apollo Guidance Computer
AGS	Abort Guidance System
AIA	Apollo Image Atlas, Photographien der Apollo-Missionen
ALSJ	Apollo Lunar Surface Journal.
AOS	Acquisition of Signal. Der Teil des lunaren Orbits, in dem das CSM hinter dem Mond hervorkommt und damit wieder Funkverbindung zur Erde möglich wird.
C&I	Communications and Instrumentation
CAPCOM	Capsule Communications. Bodenpersonal in Houston (meist Astronauten), welche die Sprechfunkkommunikation zwischen Mission Control und Apollo-Besatzung durchführten.
CCATS	Command Communication and Telemetry System
CDU	Coupling Data Unit. Analog-Digital-Wandler, wandelt Daten von IMU (CM+LM), Radar (LM) und Sextant (CM) in digitale Signale für AGC um.
CGC	Command Module Guidance Computer
CM	Command Module. Der einzige Teil des Apollo-Raumschiffs, der zur Erde zurückkehrte.
CSM	Command & Service Module. Der Teil des Apollo-Raumschiffs, das während der Mondlandungen im lunaren Orbit verblieb.
CTSS	Compatible Time-Sharing System
DEDA	Data Entry and Display Assembly
DSKY	DISPLAY AND KEYBOARD
DSN	Deep Space Network
ECS	Environmental Control System
EVA	Extra-vehicular activity. Aktivitäten außerhalb eines Raumschiffs.
GET	Ground Elapsed Time. Zeit am Boden seit Start.
HEO	High Earth Orbit. Bereich für geostationäre Satelliten, ca. 36000 km über der Erde.
IBM	International Business Machines
IC	Integrated Circuit
IMU	Inertial Measurement Unit Eine durch drei kardanische Aufhängungen stabilisierte Plattform mit Beschleunigunssensoren (PIPA) inertiale Referenzplattform (IRIG).
IRIG	Inertial Reference Integrating Gyro. Inertiale Referenzplattform.
JPL	Jet Propulsion Laboratory.
LEO	Low Earth Orbit. Bereich von etwa 160 bis etwa 2000 km.
LGC	Lunar Module Guidance Computer
LM	Lunar Module.
LOS	Loss of signal. Der Bereich des lunaren Orbit, in dem das CSM sich hinter dem Mond befindet und deshalb keine Funkverbindung zur Erde hat.
LRV	Lunar Roving Vehicle.
MET	Mission Elapsed Time. Zeit seit Start.
MSFN	Manned Space Flight Network

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Abkürzung	**Beschreibung**
NACA	National Advisory Committee for Aeronautics
NASA	National Aeronautics and Space Administration.
NOAA	National Oceanic and Atmospheric Administration.
PGNS	PRIMARY NAVIGATION AND GUIDANCE SUBSYSTEM
PIPA	Pulsed Integrating Pendulum Accelerometer.
PLSS	Portable Life Support System
RTCC	Real-Time Computer Complex
S-IC	Erste Stufe der Saturn V
S-II	Zweite Stufe der Saturn V
S-IVB	Dritte Stufe der Saturn V, enthält LM , bevor dieses an das CSM angedockt wird.
SLT	Solid Logic Technology
STS	Space Transport System (Space Shuttle)
TEC	Trans-Earth Coast. Die Reise vom Mond zur Erde.
TEI	Trans-Earth Injection. Schubmanöver, das das Apollo-Raumschiff aus dem lunaren Orbit heraus in Richtung Erde bringt, dies findet gewöhnlich hinter dem Mond statt.
TLC	Trans-Lunar Coast. Die Reise von der Erde zum Mond.
TLI	Trans-Lunar Injection. Schubmanöver, das das Apollo-Raumschiff aus dem Erdorbit heraus in Richtung Mond bringt.
TMC	Tabulating Machine Company. Vorläufer von IBM
TPS	Thermal Protection Subsystem
UCD	Urine Collection Device
USVMS	Urine Sample Volume Measuring System Unit
VAB	Vertical Assembly Building

Tabelle 28. Liste relevanter Abkürzungen

16.7 Analyseprogramm zur Berechnung der Halstead-Metriken

Der Quelltext des Analyseprogramms zur Berechnung der Halstead-Metriken des AGC-Codes.

```
/*
 * halstead_AGC.c
 *
 * Copyright 2016 M. Seidel
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
```

```
 *
 *
 */

# include <unistd.h>
# include <stdio.h>
# include <string.h>
# include <stdlib.h>

# include <glib.h>
# include <math.h>

# define ARRAY_LEN 20
# define STR_LEN 200
# define COMMENT_CHAR "#"

extern int find_frequencies(char* filename, char** instr_array);

char *argumente[1000]={"-06250","-2","0","1","2","3","4","5","6","7","10","11","12",
    "15","16","20","21","22","23","25",
"28","33","34","37","40","41","42","50","62","73","104","115","144","175","240","244
    ","377","764","1206","1217","1501",
"1727","1740","2000","2003","2104","2476","3240","3656","3740","3777","4000","5605",
    "5675","7675","10422","13557","13560",
"16001","16040","16314","23147","23346","30000","31463","32445","37016","37100","
    37600","40650","41126","55000","60000",
"73777","75377","76000","77743","77753","77772","77774","-ON",".16384","+8D","+
    DECSGN","+LIMIT","+ON","02/PIN","07/PIN",
"1.666666666E-4B12*","11DSPIN","13-11,1","1E-5B14","2.222222222B-2","20BIAS","21/22
    REG","2BLANK","2INTOUT","2K+3","2ROUND",
"2ROUND+2","33DEC","34DEC","40/PIN","41/PIN","42/PIN","5.555555555B-3","59.99SEC","
    59MIN","5BLANK","5BLANK+2","89TEST","A",
"ABCLOAD","ABLOAD","ACCEPTWD","ALL3DEC","ALLDC/OC","ALMCYCLE","ALOAD","AROUT1SF","
    ARTHINSF","ARTIN1SF","ARTOUTSF","BANKCALL",
"BANKJUMP","BBANK","BIASCOM","BINCON","BINROUND","BIT1","BIT10","BIT11","BIT12","
    BIT13","BIT14","BIT15","BIT15/14","BIT2",
"BIT3","BIT4","BIT5","BIT6","BIT7","BITSOFF","BITSOFF1","BLANKCON","BLANKCON+1","
    BLANKDSP","BLANKDSP+2","BLANKSUB","BLNKBBNK",
"BLNKSUB1","BLOAD","BOTHSGN","BRNCHCON","BUF","CADRSTOR","CCSHOLE","CHAN12","CHAN13"
    ,"CHANDSP","CHANLOAD","CHAR","CHARALRM",
"CHARALRM+2","CHARIN2","CHKP00H","CHRPRIO","CLEAR","CLEAR1","CLOAD","CLPASHI","
    CLPASS","CLR5","CODE","COMPICK-1","COMPTEST",
"COMPTST1","CONUMNOR","COUNT","CRITCON","CYL","CYR","DCOMPTST","DCTSTCYC","DECBRNCH"
    ,"DECDSP","DECDSP3","DECEND","DECON",
"DECOUNT","DECRET","DECROUND-1","DECTEM","DECTEST","DECTOBIN","DEGCOM","DEGCON1","
    DEGCON2","DEGINSF","DEGINSF2","DEGOUTSF",
"DEGOUTSF+1","DEGTAB","DISPLACE","DISTEM","DMP","DOPROC","DOPROC+2","DOTERM","DOUBLK
    ","DOUBLK+2","DP1OUTSF","DP2OUTSF",
"DP3OUTSF","DPFRACIN","DPFRACOT","DPINCOM","DPINORM","DPINSF","DPINSF+2","DPINSF2","
    DPINSF4","DPOSMAX","DPOUT","DPTEST",
"DPTEST1","DSALMOUT","DSEXIT","DSLV","DSMAG","DSMSK","DSP2BIT+10D","DSP2DEC","DSPA",
    "DSPAB","DSPABC","DSPABORT","DSPALARM",
```

```
"DSPB","DSPC","DSPCOM1","DSPCOM2","DSPCOM2+2","DSPCOM3","DSPCOUNT","DSPDC2NR","
    DSPDC2NR+3","DSPDCEND","DSPDCEND+2","DSPDCGET",
"DSPDCPUT","DSPDCWD1","DSPDCWD1-1","DSPDECVN","DSPDECWD","DSPDPDEC","DSPFMEM","DSPIN
    ","DSPIN1","DSPIN1-1","DSPLIST","DSPLOCK",
"DSPMM","DSPMMJB","DSPMMTEM","DSPMSK","DSPOCTIN","DSPOCTWO","DSPRND","DSPSFNOR","
    DSPSIGN","DSPTAB","DSPTAB+1","DSPTAB+11D",
"DSPTAB+4","DSPTAB+6","DSPTEM1","DSPTEM1+1","DSPTEM1+2","DSPWDRET","DSREL","EBANK","
    EDOP","ELEVEN","ELRCODE1","ENDALL","ENDALM+1",
"ENDBLFF","ENDBSUB1","ENDDPDEC+1","ENDECVN","ENDINST","ENDMONDO+1","ENDNMTST","
    ENDNUM","ENDNVSB1+1","ENDOFJOB","ENDRDLO+1",
"ENDRELDS","ENDRQWT+1","ENDRTOUT","ENDRUTIN","ENDSCALE","ENDSCALE-1","ENDSPF+1","
    ENDSPMIN+1","ENDSPMM+1","ENTER","ENTERJMP",
"ENTEXIT","ENTPAS0","ENTRET","ENTSET","ENTSET-2","ERCNT","ERCOM","ERCON","ERPLUS","
    ERROR","EXITEM","FAILREG","FAILREG+1",
"FAILREG+2","FALTON","FBANK","FFTAG8","FINDVAC","FIVE","FIXRANGE","FLASHOFF","
    FLASHON","FORCEV25","FOUR","FULLDSP","FULLDSP1",
"GETCOMP","GETINREL","GL+NOATT","GOALMCYC","GODSPALM","GOEXTVB","GOQ","GOVNUPDT","
    GTSFIN","GTSFINLC","GTSFOUT","GTSFOUTL","HI5",
"HIMINCON","HISECON","HITEMIN","HITEMOUT","HMSIN","HMSOUT","HRCON","HRCON1","
    IDAD1TEM","IMODES30","IMODES33","INREL","INRELTAB",
"INTMCTBS","INTMCTBS+2","ISCADR+0","ISLIST+0","JOBSLEEP","JOBWAKE","KILLMON","
    KILMONON","L","L14/OUT","LEFT5","LEFTNCOM","LEGALTST",
"LIMITCOM","LOADLV","LOADSTAT","LOC","LOCCTR","LODNNLOC","LODNNTAB","LODSAMPT","
    LOTEMOUT","LOW10","LOW11","LOW5","LOW7","LOW8","LOW9",
"LOWSUPER","LOWVERB","LST2CON","LST2FAN","M/SCON1","M/SCON2","M/SCON3","M/SCON3+1","
    M/SLIMIT","M/SNORM","M/SNORM+1","M/SOUT","MD1",
"MID5","MID7","MID7+1","MINCON","MINCON1","MINCON2","MINCON2+1","MIXAD","MIXBR","
    MIXNN1","MIXNN2","MIXNOUN","MIXTEMP","MMADREF",
"MMCHANG","MMCHANG+1","MODREG","MODROUTB","MONADR","MONBACK","MONBUSY","MONDEL","
    MONDO","MONIT2","MONITOR","MONREF","MONREQ","MONSAVE",
"MONSAVE1","MONSAVE2","MORNUM","MORNUM+1","MPAC","MPAC+1","MPAC+2","MPAC+3","MPAC+4"
    ,"MPAC+5","MPAC+6","MPACTST","MPTEMP","ND1",
"NEG.2","NEG1","NEG180","NEG2","NEGOPT","NEGSGN","NNADTEM","NNTYPTEM","NORMADR","
    NOTBIT12","NOUN","NOUNADD","NOUNCADR","NOUNREG",
"NOUNTEM","NOUNTEST","NOUT","NOVAC","NUM","NVBNKTEM","NVCOM","NVQTEM","NVSBBBNK","
    NVSBCOM","NVSBENDL","NVSBENDL+1","NVSBWT1","NVSUB1",
"NVSUB2","NVSUBB","NVSUBEND","NVSUBEND+2","NVSUBSY1","NVSUBUSY+3","NVTEMP","OCT1400"
    ,"OCT31","OCT34BAR","OCT55000","OCTBACK","ONE",
"OPDEGOUT","OPTDEGIN","OPTMODES","P00DOO","PASTEOPT","PASTEVB","PINBALL1","PINBALL2"
    ,"PINBALL3","PINBALL4","PINBRNCH","PINSUPBT",
"POSEC","POSGN","POSMAX","POSTJUMP","PREDSPAL","PRENVBSY","PRIO15","PRIO16","
    PRSHRTMP","PUTADD","PUTADD+1","PUTCOM","PUTCOM2","PUTCOM2-4",
"PUTDCSF2","PUTDECSF","PUTDPCOM","PUTNORM","PUTSFNOR","Q","R1D1","R2D1","R3D1","
    RDLONOR","READLO","READLO1","RECAL1","RECAL2","RECALTST",
"RELDSP","RELDSP1","RELDSP2","RELDSPON","RELRET","RELTAB","REQADD","REQCOM","REQDATX
    ","REQDATY","REQDATZ","REQEX1","REQMM","REQRET",
"REQUESTC-1","RIGHT5","RND/TST","RNDCON","RNDCON-1","RUTMXTEM","SAMPTIME","SCOUTEND"
    ,"SECON1","SECON2","SEPMIN","SEPMNRET","SEPSCRET",
"SEPSEC","SEPSEC1","SEPSECNR","SEPSECNR+1","SETAUG","SETEBANK","SETNADD","SETNCADR",
    "SETNCADR+1","SETVAC","SEVEN","SFAIL","SFCONUM",
"SFINTABR","SFOUTABR","SFRUTMIX","SFRUTNOR","SFTEMP1","SFTEMP2","SGNCOM","SGNOFF","
    SGNON","SGNTAB-2","SGNTO1","SGNTST1","SHOLTS","SHORTMP",
```

```
"SHORTMP+1","SIGNFIX","SIGNTEST","SINBLANK-2","SIX","SIZETST","SLAP1","SLEFT5","SR",
    "SUPDACAL","SUPDXCHZ","SUPDXCHZ+1","SUPERBNK",
"TASKOVER","TCFINDVC","TCNOVAC","TCWAIT","TEM4","TEN","TESTBIT","TESTNN","TESTOFUF",
    "TESTOFUF+4","THREE","TIME2","TPAGREE","TPLEFTN",
"TPSL1","TSTAB","TSTCON1","TSTCON2","TSTCON3","TSTFORDP","TSTLTS1","TSTLTS2","
    TSTLTS3","TSTLTS4","TWO","UNSUSPEN","UPDAT1","UPDAT1+2",
"UPDATNN-1","UPDATRET","UPDATVB","UPDATVB-1","USEADD","V37","VBFANDIR","VBPROC","
    VBPROC+1","VBRELDSP","VBRESEQ","VBRQEXEC","VBRQWAIT",
"VBSP1LD","VBSP2LD","VBSP3LD","VBTERM","VBTSTLTS","VD1","VERB","VERBFAN","VERBFAN-2"
    ,"VERBREG","VERBSAVE","VERBTAB","VNDSPCON","WAITLIST",
"WDAGAIN","WDAGAIN+5","WDCNT","WDRET","WHOLECON","XREG","XREGLP","XREGLP-2","YREG","
    YREGLP","Z","ZERO","ZREG","ZREGLP"};

char *befehle[100]={"=","2CADR","2DEC","2DEC*","AD","ADRES",
"ADS","BANK","BBCON","BLOCK","BZF","BZMF","CA","CADR","CAF","CCS",
"COM","COUNT","CS","DAS","DCA","DCS","DEC","DIM","DOUBLE","DXCH","EBANK=",
"EQUALS","EXTEND","GENADR","INCR","INDEX","INHINT","LXCH","MASK","MP","OCT",
"QXCH","RAND","READ","RELINT","ROR","SBANK=","SETLOC","SU","TC","TCF","TS","WAND",
"WOR","WRITE","XCH"};

// Array-Funktionen
int count_elements(char *array[])
{       int i=0;
        for (i =0; array[i] != NULL; i++)
        {
           // printf("%s,%d\n",array[i],i);
        }
        return i;

}

int find(char *array[], char *instr )
{
    int found=-1;
    int i;
    for (i=0; array[i] != '\0';i++)
    {
       if (strcmp(array[i],instr) == 0)
       {
            found=i;
            break;
        }
    }
    return found;
}

// String im Array finden
int gfind(GArray* a,char * str) {
 char * comp;
 int i;
 int found =-1;
 for (i = 0; i < a->len; i++)
 {
```

```
    comp =  g_array_index(a, char *, i);
    //printf("Debug: comp: %s, str: %s \n", comp,str);
    if (strcmp(str,comp)==0)
    {
        found =0;
        break;
    }
 }
    return found;
}

// Restliche Eingabezeile überspringen
void skip_to_end_of_line(FILE *ip)
{
    while (getc(ip) != '\n')
    {
    }
}

// Sekunden in Tage, Stunden und Minuten umrechnen
char * calculate_time(float s)
{
    float minutes_f;
    float hours_f;
    float days_f;
    char time[100];
    char * result;
    char * days_s = "Tag";
    hours_f = fmod(s/3600.0,24);
    minutes_f = fmod(s/60,60);
    days_f = trunc(s/86400 );
    if (days_f > 1)
    {
        days_s="Tage";
    }

    if (days_f >= 1)
    {
       sprintf(time,"%2.0f %s, %02.0f:%02.0f Stunden",days_f,days_s,hours_f,
          minutes_f);
    }
    else
    {
        sprintf(time,"%02.0f:%02.0f Stunden",hours_f,minutes_f);
    }
    result = strdup(time);
    return result;
}
void print_help_text_and_exit(char * program_name, int exit_condition)
{
            printf("%s%s%s\n", "Verwendung: ", program_name, " -f CODE-FILE [-c]");
            printf("%s\n","CODE-FILE: ein AGC-Quellcode");
            printf("%s\n","Option -c: Häufigkeitsanalyse der Befehle durchführen.");
```

```
            printf("%s\n","Option␣-h:␣Dieser␣Hilfetext.");
        exit(exit_condition);
}

int main(int argc, char **argv)
{
    // Schalter für zusätzliche Berechnung der Häufigkeiten
    int calculate_frequencies =0;
    // Optionen
    int opt=0;

    // Counter-Variablen
    int counter_comments=0;
    int counter_befehlsvorkommnisse=0;
    int counter_argumentenvorkommnisse =0;
    int untersuchte_befehle =0;
    int untersuchte_argumente =0;

    // Berechnungs-Variablen
    int program_length=0,program_vocabulary=0;
    float calculated_program_length=0.0;
    float volume=0.0,difficulty=0.0,effort=0.0,niveau=0.0,program_complexity=0.0;
    float time=0.0, bugs=0.0;

    // Ein/Ausgabe-Strings
    char *filename;
    char *str;
    char *cmp;
    char *time_s;

    // Vergrö ß erbare Glib-Arrays zur Aufnahme der gefundenen Befehle/Argumente
    GArray* verwendete_argumente = g_array_new(FALSE, FALSE, sizeof(char*));
    GArray* verwendete_befehle = g_array_new(FALSE, FALSE, sizeof(char*));
    // Glib-Unsigned-Int-Variablen für Länge der gefüllten Arrays
    guint anzahl_verwendeter_befehle =0;
    guint anzahl_verwendeter_argumente=0;

    // exit (0); // Debug
    FILE  *fp_code_file;
    if (argc < 3 ) // Prüfung, ob dem Programm überhaupt Argumente übergegen wurden
    {
        print_help_text_and_exit(argv[0],EXIT_SUCCESS);
    }
        // Überprüfung der Schalter
       while ((opt = getopt(argc, argv, "f:c")) != -1) {
       switch (opt) {
       case 'f':
           filename = strdup(optarg);
           break;
        case 'c':
            calculate_frequencies =1;
            break;
       default: /* '?' */
```

```
            print_help_text_and_exit(argv[0],EXIT_SUCCESS);
        }
    }
    // Fehlerhafte Verwendung der Option -c abfangen
    if (argc < 4 && calculate_frequencies)
    {
        print_help_text_and_exit(argv[0],EXIT_SUCCESS);
    }


    // Speicherfehler abfangen
    if ((str = (char *) malloc(STR_LEN)) == NULL)
    {
        goto exit_mem_failure;
    }
    // Dateiöffnungsfehler abfangen
    if ((fp_code_file = fopen(filename, "r")) == NULL)
    {
        fprintf(stderr, "%s\"%s\".\n", "Fehler beim Öffnen der Datei ", filename);
        print_help_text_and_exit(argv[0],EXIT_FAILURE);
    }
        // Gesamtzahl vorkommender Operatoren und Operanden zählen
        while (fscanf(fp_code_file, "%s", str) != EOF)
        {
            if (strcmp(str, COMMENT_CHAR) == 0 || strstr(str,COMMENT_CHAR) != NULL)
                // Nach Kommentarzeichen rest der Zeile überspringen
            {
                // strstr für Kommentarzeichen direkt am Word
                counter_comments++;
                skip_to_end_of_line(fp_code_file);
            } else
            {
                if (find(argumente,str) != -1) // Prüfen, ob aktuell eingelesenes
                    Word ein Argument ist
                {
                    counter_argumentenvorkommnisse++;
                    cmp = strdup(str);
                        if (gfind(verwendete_argumente,cmp) != 0) // Prüfen, ob
                            schon im Array,
                        {
                            g_array_append_val(verwendete_argumente,cmp); // Wenn
                                nicht im Array, anhängen
                        }
                        else
                        {
                        }
                }
                if (find(befehle,str) != -1) // Prüfen, ob aktuell eingelesenes Word
                     ein Befehl ist
                {
                        counter_befehlsvorkommnisse++;
                        cmp = strdup(str);
```

```
                        if (gfind(verwendete_befehle,cmp) != 0)   // Prüfen, ob
                            schon im Array,
                        {
                            g_array_append_val(verwendete_befehle,cmp); // Wenn
                                nicht im Array, anhängen
                        }
                        else
                        {
                        }
                }
            }
        }
        rewind(fp_code_file);
    // Halstead-Metriken
    // Untersuchte/verwendete Befehle/Argumente erfassen
    anzahl_verwendeter_argumente = verwendete_argumente->len; //n1
    untersuchte_argumente=  count_elements(argumente);
    untersuchte_befehle=  count_elements(befehle);
    anzahl_verwendeter_befehle =  verwendete_befehle->len; // n2

    // Berechnungen
    // Vokabular: verwendete Argumente + verwendete Befehle
    program_vocabulary = anzahl_verwendeter_argumente+anzahl_verwendeter_befehle; //
         n

    // Komplexität: (argumente / argumentenvorkommnisse) * (befehle /
        befehlsvorkommnisse)
    program_complexity = ((float)anzahl_verwendeter_argumente / (float)
        counter_argumentenvorkommnisse) * ((float) anzahl_verwendeter_befehle / (
        float) counter_befehlsvorkommnisse);

    // Länge: Gesamtanzahl verwendeter Argumente + verwendeter Befehle
    program_length = counter_argumentenvorkommnisse+counter_befehlsvorkommnisse; //
        N^

    // Halstead-Länge
    calculated_program_length = (float) anzahl_verwendeter_argumente * log2f((float)
        anzahl_verwendeter_argumente)  + (float) anzahl_verwendeter_befehle * log2f
        ((float) anzahl_verwendeter_befehle);

    // Volumen: Länge * log2(Vokabular))
    volume = (float)program_length * log2f((float)program_vocabulary);

    // Schwierigkeit: verwendete unterschiedliche Befehle / 2 * Argumente insgesamt
        / unterschiedliche Argumente
    difficulty = ((float) anzahl_verwendeter_befehle / 2.0) * ((float)
        counter_argumentenvorkommnisse / (float) anzahl_verwendeter_argumente);

    // Aufwand
    effort = difficulty*volume;

    // Niveau
    niveau = 1.0/difficulty;
```

```
    // Zeitbedarf nach Halstead
    time = effort/18.0;

    // Ausgelieferte (Halstead-) Fehler
    bugs = (pow(effort,(2.0/3.0)))/3000.0;

    calculate_time(time);
    if (calculate_frequencies)
    {
    find_frequencies(filename,befehle);
    }
    printf("Halstead-Metriken für %s\n",filename);
    printf("Untersucht auf %d Operatoren (Befehle) und %d Operanden (Argumente)\n",
        untersuchte_befehle,untersuchte_argumente);
    printf("-----------------------------------------\n");
    printf("Anzahl Befehle gesamt: %d\n",counter_befehlsvorkommnisse);
    printf("Anzahl Argumente gesamt: %d\n",counter_argumentenvorkommnisse);
    printf("Anzahl Kommentare: %d\n",counter_comments);
    printf("Anzahl unterschiedlicher Operanden: %d\n",anzahl_verwendeter_argumente);
    printf("Anzahl unterschiedlicher Operatoren: %d\n",anzahl_verwendeter_befehle);
    printf("Komplexität: %2.5f\n",program_complexity);
    printf("Programmlänge (Implementierungslänge): %d\n",program_length);
    printf("Programmvokabular: %d\n",program_vocabulary);
    printf("Berechnete Programmlänge (Halstead-Länge): %2.2f\n",
        calculated_program_length);
    printf("Programmvolumen: %2.2f\n",volume);
    printf("Schwierigkeit: %2.2f\n",difficulty);
    printf("Effort: %2.2f\n",effort);
    printf("Niveau: %2.5f\n",niveau);
    printf("Programmierzeit: %2.2f Sekunden\n",time);
    if (time > 3600)
    {
        time_s=calculate_time(time);
        printf("Programmierzeit: %s\n",time_s);
    }
    printf("Ausgelieferte Fehler nach Halstead: %2.2f\n",bugs);
    printf("-----------------------------------------\n");
    return 0;
exit_mem_failure:
    printf("Nicht genug Speicher, um den Puffer zu allokieren\n");
    exit(EXIT_FAILURE);
}
```

16.8 McCabe-Analyseprogramm für AGC-Code

```
!  McCabe_AGC.f90
!
!  Copyright 2016
!> @author M. Seidel \n
!
!  This program is free software; you can redistribute it and/or modify
```

```
!  it under the terms of the GNU General Public License as published by
!  the Free Software Foundation; either version 2 of the License, or
!  (at your option) any later version.
!
!  This program is distributed in the hope that it will be useful,
!  but WITHOUT ANY WARRANTY; without even the implied warranty of
!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!  GNU General Public License for more details.
!
!  You should have received a copy of the GNU General Public License
!  along with this program; if not, write to the Free Software
!  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
!  MA 02110-1301, USA.
!
!> @brief Berechnung von McCabe-Metriken für Quellcodedateien des Apollo Guidance
    Computers (AGC).
!> @details Nach dem Einlesen der Quelldatei werden Sprungmarken (Labels) gezählt,
!! und der Code wird untersucht auf das
!! Vorhandensein von Sprunganweisungen: \n
!!  + TC - Transfer Control \n
!!  + TCF - Transfer Control to Fixed \n
!!  + BZMF - Branch on Zero or Minus to Fixed \n
!!  + CCS - Count, Compare and Skip \n
!! CCS bietet vierfache Fallunterscheidung: >0, <0, = -0, =+0
!! (Zwei Werte für 0 sind vorhanden, da der AGC im Einerkomplement rechnet)\n
!! Aufgrund der Fallunterscheidungen wird die Anzahl der CCS-Anweisungen
!! für die Berechnung der McCabe-Werte daher mit 4 multipliziert.\n
!! TC und TCF werden zwar gezählt und ausgegeben, fließen aber nicht in die \\
!! Berechnung der McCabe-Werte ein, da es sich bei TC und TCF um unbedingte \\
!! Sprünge handelt. Die Sprungmarken fließen mit 1 in die Berechnung ein, \\
!! da es sich hierbei um Einsprungpunkte handelt und somit um den Beginn \\
!! einer Funktion.\\
!! Die übrigen Sprunganweisungen fließen jeweils mit 1 in die Berechnung ein.\\
!! Anhand der Sprungmarken werden die ermittelten Werte den jeweiligen Funktionen
!! zugeordnet.

program McCabe_AGC
    implicit none
    integer, parameter :: LINE_LEN = 130
    integer, parameter :: ARRAY_LEN = 2000
    character (len=*), parameter :: SCAN_SET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
        "
    character (len=LINE_LEN) :: filename, filename_trim, line, linecomplete
    integer :: char1
    integer :: mccabe = 0, label = 0, tc = 0, tcf = 0, bzmf = 0, ccs=0, jmp=0
    integer :: fehler, i, line_count = 0
    integer , dimension(1:ARRAY_LEN) :: tc_array = 0, tcf_array = 0, &
    & bzmf_array = 0, ccs_array = 0, jmp_array = 0, mccabe_array = 0
    character (len=9), dimension(1:ARRAY_LEN) :: label_array = ""
    real :: quotient_j_c, prozent_j_c,quotient_l_c,prozent_l_c,function_len_avrg
    character (len=2) :: option
    logical :: b = .false.
    call get_command_argument(1,filename)
```

```
    if (len_trim(filename) < 1 .or. filename .eq. "-h") then
        call print_help_text_and_exit()
    end if
    call get_command_argument(2,option)
    if (option .eq. "-b") then
        b = .true.
    end if
    ! Gekürzter Dateiname für Ausgabe
    filename_trim = filename(index(filename,"/",.true.)+1:)
    ! Datei Öffnen
      OPEN (10, FILE = filename, STATUS = 'OLD', IOSTAT = fehler)
!     Fehlermeldung, wenn Datei nicht existiert
      IF (fehler /= 0) then
        write(*,*) 'Datei ', trim(filename), ' existiert nicht!'
        call exit (0)
      end if
    ! Schleife - Zeile lesen
   DO
   read (10, '(A)', end=999) linecomplete
        ! write (*,*) len_trim(linecomplete), linecomplete(1:1)
        ! Codezeilen zählen:
        ! Codezeilen beginnen nicht mit kommentarzeichen und
        ! enthalten alphanumerische Zeichen
        if (linecomplete(1:1) .ne. "#" .and. scan(linecomplete,SCAN_SET) > 0) then
            line_count = line_count +1
        end if
   ! Nur den nicht auskommentierten Bereich analysieren
   if (index(linecomplete,"#") .ge. 1 ) then
        line = linecomplete(1:index(linecomplete,"#"))
   else
        line = linecomplete
   end if
   ! Erstes Zeichen festhalten für Überprüfung:
   ! Wenn Wortzeichen, dann handelt es sich
   ! um eine Sprungmarke (Label)
    char1 = iachar(line(1:1))
   ! Label zählen
   if ((char1 > 42) .and. (char1 < 91)) then
     ! write (*,*) line(1:1), iachar(line(1:1))
      ! write (*,*) line(1:index(line,"\t"))
       label = label +1
       label_array(label)=line(1:index(line,"\t"))
    end if
    ! TC zählen
    if (index(line,"TC\t") > 0 ) then
        tc_array(label) = tc_array(label) +1
    end if
    ! TCF zählen
    if (index(line,"TCF\t") > 0 ) then
        tcf_array(label) = tcf_array(label) +1
     end if
    ! BZMF zählen
    if (index(line,"BZMF\t") > 0 ) then
```

```
        bzmf_array(label) = bzmf_array(label) +1
     end if
    ! CCS zählen
    if (index(line,"CCS\t") > 0 ) then
        ccs_array(label) = ccs_array(label) +1
     end if
     mccabe_array(label) =  &
     & bzmf_array(label) + 4 * ccs_array(label) + 1
     jmp_array(label) = tc_array(label) + tcf_array(label) + &
     & bzmf_array(label) + 4 * ccs_array(label)
   END DO
999 continue
    tc = sum(tc_array(1:label))
    tcf = sum(tcf_array(1:label))
    bzmf = sum(bzmf_array(1:label))
    ccs =sum(ccs_array(1:label))
    jmp = sum(jmp_array(1:label))
    quotient_j_c = real(jmp) / real(line_count)
    prozent_j_c = quotient_j_c * 100
    quotient_l_c = real(label) / real(line_count)
    prozent_l_c = quotient_l_c * 100
    function_len_avrg = 1.0 / quotient_l_c
    if (.not. b) then
        write (*,"(5X,A)") "Werte␣pro␣Funktion␣(Sprungmarken␣(Label)):" !
            Sprungmarken, Label
    end if
    write (*,'(5A9,A12,A10,A7,A)') "TC" , "TCF", "BZMF", "CCS", "McCabe", "Sprunganw
        .", "Funktion","Datei:",filename_trim
    do i=1,label
        write (*,"(5I9,I12,1X,A12)")  tc_array(i), tcf_array(i), bzmf_array(i), &
        ccs_array(i), mccabe_array(i), jmp_array(i), label_array(i)
    end do
        if (.not. b) then
        write (*,"(5X,A,A,/5(A8,I4),/2X,A28,I4,A22,I4)") "Gesamtwerte␣für␣",
            filename_trim, "TC:", tc, "TCF:", tcf, "BZMF:", bzmf, &
        & "CCS:", ccs, "McCabe:", sum(mccabe_array), "Sprunganweisungen␣gesamt:",
            sum(jmp_array), &
        & "Funktionen␣(Label):", label
        write (*,"(5X,A11,I6)") "Codezeilen:", line_count
        write (*,"(5X,A41,F8.2)") "Verhältnis␣Sprunganweisungen/Codezeilen:",
            quotient_j_c
        write (*,"(5X,A9,F6.2,1X,A)") "Entpricht␣", prozent_j_c , "Sprunganweisungen
            ␣auf␣100␣Codezeilen"
        write (*,"(5X,A32,F8.2)") "Verhältnis␣Funktionen/Codezeilen:", quotient_l_c
        write (*,"(5X,A9,F6.2,1X,A)") "Entpricht␣", prozent_l_c , "Funktionen␣auf␣
            100␣Codezeilen"
        write (*,"(5X,A,F6.2,1X,A)") "Durchschnittliche␣Funktionslänge:␣",
            function_len_avrg, "Zeilen"
    end if
end program McCabe_AGC

!> Hilfetext bei nicht angegebener Eingabedatei
subroutine print_help_text_and_exit()
```

```
character * 130 :: progname
call get_command_argument(0,progname)
    write (*,*) "Benutzung: ",trim(progname)," Dateiname [-b]\n"
    write (*,*) "Beschreibung:"
    write (*,*) "Berechnung von McCabe-Metriken für Quellcodedateien des Apollo
        Guidance Computers (AGC)."
    write (*,*) "Nach dem Einlesen der Quelldatei werden Sprungmarken (Labels) gezä
        hlt,"
    write (*,*) "und der Code wird untersucht auf das Vorhandensein von
        Sprunganweisungen: "
    write (*,*) " + TC   - Transfer Control "
    write (*,*) " + TCF  - Transfer Control to Fixed "
    write (*,*) " + BZMF - Branch on Zero or Minus to Fixed "
    write (*,*) " + CCS  - Count, Compare and Skip  "
    write (*,*) "CCS bietet vierfache Fallunterscheidung: >0, <0, =-0, =+0"
    write (*,*) "(Zwei Werte für 0 sind vorhanden, da der AGC im Einerkomplement
        rechnet)"
    write (*,*) "Aufgrund der Fallunterscheidungen wird die Anzahl der CCS-
        Anweisungen"
    write (*,*) "für die Berechnung der McCabe-Werte daher mit 4 multipliziert.\n"
    write (*,*) "TC und TCF werden zwar gezählt und ausgegeben, flie ß en aber nicht
        in die "
    write (*,*) "Berechnung der McCabe-Werte ein, da es sich bei TC und TCF um
        unbedingte "
    write (*,*) "Sprünge handelt. Die Sprungmarken flie ß en mit 1 in die Berechnung
        ein, "
    write (*,*) "da es sich hierbei um Einsprungpunkte handelt, die als Beginn einer
         "
    write (*,*) "funktionalen Einheit Funktion angesehen werden können "
    write (*,*)
    write (*,*) "Die übrigen Sprunganweisungen flie ß en jeweils mit 1 in die
        Berechnung ein."
    write (*,*) "Anhand der Sprungmarken werden die ermittelten Werte den jeweiligen
        Funktionen "
    write (*,*) "zugeordnet."
    write (*,*)
    write (*,*) "Option -b: Keine Überschrift und keine Zusammenfassung "
call exit (0)
end subroutine
```

16.9 McCabe-Analyseprogramm für Assembler-Code

```
!  McCabe_Assembler.f90
!
!  Copyright 2016
!> @author M. Seidel \n
!
!  This program is free software; you can redistribute it and/or modify
!  it under the terms of the GNU General Public License as published by
!  the Free Software Foundation; either version 2 of the License, or
!  (at your option) any later version.
!
```

```
!  This program is distributed in the hope that it will be useful,
!  but WITHOUT ANY WARRANTY; without even the implied warranty of
!  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!  GNU General Public License for more details.
!
!  You should have received a copy of the GNU General Public License
!  along with this program; if not, write to the Free Software
!  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
!  MA 02110-1301, USA.
!
!> @brief Berechnung von McCabe-Metriken für Assembler (x86 und ähnliche)
!> @details Nach dem Einlesen der Quelldatei werden Sprungmarken (Labels) gezählt,
!! und der Code wird untersucht auf das
!! Vorhandensein von Sprunganweisungen: \n
!!   + call\n
!!   + jmp / jmpl \n
!!   + jle / jl \n
!!   + jg / jge \n
!!   + je / jecxz\ n
!!   + jn / jne / ... \n
!!   + js\n
!!   + jb / jbe \n
!!   + jc / jcxz \n
!!   + ja / jae \n
!!   + jo \n
!!   + jp / jpe \n
!! Anhand der Sprungmarken werden die ermittelten Werte den jeweiligen Funktionen
!! zugeordnet. Die unbedingten Sprünge werden zwar mitgezählt und ausgegeben,
!! flie ß en jedoch nicht in die Berechnung der McCabe-Werte ein, da es sich
!! nicht um Entscheidungen handelt.
!!

program McCabe_Assembler
    implicit none
    integer, parameter :: LINE_LEN = 130
    integer, parameter :: LABEL_LEN = 26
    integer, parameter :: ARRAY_LEN = 2000
    character (len=12), parameter :: FORMAT_SUMMARY = "(7I9,1X,A24)"
    character (len=LINE_LEN) :: filename, filename_trim, &
    & commented_lines, line ="", linecomplete ="␣"
    character (len=2) :: option
    integer :: char1
    integer :: label = 0
    integer :: fehler, i
    integer :: sum_one = 0, sum_two = 0, sum_three = 0, sum_four =0, &
    & sum_five = 0, sum_more_than_five = 0
    integer , dimension(1:ARRAY_LEN) :: call_array = 0, jmp_array = 0, &
    & jl_array = 0, jg_array = 0, mccabe_array = 0, je_array =0, &
    & jn_array = 0, js_array = 0, jb_array = 0, jc_array = 0, &
    & jz_array = 0, ja_array = 0, jo_array = 0, jp_array = 0
    character (len=LABEL_LEN), dimension(1:ARRAY_LEN) :: label_array = ""
    logical :: b = .false., n = .false., s = .false.
    call get_command_argument(1,filename)
```

```
    if (len_trim(filename) < 1 .or. filename .eq. "-h") then
        call print_help_text_and_exit()
    end if
    call get_command_argument(2,option)
    if (option .eq. "-b") then
        b = .true.
    end if
    if (option .eq. "-n") then
        n = .true.
    end if
    if (option .eq. "-s") then
        s = .true.
    end if

    ! Gekürzter Dateiname für Ausgabe
    filename_trim = filename(index(filename,"/",.true.)+1:)
    ! Datei Öffnen
      OPEN (10, FILE = filename, STATUS = 'OLD', IOSTAT = fehler)
!     Fehlermeldung, wenn Datei nicht existiert
      IF (fehler /= 0) then
        write(*,*) 'Datei ', trim(filename), ' existiert nicht!'
        call exit (0)
      end if
    ! Schleife - Zeile lesen
   DO
   read (10, '(A)', end=999) linecomplete
   ! Nur den nicht auskommentierten Bereich analysieren
   if (index(linecomplete,"#") .ge. 1 ) then
        line = linecomplete(1:index(linecomplete,"#"))
   ! Blockkommentare überspringen
   else if (index(linecomplete,"/*") .ge. 1 ) then
        do while (index(linecomplete,"*/") .lt. 1)
            read (10,"(A)") linecomplete
        end do
   else
        line = linecomplete
   end if
   !write (*,*) line
   ! Erstes Zeichen festhalten für Überprüfung:
   ! Wenn Wortzeichen, und Wort beendet mit ":"
   ! dann handelt es sich um eine Sprungmarke (Label)
    char1 = iachar(line(1:1))
   ! Label zählen
   ! write (*,*) char1, "<", line(1:10) ,">"
   if (( &
   & (char1 > 47 ) .and. (char1 < 58) .or. &
   & (char1 > 64) .and. (char1 < 91)  .or. &
   & (char1 .eq. 46 ) .or. &
   & (char1 > 96) .and. (char1 < 123)) .and. &
   & (index(line,":")>0)) then
      ! write (*,*) line(1:index(line," ")),"index ':'", index(line,":"),"<",line(
          index(line," ")-1:index(line," ")),">"
       label = label + 1
```

```
      label_array(label)=line(1:index(line,":"))
      !write (*,*) "label", label
    end if
    ! call zählen
    if (index(line,"call") > 0 ) then
        call_array(label) = call_array(label) +1
    end if
    ! jmp / jmpl, etc. zählen
    if (index(line,"jmp") > 0 ) then
        jmp_array(label) = jmp_array(label) +1
     end if
    ! jl / jle zählen
    if (index(line,"jl") > 0 ) then
        jl_array(label) = jl_array(label) +1
     end if
    ! jg / jge zählen
    if (index(line,"jg") > 0 ) then
        jg_array(label) = jg_array(label) +1
     end if
    ! je / jecxz zählen
    if (index(line,"je") > 0 ) then
        je_array(label) = je_array(label) +1
     end if
    ! jn / jne etc. zählen
    if (index(line,"jn") > 0 ) then
        jn_array(label) = jn_array(label) +1
     end if
    ! js zählen
    if (index(line,"js") > 0 ) then
        js_array(label) = js_array(label) +1
     end if
    ! jb / jbe zählen
    if (index(line,"jb") > 0 ) then
        jb_array(label) = jb_array(label) +1
     end if
    ! jc zählen
    if (index(line,"jc") > 0 ) then
        jc_array(label) = jc_array(label) +1
     end if
    ! jz zählen
    if (index(line,"jz") > 0 ) then
        jz_array(label) = jz_array(label) +1
     end if
    ! ja / jae zählen
    if (index(line,"ja") > 0 ) then
        ja_array(label) = ja_array(label) +1
     end if
    ! jo  zählen
    if (index(line,"jo") > 0 ) then
        jo_array(label) = jo_array(label) +1
     end if
    ! jp /jpe etc. zählen
    if (index(line,"jp") > 0 ) then
```

```
        jp_array(label) = jp_array(label) +1
     end if
     mccabe_array(label) = jl_array(label) + &
     & jg_array(label) + je_array(label) + &
     & jn_array(label) + js_array(label) + &
     & jb_array(label) + jc_array(label) + &
     & jc_array(label) + jz_array(label) + &
     & jo_array(label) + jp_array(label) + &
     & ja_array(label) + 1
   END DO
999 continue
    do i=1,label
        if (mccabe_array(i) .eq. 1) then
            sum_one = sum_one +1
        else  if (mccabe_array(i) .eq. 2) then
            sum_two = sum_two +1
        else if (mccabe_array(i) .eq. 3) then
            sum_three = sum_three +1
        else  if (mccabe_array(i) .eq. 4) then
            sum_four = sum_four +1
        else   if (mccabe_array(i) .eq. 5) then
            sum_five = sum_five +1
        else  if (mccabe_array(i) .gt. 5) then
            sum_more_than_five = sum_more_than_five +1
        else
        end if
    end do

    if (.not. b .and. .not. n .and. .not. s) then
        write (*,*) "Werte␣pro␣Funktion:"
    end if
    if (.not. s) then
        write (*,'(14A9,A15,A8,A12)') "call" , "jmp", "jl+", "jg+", "je+",&
        & "jn+", "js", "jb+", "jc", "jz", "ja+", "jo","jp+","McCabe", &
        & "Funktion", "Datei:",filename_trim
        do i=1,label
            write (*,"(14I9,5X,A)")  call_array(i), jmp_array(i), jl_array(i), &
            jg_array(i), je_array(i), jn_array(i), js_array(i), jb_array(i),
                jc_array(i),&
            & jz_array(i), ja_array(i),jo_array(i),jp_array(i), mccabe_array(i),
                label_array(i)
        end do
    end if
    if (b) then
        write (*,"(14I9,5X)") &
        & sum(call_array(1:label)), sum(jmp_array(1:label)), &
        & sum(jl_array(1:label)),sum(jg_array(1:label)), &
        & sum(je_array(1:label)), sum(jn_array(1:label)), &
        & sum(js_array(1:label)), sum(jb_array(1:label)), &
        & sum(jc_array(1:label)), sum(jz_array(1:label)), &
        & sum(ja_array(1:label)), sum(jo_array(1:label)), &
        & sum(jp_array(1:label)), sum(mccabe_array(1:label)), label
    else if (s) then
```

```
        write (*,2010) &
        & "McCabe-Wert:", "1␣","2","3","4","5",">5", &
        & "Anzahl:",sum_one, sum_two, sum_three, sum_four, sum_five,
            sum_more_than_five
    else if (.not. n) then
        write (*,2000) "Gesamtwerte␣für␣",filename_trim, &
        &  "call:", sum(call_array(1:label)), "jmp:", sum(jmp_array(1:label)), &
        & "jl+:", sum(jl_array(1:label)), "jg+:", sum(jg_array(1:label)), &
        & "jn+:" , sum(jn_array(1:label)), "je+:", sum(je_array(1:label)), &
        & "js:", sum(js_array(1:label)), "jb+:", sum(jb_array(1:label)), &
        & "jc:", sum(jc_array(1:label)), "jz:", sum(jz_array(1:label)), &
        & "ja+:", sum(ja_array(1:label)), "jo:", sum(jo_array(1:label)), &
        & "jp+", sum(jp_array(1:label)), &
        & "McCabe:",sum(mccabe_array(1:label)),"Label:",label
        write (*,*) "Anzahl␣Funktionen␣nach␣McCabe-Werten"
        write (*,2010) &
        & "McCabe-Wert:", "1␣","2","3","4","5",">5", &
        & "Anzahl:",sum_one, sum_two, sum_three, sum_four, sum_five,
            sum_more_than_five
    end if
! Format für Zusammenfassung lang
2000 format(5X,A,A,/15(A8,I4))
! Format für Statistik
2010 format(5X,A12,6A5/5X,A12,6I5)
end program McCabe_Assembler


!> Hilfetext bei nicht angegebener Eingabedatei
subroutine print_help_text_and_exit()
    character * 130 :: progname
    call get_command_argument(0,progname)
    write (*,*) "Benutzung␣:␣",trim(progname),"␣Dateiname␣[-b|-n]"
    write (*,*) "Beschreibung:"
    write (*,*) "Berechnung␣von␣McCabe-Metriken␣für␣Assembler␣(x86␣und␣ähnliche)"
    write (*,*) "Nach␣dem␣Einlesen␣der␣Quelldatei␣werden␣Sprungmarken␣(Labels)␣gezä
        hlt,"
    write (*,*) "und␣der␣Code␣wird␣untersucht␣auf␣das␣Vorhandensein␣von␣
        Sprunganweisungen:"
    write (*,*) "␣+␣call"
    write (*,*) "␣+␣jmp␣/␣jmpl"
    write (*,*) "␣+␣jle␣/␣jl"
    write (*,*) "␣+␣jg␣/␣jge"
    write (*,*) "␣+␣je␣/␣jecxz"
    write (*,*) "␣+␣jn␣/␣jne␣/␣..."
    write (*,*) "␣+␣js"
    write (*,*) "␣+␣jb␣/␣jbe"
    write (*,*) "␣+␣jc"
    write (*,*) "␣+␣ja␣/␣jae"
    write (*,*) "␣+␣jo"
    write (*,*) "␣+␣jp␣/␣jpe"
    write (*,*) "Anhand␣der␣Sprungmarken␣werden␣die␣ermittelten␣Werte␣den␣jeweiligen
        ␣Funktionen"
```

```
    write (*,*) "zugeordnet. Die unbedingten Sprünge werden zwar mitgezählt und
        ausgegeben,"
    write (*,*) "fließen jedoch nicht in die Berechnung der McCabe-Werte ein, da es
        sich "
    write (*,*) "nicht um Entscheidungen handelt."
    write (*,*) 'Option "-b": Keine Überschrift, verkürzte Zusammenfassung (nur
        Summen).'
    write (*,*) 'Option "-n": Keine Überschrift, keine Zusammenfassung.'
    write (*,*) 'Option "-s": Nur Statistik der Funktionen nach McCabe-Werten'
    call exit (0)
end subroutine
```

16.10 Perl-Skript zur Umwandlung des AGC-Codes in analysierbaren C-Code

```
#!/usr/bin/perl
#
#  convert_to_c.pl
#
#  Copyright 2016 Matthias Seidel
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
#  MA 02110-1301, USA.
#

use strict;
use warnings;

my $linecomplete;
my $line;
my $functionindex =0;
my $index;
my $num_args;
my $fh;
$num_args = $#ARGV + 1;
if ($num_args < 1) {
    print "\nBenutzung: $0 Dateiname\n";
    exit;
}
open($fh, "<", $ARGV[0])
    or die "Öffnen von \"$ARGV[0]\" nicht möglich: $!";
```

```
     # include stdio und main-Funktion einfügen
     print "#include␣<stdio.h>\n";
     print "int␣main()␣{\n";
# Hauptschleife zeilenweise einlesen
while ($linecomplete = readline($fh))
{
  $index = index($linecomplete,'#');
   if ($index >-1 )
  {
    $line = substr($linecomplete,0,$index);
  }
  else
  {
    $line = $linecomplete;
  }

  if ($line =~ /^(\w+)\s/ )
  {
    $functionindex +=1;
    if ($functionindex > 1)
    {
      $line =~ s/^(\w+)\s/return 0;}\nint $1() { /;
    }
    else
    {
    $line =~ s/^(\w+)\s/int $1() { /;
    }
  }
  # Sonderzeichen am Bezeichneranfang in Text umwandeln
  $line =~ s/(\s+|^)\/([A-Z0-9])/$1SLASH$2/;


  # Ziffern am Bezeichneranfang
  # ZERO, ONE, etc können nicht immer verwendet werden,
  # daher Bezeichnungen in Kleinbuchstaben zur Unterscheidung
  $line =~ s/(\s|^)0(\w+)/$1zero$2/;
  $line =~ s/(\s|^)1(\w+)/$1one$2/;
  $line =~ s/(\s|^)2(\w+)/$1two$2/;
  $line =~ s/(\s|^)3(\w+)/$1three$2/;
  $line =~ s/(\s|^)4(\w+)/$1four$2/;
  $line =~ s/(\s|^)5(\w+)/$1five$2/;
  $line =~ s/(\s|^)6(\w+)/$1six$2/;
  $line =~ s/(\s|^)7(\w+)/$1seven$2/;
  $line =~ s/(\s|^)8(\w+)/$1eight$2/;
  $line =~ s/(\s|^)9(\w+)/$1nine$2/;

  # TC zu Funktionsaufruf
  if ($line =~ /\sTC\s/ )
  {
    $line =~ s/\sTC\s(\w+)/$1(); \/\/ TC /;
  }
  # TCF zu Funktionsaufruf
```

```
  elsif ($line =~ /\sTCF\s/ )
  {
    $line =~ s/\sTCF\s([A-Z0-9,\+-]+)/$1(); \/\/ TCF /;
  }
  # BZMF zu Funktionsaufruf
  elsif ($line =~ /\sBZMF\s/ )
  {
    $line =~ s/\sBZMF\s(\w+)/$1(); \/\/ BZMF /;
  }
  else
        {
                # standardfuntionen mittel printf ausgeben
                substr($line,0,$index)=~ s/^(\s+)(.*)/$1 printf(\"%s\\n\",\"$2\")\;␣
                    /;
␣␣␣␣␣␣␣␣}


␣␣$line␣=~␣s/(\s+|^)\+([0-9])/$1plus$2/;

␣␣␣␣#␣Bisher␣nicht␣behandelte␣Operationen␣auskommentieren
␣␣#␣Operationen␣nach␣"{"␣auskommentieren
␣␣$line␣=~␣s/({)(.*)/$1␣\/\/$2/;
␣␣#␣Übrige␣Label␣auskommentieren
␣␣$line␣=~␣s/^([A-Z0-9+,-]+)/\/\/\/$1/;

#␣Ausgabe,␣aufgeteilt␣in␣Code-␣und␣Kommentarbereich,␣sofern␣Kommentare␣vorhanden
␣␣if␣($index␣>-1␣)
␣␣{
␣␣␣␣print␣␣$line␣."//".␣␣substr($linecomplete,$index+1);
␣␣}
␣␣else
␣␣{
␣␣␣␣print␣$line;
␣␣}
}
#␣letztes␣return␣und␣schlie ß ende␣Klammer
print␣"\nreturn 0; } /* end */";
```

16.11 Perl-Skript zur Rückübersetzung

```
#!/usr/bin/perl
#
#  convert_back.pl
#
#  Copyright 2016 Matthias Seidel
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
```

```
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
#  MA 02110-1301, USA.
#

use strict;
use warnings;

my $fh;
my $line;
my $num_args = $#ARGV + 1;
my $index;
if ($num_args < 1) {
    print "\nBenutzung: $0 Dateiname\n";
    exit;
}
# Datei öffnen
open($fh, "<", $ARGV[0])
    or die "Öffnen von \"$ARGV[0]\" nicht möglich: $!";
# Zeilenweise lesen
while ($line =readline($fh))
{

  # in Kommentaren nach TC, TCF, BZMF suchen und Funktionsaufrufe zurückübersetzen
  if ($line =~ /\/\/ TC\s/ )
  {
    $line =~ s/(\w+)\(\); \/\/ TC/TC $1/;
  }
  if ($line =~ /\/\/ TCF\s/ )
  {
    $line =~ s/([\w,\+-]+)\(\); \/\/ TCF/TCF $1/;
  }
   if ($line =~ /\/\/ BZMF\s/ )
  {
    $line =~ s/(\w+)\(\); \/\/ BZMF/BZMF $1/;
  }
  # eingefügte return 0, include- und main-Zeilen aus Ausgabe entfernen
  if ($line =~ /^return|include|main/ )
  {
    next;
  }
  # Funktionsaufrufe entfernen
  if ($line =~ /^int/ )
  {
    $line =~ s/int //;
    $line =~ s/\(\) {//;
  }
  # Kommentare zurückübersetzen
```

```
  $line =~ s#///\+\+##;
  $line =~ s{//}{#};
  $line =~ s/^(\w+\s)#(.*)/$1$2/;
  if ($line !~ /http:/) # Sondefall für Hyperlink-Angabe in veröffentlichtem AGC-
      Code, nicht im Original vorhanden
  {
  $line =~ s/\/\//#/;
  }
  # Zeichenkombination #/
  $line =~ s/#\///;

  $index = index $line,'#';

  # Sonderzeichen zurückübersetzen
  $line =~ s/(\s|^)SLASH(\w+)/$1\/$2/;


  if ( substr($line,0,$index) =~ m/printf/)
        {
                $line =~ s/printf\(\"\%s\\n\",\"(.*)\"\)\;/$1/;
␣␣␣␣␣␣␣␣}


␣␣#␣Ziffern␣zurückübersetzen
␣␣$line␣=~␣s/zero/0/;
␣␣$line␣=~␣s/one/1/;
␣␣$line␣=~␣s/two/2/;
␣␣$line␣=~␣s/three/3/;
␣␣$line␣=~␣s/four/4/;
␣␣$line␣=~␣s/five/5/;
␣␣$line␣=~␣s/six/6/;
␣␣$line␣=~␣s/seven/7/;
␣␣$line␣=~␣s/eight/8/;
␣␣$line␣=~␣s/nine/9/;

␣␣#␣+␣zurückübersetzen
␣␣$line␣=~␣s/plus(\d)/+$1/;

␣␣#␣Ausgabe
␣␣␣␣unless␣($line␣=~␣/end/)
␣␣␣␣{
␣␣␣␣print␣$line;
␣␣␣␣}
}
```

16.12 Sed-Skript zur Ermittlung von Interpreter-Befehlen/Argumenten

```
#! /bin/bash
interpreter=(ABS ABVAL ARCCOS ACOS ARCSIN ASIN AXC,1 AXC,2 AXT,1 AXT,2 BDDV BDDV*
    BDSU BDSU* BHIZ BMN BOFCLR BOF BOFF BOFINV BOFSET BON BONCLR BONINV BONSET BOV
    BOVB BPL BVSU BVSU* BZE CALL CALRB CCALL CCALL* CGOTO CGOTO* CLEAR CLR CLRGO COS
     COSINE DAD DAD* DCOMP DDV DDV* DLOAD DLOAD* DMP DMP* DMPR DMPR* DOT DOT* DSQ
```

```
    DSU DSU* EXIT GOTO INCR,1 INCR,2 INVERT INVGO ITA LXA,1 LXA,2 LXC,1 LXC,2 MXV
    MXV* NORM NORM* PDDL PDDL* PDVL PDVL* PUSH ROUND RTB RVQ SET SETGO SETPD SIGN
    SIGN* SIN SINE SL SL* SLOAD SLOAD* SL1 SL1R SL2 SL2R SL3 SL3R SL4 SL4R SLR SLR*
    SQRT SR SR* SR1 SR1R SR2 SR2R SR3 SR3R SR4 SR4R SRR SRR* SSP SSP* STADR STQ SXA
    ,1 SXA,2 TAD TAD* TIX,1 TIX,2 TLOAD TLOAD* UNIT V/SC V/SC* VAD VAD* VCOMP VDEF
    VLOAD VLOAD* VPROJ VPROJ* VSL VSL* VSL1 VSL2 VSL3 VSL4 VSL5 VSL6 VSL7 VSL8 VSQ
    VSR VSR* VSR1 VSR2 VSR3 VSR4 VSR5 VSR6 VSR7 VSR8 VSU VSU* VXM VXM* VXSC VXSC*
    VXV VXV* XAD,1 XAD,2 XCHX,1 XCHX,2 XSU,1 XSU,2)
for  s in  ${interpreter[*]}; do
#echo $s
sed -r -n 's/.*(\<'"$s"'\>|$s\*)[[:cntrl:]](.*)$/Command:\t\1\tArgument:\t\2/p' $1
done
```

17 Abbildungsverzeichnis

Literatur

[1] AC Electronics Division of General Motors.
APOLLO GUIDANCE AND NAVIGATION SYSTEM LUNAR MODULE STUDENT STUDY GUIDE.
MILWAUKEE, WISCONSIN, 1967.

[2] Airbus. *A320 introduces new features | Airbus Press release*.
URL: http://www.airbus.com/newsevents/news-events-single/detail/a320-introduces-new-features/ (besucht am 16. 03. 2016).

[3] Airbus. *Fly-by-wire | Airbus, a leading aircraft manufacturer*.
URL: http://www.airbus.com/innovation/proven-concepts/in-design/fly-by-wire/?contentId=[_TABLE:tt_content;_FIELD:uid],&cHash=22935adfac92fcbbd4ba4e1441d13383 (besucht am 16. 03. 2016).

[4] Aivosto Oy. *Project Metrics Help - Complexity metrics*.
URL: http://www.aivosto.com/project/help/pm-complexity.html (besucht am 23. 02. 2016).

[5] Charles Draper et Al. *Space Navigation Guidance and Control*. Juni 1965.

[6] Lydia Parziale et Al. *Introduction to the New Mainframe Z/Vm Basics*.
IBM, International Technical Support Organization, 2007. ISBN: 9780738488554.

[7] Alan I. Greene and Robert J. Filene. „Keyboard and Display Program and Operation“. In: (1967-06-01).

[8] Jason Allen, Scott Collison und Robin Luckey. *Ohloh - About*.
URL: https://web.archive.org/web/20070102180201/http://www.ohloh.net/wiki/about/us (besucht am 03. 01. 2016).

[9] Paul Bame. *PMCCABE*.
URL: https://people.debian.org/~bame/pmccabe/ (besucht am 01. 01. 2016).

[10] Keith L Bedingfield, Richard D Leach und Margaret B Alexander.
Spacecraft system failures and anomalies attributed to the natural space environment.
National Aeronautics und Space Administration, Marshall Space Flight Center, 1996. URL: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19960050463.pdf.

[11] Roger E. Bilstein. *Stages to Saturn: A Technological History of the Apollo/Saturn Launch Vehicle*.
National Aeronautics und Space Administration, NASA History Office, 1980.

[12] Black Duck Software. *Open Hub, the open source network*.
URL: https://www.openhub.net/ (besucht am 03. 01. 2016).

[13] Hugh Blair-Smith. *AGC - Annotations to Eldon Hall's Journey to the Moon.* 1997.
URL: http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/blairsmith.htm (besucht am 30. 03. 2016).

[14] Columbia Accident Investigation Board. *COLUMBIA ACCIDENT INVESTIGATION BOARD*.
August 2003. URL: http://spaceflight.nasa.gov/shuttle/archives/sts-107/investigation/CAIB_medres_full.pdf (besucht am 06. 03. 2016).

[15] Boeing. *Innovative 787 Flight Deck Designed for Efficiency, Comfort, and Commonality*.
URL: http://www.boeing.com/commercial/aeromagazine/articles/2012_q1/3/ (besucht am 16. 03. 2016).

[16] Boeing. *The Boeing 777 Family Backgrounder*.
URL: http://www.boeing.com/paris2013/pdf/BCA/bck-777%20Family%20Backgrounder.pdf (besucht am 16. 03. 2016).

[17] Dominik Borowiec. *dborowiec/commentedCodeDetector · GitHub*. URL: https://github.com/dborowiec/commentedCodeDetector (besucht am 03. 01. 2016).

[18] Robert A. Braeunig. *Apollo and the Van Allen Belts*.
URL: http://www.braeunig.us/apollo/VABraddose.htm (besucht am 23. 03. 2016).

[19] Courtney G Brooks, James M Grimwood und Loyd S Swenson Jr.
Chariots for Apollo: A History of Manned Lunar Spacecraft.
National Aeronautics and Space Administration, 1979.
URL: http://ntrs.nasa.gov/search.jsp?R=19790020032.

[20] TomTom International B.V. *TomTom GO 61*.
URL: http://www.tomtom.com/de_de/drive/car/products/go-61/ (besucht am 15. 01. 2016).

[21] Paul E. Ceruzzi. *Eine kleine Geschichte der EDV*. 1. Aufl. Bonn: mitp-Verl., 2003, 411 S.
ISBN: 3-8266-0759-7.

[22] Noam Chomsky. „Three models for the description of language“. Englisch.
In: *IRE Transactions on Information Theory* 2 (1956), S. 113–124.
URL: http://www.chomsky.info/articles/195609--.pdf.

[23] *CodeAnalyzer - Multi-Platform Java Code Analyzer*.
URL: http://www.codeanalyzer.teel.ws/ (besucht am 11. 03. 2016).

[24] Commonwealth Scientific and Industrial Research Organisation. *Apollo 11 Moon landing - CSIRO*.
URL: http://www.csiro.au/en/Research/Astronomy/Spacecraft-tracking-and-space-science/Apollo-11-Moon-landing (besucht am 31. 01. 2016).

[25] Commonwealth Scientific and Industrial Research Organisation.
CSIRO Parkes Observatory | Australia Telescope National Facility.
URL: http://www.parkes.atnf.csiro.au/ (besucht am 31. 01. 2016).

[26] Edward M. Copps Jr. *Recovery From Transient Failures of the Apollo Guidance Computer*. August 1968.

[27] Fernando J. Corbató. *E-Mail-Antwort an den Autor*. 12. Jan. 2016.

[28] Fernando J. Corbató, Marjorie Merwin-Daggett und Robert C. Daley.
„An Experimental Time-sharing System“.
In: *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*. AIEE-IRE '62 (Spring).
San Francisco, California: ACM, 1962, S. 335–344. DOI: 10.1145/1460833.1460871.
URL: http://doi.acm.org/10.1145/1460833.1460871.

[29] John Pina Craven. *The Silent War: The Cold War Battle Beneath the Sea*. Simon und Schuster, 2002.

[30] Frank da Cruz. *Herman Hollerith Tabulating Machine*.
URL: http://www.columbia.edu/cu/computinghistory/hollerith.html (besucht am 18. 03. 2016).

[31] Deutsche Gesellschaft für Qualität e.V. Frankfurt am Main, Nachrichtentechnische Gesellschaft im VDE (NTG) (Hg.) *Software-Qualitätssicherung. Aufgaben, Möglichkeiten, Lösungen*.
Berlin: Beuth Verlag, 1986. ISBN: 3-8007-1462-0.

[32] The Dibner Institute for the History of Science and Technology. *AGC - ICs in the AGC*.
URL: http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/ic.htm (besucht am 17. 03. 2016).

[33] The Dibner Institute for the History of Science and Technology. *AGC - Visual Introduction to the AGC*.
URL: http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/visualintro.htm (besucht am 08. 04. 2016).

[34] Larry Dignan. *IBM Watson computer wins US quiz show round | ZDNet*.
URL: http://www.zdnet.com/article/ibm-watson-computer-wins-us-quiz-show-round/#! (besucht am 16. 01. 2016).

[35] Discovery Channel. *Moon Machines, part 3: The Navigation Computer*. 2008.

[36] Douglas Engelbart Institute. *Curriculum Vitae - Doug Engelbart Institute*.
URL: http://dougengelbart.org/about/cv.html (besucht am 27. 02. 2016).

[37] Douglas Engelbart Institute. *Mouse - Doug Engelbart Institute*.
URL: http://www.dougengelbart.org/firsts/mouse.html (besucht am 19. 03. 2016).

[38] Charles Draper. *Letter from Charles Stark Draper to Charles W. Frick*.
abgedruckt in Journey to the Moon. 16. Nov. 1962.
[39] Charles Stark Draper. *Letter from Charles Stark Draper to Robert Seamans*. 21. November 1961.
[40] Hugh L. Dryden. *FUTURE EXPLORATION AND UTILIZATION OF OUTER SPACE*. 9. Sep. 1961.
[41] El Economista. *Envia por email : Un alumno de la UMH realiza el software de una antena de estación de la NASA - 628474 - 30/06/08 - EcoDiario.es - EcoDiario.es*.
URL: http://ecodiario.eleconomista.es/noticias-email/628474/Un-alumno-de-la-UMH-realiza-el-software-de-una-antena-de-estacion-de-la-NASA (besucht am 26. 02. 2016).
[42] Dale C Ferguson, William F Denig und Juan V Rodriguez.
„Plasma conditions during the Galaxy 15 anomaly and the possibility of ESD from subsurface charging“.
In: *Proceedings of the 49th AIAA Aerospace Sciences meeting in Orlando, Florida*. 2011, S. 4–7.
URL: http://arc.aiaa.org/doi/pdf/10.2514/6.2011-1061;%20http://www.enu.kz/repository/2011/AIAA-2011-1061.pdf.
[43] Jack Garman. *Console Audio of Apollo 11 Landing*.
URL: http://klabs.org/history/apollo_11_alarms/console/index.htm (besucht am 19. 03. 2016).
[44] R.L. Garthoff. *Reflections on the Cuban Missile Crisis: Revised to include New Revelations from Soviet & Cuban Sources*. Brookings Institution Press, 1989. ISBN: 9780815717393.
[45] Sharon Gaudin. *The Orion spacecraft is no smarter than your phone | Computerworld*.
URL: http://www.computerworld.com/article/2855604/the-orion-spacecraft-is-no-smarter-than-your-phone.html (besucht am 24. 04. 2016).
[46] The Global Aircraft Organization. *Global Aircraft – Concorde*.
URL: http://www.globalaircraft.org/planes/concorde.pl (besucht am 16. 03. 2016).
[47] R. D. Gordon und M. H. Halstead.
„An Experiment Comparing Fortran Programming Times with the Software Physics Hypothesis“.
In: *Proceedings of the June 7-10, 1976, National Computer Conference and Exposition*. AFIPS '76.
New York, New York, 1976, S. 935–937.
[48] Grumman. *LEM CES and AGS*. Sep. 1966.
[49] Barton C. Hacker und James M. Grimwood. *On the Shoulders of Titans: A History of Project Gemini*.
1977. URL: http://ntrs.nasa.gov/search.jsp?R=19780012208.
[50] Eldon C. Hall. *APOLLO GUIDANCE COMPUTER (AGC) Schematics*. 1965.
URL: http://www.klabs.org/history/ech/agc_schematics/index.htm (besucht am 04. 04. 2016).
[51] Eldon C. Hall. „CASE HISTORS OF THE APOLLO GUIDANCE COMPUTER“. In: (1966).
URL: http://klabs.org/history/history_docs/mit_docs/1675.pdf.
[52] Eldon C. Hall. „From the Farm to Pioneering with Digital Control Computers: An Autobiography“.
In: *IEEE Annals of the History of Computing* (2000).
[53] Eldon C. Hall. *Journey to the Moon: The History of the Apollo Guidance Computer*.
AIAA (American Institute of Aeronautics & Ast, 1996. ISBN: 9781563471858.
[54] Eldon C. Hall. *MITI's ROLE IN PROJECT APOLLO*. MIT, 1972.
[55] Eldon C. Hall.
Problems with IBM Approach as it is Defined in the IBM Apollo Study Report No. 63-928-129. 1963.
[56] Eldon C. Hall. *Reliability History of the Apollo Guidance Computer*. January 1972.
[57] Eldon C. Hall. *The Apollo Guidance Computer, part one*. 10 June 1982.
URL: http://www.computerhistory.org/collections/catalog/102624617 (besucht am 16. 01. 2016).

[58] Margaret H. Hamilton.
„Guidance system operations plan for manned CM earth orbital missions using program Skylark“.
In: (1972). URL: http://ntrs.nasa.gov/search.jsp?R=19720017954.

[59] Margaret H. Hamilton. *Hamilton Technologies*.
URL: http://www.htius.com/ (besucht am 16. 01. 2016).

[60] Margaret H. Hamilton und William R Hackler.
„Universal Systems Language for Preventative Systems Engineering“.
In: *Proc. 5th Ann. Conf. Systems Eng. Res*. 2007.
URL: http://www.htius.com/Articles/36.pdf.

[61] Margaret H. Hamilton und William R. Hackler.
„Universal Systems Language: Lessons Learned from Apollo.“
In: *IEEE Computer* 41.12 (2008), S. 34–43. URL: http://dblp.uni-trier.de/db/journals/computer/computer41.html#HamiltonH08.

[62] Ralf-Udo Hartmann. *drhart - Core Rope Memory*. URL:
http://drhart.ucoz.com/index/core_memory/0-123-0-123 (besucht am 26. 02. 2016).

[63] R. H. Hoffman. *Automated verification system user's guide*. Techn. Ber. 12. Januar 1972.

[64] IBM. *IBM - Archives - History of IBM - 1960s*.
URL: http://www-03.ibm.com/ibm/history/history/decade_1960.html (besucht am 30. 04. 2016).

[65] IBM. *IBM Archives: Space flight chronology page 2*. URL: http://www-03.ibm.com/ibm/history/exhibits/space/space_chronology2.html (besucht am 24. 02. 2016).

[66] IBM. *IBM NEWS EXTRA*. 7. Apr. 1964.

[67] IBM. *IBM PowerPC 750FX RISC Microprocessor*. 2003.

[68] IBM. *IBM100 - System 360*.
URL: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/system360/ (besucht am 19. 03. 2016).

[69] IBM. *IBM100 - The Making of International Business Machines*.
URL: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/makingibm/ (besucht am 18. 03. 2016).

[70] IBM. *Watson announcing the IBM System/360 - CHM Revolution*.
Computer History Museum, Object ID 500004184. URL:
http://computerhistory.org/revolution/mainframe-computers/7/161/576 (besucht am 01. 01. 2016).

[71] Indiana University. *What is the difference between 16-bit and 32-bit Windows applications?* 2010.
URL: https://kb.iu.edu/d/ahop (besucht am 26. 02. 2016).

[72] James E. Tomayko. *Computers take flight: a history of NASA's pioneering digital fly-by-wire project*.
Washington, D.C. : National Aeronautics, Space Administration, NASA Office of Policy und Plans, NASA History Office : 2000. ISBN: 9780160590535.

[73] Richard A. Gustafson, Jeffrey N. Wilkes.
APOLLO EXPERIENCE REPORT - A USE OF NETWORK SIMULATION TECHNIQUES IN THE DESIGN OF THE APOLLO LUNAR SURFACE EXPERIMENTS PACKAGE SUPPORT SYSTEM.
September 1974.

[74] Dennis Jenkins. *Advanced Vehicle Automation and Computers Aboard the Shuttle*.
URL: http://history.nasa.gov/sts1/pages/computer.html (besucht am 26. 02. 2016).

[75] J.S. Kilby. „Invention of the integrated circuit“.
In: *Electron Devices, IEEE Transactions on* 23.7 (1976), S. 648–654. ISSN: 0018-9383.
DOI: 10.1109/T-ED.1976.18467.
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1478481.

[76] Andrey Konstantinov. *Metrix++ | SourceForge.net*.
URL: http://metrixplusplus.sourceforge.net/ (besucht am 01. 01. 2016).
[77] Marcus Lindroos. *THE SOVIET MANNED LUNAR PROGRAM EDITED AND COMPILED BY*.
[78] *Listing des Colossus-Codes | NASA*. April 1969.
[79] Glynn S. Lunney. *"Before This Decade Is Out...", Chapter 9*.
URL: http://history.nasa.gov/SP-4223/ch9.htm (besucht am 01. 03. 2016).
[80] R.E. Lyons und W. Vanderkulk.
„The Use of Triple-Modular Redundancy to Improve Computer Reliability“.
In: *IBM Journal of Research and Development* 6.2 (1962), S. 200–209. ISSN: 0018-8646.
DOI: 10.1147/rd.62.0200.
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5392355.
[81] Colin Mackellar. *A Tribute to Honeysuckle Creek Tracking Station*.
URL: http://www.honeysucklecreek.net/ (besucht am 26. 02. 2016).
[82] Manfred Baumgartner, Richard Seidl, Harry M. Sneed, *Software in Zahlen*.
Hanser Fachbuchverlag, 2010. ISBN: 9783446421752.
[83] Nathaniel Manista. *Apollo 11 mission's 40th Anniversary: One large step for open source code... - The official Google Code blog*. URL: http://googlecode.blogspot.de/2009/07/apollo-11-missions-40th-anniversary-one.html (besucht am 10. 03. 2016).
[84] Thomas J. McCabe. „A Complexity Measure“.
In: *Proceedings of the 2nd International Conference on Software Engineering*. ICSE '76.
San Francisco, California, USA: IEEE Computer Society Press, 1976, S. 407–.
[85] Microsoft. *Multitasking von 16-Bit-/32-Bit-Anwendungen in Windows 95*.
URL: https://support.microsoft.com/de-de/kb/117567 (besucht am 25. 03. 2016).
[86] MIT und WGBH Boston. *A Solution to Computer Bottlenecks*. Mai 1963.
URL: http://techtv.mit.edu/videos/10166-a-solution-to-computer-bottlenecks-1963-science-reporter-tv-series (besucht am 18. 01. 2016).
[87] MIT und WGBH Boston. *Computer For Apollo (1965)*. 1965.
URL: http://teachingexcellence.mit.edu/from-the-vault/computer-for-apollo-1965-science-reporter-tv-series (besucht am 28. 01. 2016).
[88] NASA. *0100806 - Saturn IB/V Instrument Unit (IU) Configuration*.
URL: https://mix.msfc.nasa.gov/abstracts.php?p=1027 (besucht am 19. 03. 2016).
[89] NASA. *2005 MAPLD International Conference: Jack Garman Biography*.
URL: http://klabs.org/mapld05/invited/garman_bio.htm (besucht am 19. 03. 2016).
[90] NASA. *8909250 - Apollo 11 Launched Via Saturn V Rocket*.
URL: https://mix.msfc.nasa.gov/abstracts.php?p=4051 (besucht am 19. 03. 2016).
[91] NASA. *A MEETING WITH THE UNIVERSE*.
URL: http://history.nasa.gov/EP-177/ch5-2.html (besucht am 09. 04. 2016).
[92] NASA. *a12StarList.jpg (JPEG Image, 620 x 793 pixels)*.
URL: http://www.hq.nasa.gov/alsj/a12/a12StarList.jpg (besucht am 16. 01. 2016).
[93] NASA. *Alan Shepard Completes His Mission | NASA*. URL:
http://www.nasa.gov/multimedia/imagegallery/image_feature_1344.html
(besucht am 02. 03. 2016).
[94] NASA. *APOLLO 10 FINAL FLIGHT MISSION RULES*. 15. Apr. 1969.
[95] NASA. *Apollo 11 Flight Journal - Day 1: Transposition, Docking and Extraction*.
URL: http://history.nasa.gov/ap11fj/03tde.htm (besucht am 23. 03. 2016).
[96] NASA. *Apollo 11 Flight Journal - Launch*. URL:
http://history.nasa.gov/ap11fj/01launch.htm#f1start (besucht am 22. 03. 2016).
[97] NASA. *Apollo 11 Flight Journal - Navigation and Housekeeping*. URL:
http://history.nasa.gov/ap11fj/04nav-housekeep.htm (besucht am 23. 03. 2016).

[98] NASA. *Apollo 13 Flight Journal*.
URL: http://history.nasa.gov/ap13fj/index.htm (besucht am 13. 04. 2016).
[99] NASA. *Apollo Expeditions to the Moon: Chapter 3*.
URL: http://history.nasa.gov/SP-350/ch-3-2.html (besucht am 19. 03. 2016).
[100] NASA. *Apollo Flight Journal - Index Page*.
URL: http://history.nasa.gov/afj/index.htm (besucht am 23. 03. 2016).
[101] NASA. *Apollo Flight Journal - The Apollo On-board Computers*.
URL: http://history.nasa.gov/afj/compessay.htm (besucht am 26. 02. 2016).
[102] NASA. *Astronaut Alan B. Shepard, Jr., May 5, 1961 | NASA*. URL:
http://www.nasa.gov/multimedia/imagegallery/image_feature_1076.html
(besucht am 03. 06. 2016).
[103] NASA. *Feb. 26, 1966 Launch of Apollo-Saturn 201 | NASA*.
URL: http://www.nasa.gov/content/apollo-saturn-201-launch/ (besucht am
19. 03. 2016).
[104] NASA. *File:Agc rope.jpg - Wikimedia Commons*. URL:
https://commons.wikimedia.org/wiki/File:Agc_rope.jpg (besucht am 19. 03. 2016).
[105] NASA. *Fly Me to the Moon And Back - JSC315*. 1966.
URL: https://archive.org/details/JSC315_Fly_Me_To_The_Moon_And_Back.wmv
(besucht am 22. 03. 2016).
[106] NASA. *Ground Ignition Weights*. URL: http://history.nasa.gov/SP-4029/Apollo_18-
19_Ground_Ignition_Weights.htm (besucht am 19. 03. 2016).
[107] NASA. *Launch Windows Essay*.
URL: http://www.history.nasa.gov/afj/launchwindow/lw1.html (besucht am
10. 01. 2016).
[108] NASA. *Launch Windows Essay*. URL:
http://history.nasa.gov/afj/launchwindow/lw1.html (besucht am 12. 03. 2016).
[109] NASA. *Mission Control Center Familiarization Manual*. 30. Juni 1967.
[110] NASA. *Nachbildung von Sputnik 1*.
URL: http://nssdc.gsfc.nasa.gov/nmc/masterCatalog.do?sc=1957-001B (besucht
am 01. 02. 2016).
[111] NASA. *NASA - Birth of the Mouse*.
URL: http://www.nasa.gov/vision/earth/technologies/taylor_more_prt.htm
(besucht am 27. 02. 2016).
[112] NASA. *NASA - Rocket Park - Saturn V*.
URL: http://www.nasa.gov/centers/johnson/rocketpark/saturn_v.html (besucht
am 19. 03. 2016).
[113] NASA. *NASA - Saturn V*.
URL: http://www.nasa.gov/centers/johnson/rocketpark/saturn_v.html (besucht
am 19. 03. 2016).
[114] NASA. *NASA - What's in a Name?* URL: http:
//www.nasa.gov/centers/glenn/about/history/silverstein_feature.html
(besucht am 08. 03. 2016).
[115] NASA. *NASA - Yuri Gagarin: First Man in Space*. URL: http:
//www.nasa.gov/mission_pages/shuttle/sts1/gagarin_anniversary.html
(besucht am 02. 03. 2016).
[116] NASA. *Orion Image Gallery | NASA*.
URL: http://www.nasa.gov/exploration/systems/orion/gallery/index.html
(besucht am 26. 03. 2016).

[117] NASA. *Pointing the Way | NASA*.
URL: http://www.nasa.gov/content/topics/history/pointing-the-way (besucht am 08. 05. 2016).
[118] NASA. *RECENT ADVANCES IN DISPLAY MEDIA*. 1968.
[119] NASA. *Saturn Illustrated Chronology - Part 4*.
URL: http://history.nasa.gov/MHR-5/part-4.htm (besucht am 19. 03. 2016).
[120] NASA. *Solar System Exploration: Missions: By Target: Earth: Past: Explorer 1*.
URL: http://solarsystem.nasa.gov/missions/explorer_01/indepth (besucht am 29. 03. 2016).
[121] NASA. *The Apollo 8 Flight Journal - Day 1: The Green Team and Separation*. URL:
http://history.nasa.gov/ap08fj/03day1_green_sep.htm (besucht am 12. 03. 2016).
[122] NASA. *The Centaur Upper Stage Rocket | NASA*.
URL: http://www.nasa.gov/centers/glenn/about/history/centaur.html (besucht am 30. 01. 2016).
[123] NASA.
The Decision to Go to the Moon: President John F. Kennedy's May 25, 1961 Speech before Congress.
URL: http://history.nasa.gov/moondec.html (besucht am 02. 03. 2016).
[124] NASA. *THE MANNED SPACE FLIGHT NETWORK FOR APOLLO*. August 1968.
[125] NASA. *Trajectory of Alan Shepard's Historic Flight | NASA*. URL:
http://www.nasa.gov/multimedia/imagegallery/image_feature_1939.html (besucht am 24. 03. 2016).
[126] NASA. *Van Allen Probes | NASA*.
URL: http://www.nasa.gov/mission_pages/rbsp/main/index.html (besucht am 14. 03. 2016).
[127] NASA.
Van Allen Probes: NASA renames radiation belt mission to honor pioneering scientist – ScienceDaily.
URL: http://www.sciencedaily.com/releases/2012/11/121111101748.htm (besucht am 14. 03. 2016).
[128] NASA. *Van Allen Probes Reveal Zebra Stripes in Space | NASA*.
URL: http://www.nasa.gov/content/goddard/van-allen-probes-reveal-zebra-stripes-in-space (besucht am 29. 03. 2016).
[129] NASA. *Van Allen Probes Revolutionize View of Radiation Belts | NASA*.
URL: http://www.nasa.gov/feature/goddard/2016/nasa-s-van-allen-probes-revolutionize-view-of-radiation-belts (besucht am 28. 01. 2016).
[130] NASA. *Video der Rede von John F. Kennedy im Rice Stadium*. Houston, Texas, 12. Sep. 1962.
URL: http://er.jsc.nasa.gov/seh/ricetalk.htm (besucht am 01. 02. 2016).
[131] NASA. *WHAT MADE APOLLO A SUCCESS? sp287*. NASA special publication 287.
National Aeronautics and Space Administration, 1971.
URL: http://history.nasa.gov/SP-287/sp287.htm (besucht am 06. 03. 2016).
[132] *NASA - Matthew Lemke, Orion Avionics, Power and Wiring Manager*.
URL: http://www.nasa.gov/centers/johnson/home/lemke_profile.html (besucht am 24. 04. 2016).
[133] National Academy of Engineering. *NAE Website - Dr. J. Halcombe Laning*.
URL: http://www.nae.edu/29034.aspx (besucht am 26. 02. 2016).
[134] The New Mexico Museum of Space History. *International Space Hall of Fame ::
New Mexico Museum of Space History :: Inductee Profile*.
URL: http://www.nmspacemuseum.org/halloffame/detail.php?id=6 (besucht am 26. 02. 2016).

[135] NORAD/CONAD. *Historical Summary January – June 1962*. URL: http://www.northcom.mil/Portals/28/Documents/Supporting%20documents/(U)%201962%20NORAD%20CONAD%20History%20Jan-Jun.pdf (besucht am 10. 03. 2016).
[136] Gerard O'Regan. *A brief history of computing*. 2. ed. London: Springer, 2012. ISBN: 978-1-4471-2358-3.
[137] Richard Orloff und Stephen Garber. *Apollo by the numbers: a statistical reference*. 2000. URL: http://ntrs.nasa.gov/search.jsp?R=20010008244.
[138] Erdinc Ozturk. *Enabling HighPerformance Galois-CounterMode on Intel Architecture Processors*. 2012.
[139] Jason Perlow. *To the Moon: The Integrators | ZDNet*. URL: http://www.zdnet.com/article/to-the-moon-the-integrators/ (besucht am 26. 02. 2016).
[140] David SF Portree und Robert C Treviño. „An EVA Chronology". In: *Monographs in Aerospace History Series* (1997). URL: http://54.221.221.71/spacenews/factsheets/pdfs/EVACron.pdf.
[141] Jesco von Puttkamer. *Abenteuer Apollo 11: von der Mondlandung zur Erkundung des Mars*. München: Herbig, 2009, 284 S. ISBN: 978-3-7766-2616-2.
[142] Harish Ramasubramanian. *Halstead Metrics Tool - Browse Files at SourceForge.net*. URL: http://sourceforge.net/projects/halsteadmetricstool/files/ (besucht am 01. 01. 2016).
[143] Raytheon Company. *Raytheon: Raytheon Company: History*. URL: http://www.raytheon.com/ourcompany/history/ (besucht am 19. 03. 2016).
[144] *rburkey2005/virtualagc · GitHub*. URL: https://github.com/rburkey2005/virtualagc (besucht am 10. 03. 2016).
[145] Arnold G. Reinhold. *File:IBM SLT wafers.agr.JPG - Wikimedia Commons*. 28. Jan. 2012. URL: https://commons.wikimedia.org/wiki/File:IBM_SLT_wafers.agr.JPG (besucht am 01. 01. 2016).
[146] Lawrence F. Dietlein, MD Richard S. Johnston und Charles A. Berry, MD. *Biomedical Results of Apollo*. Washington, D.C.: National Aeronautics und Space Administration, 1975.
[147] Dennis M. Ritchie. *Early Unix history and evolution*. URL: https://www.bell-labs.com/usr/dmr/www/hist.html (besucht am 27. 02. 2016).
[148] David Scott. *The Apollo Guidance Computer, part two*. 10 June 1982. URL: http://www.computerhistory.org/collections/catalog/102651598 (besucht am 16. 01. 2016).
[149] A. Sears und J.A. Jacko. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, Second Edition*. Human Factors and Ergonomics. CRC Press, 2007. ISBN: 9781410615862.
[150] Martin Shepperd. *Software Engineering Metrics Volume I - Measures and Validations*. McGraw-Hill international series in software engineering. McGraw-Hill, 1993. ISBN: 9780077074104.
[151] Abraham Silberschatz, Peter Baer Galvin und Greg Gagne. *Operating system concepts*. 8. ed. Hoboken, NJ: Wiley-VCH Verl., 2009, XX, 972 S. ISBN: 978-0-470-12872-5.
[152] SPERRY RAND CORPORATION. *UNIVAC 494 Photos*. URL: http://bitsavers.trailing-edge.com/pdf/univac/494/photos/ (besucht am 26. 02. 2016).
[153] SPERRY RAND CORPORATION. *UNIVAC 494 System Description*. 1969.
[154] Ivan Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. 30 Jan 1962.
[155] Michael Swaine. *Is Your Next Language COBOL? | Dr Dobb's*. URL: http://www.drdobbs.com/architecture-and-design/is-your-next-language-cobol/210602491?pgno=2 (besucht am 26. 03. 2016).
[156] Loyd S. Swenson Jr., James M. Grimwood und Charles C. Alexander. *This New Ocean*. NASA, 1966.

[157] Ars Technica. *October | 2012 | Ars Technica*.
URL: http://arstechnica.com/science/2012/10/ (besucht am 26. 02. 2016).
[158] technikum29. *Rechner der 2. Generation*.
URL: http://www.technikum29.de/de/rechnertechnik/transistoren (besucht am 18. 03. 2016).
[159] technikum29. *Tischrechner mit Elektronenröhren*.
URL: http://www.technikum29.de/de/rechnertechnik/elektronenroehren (besucht am 18. 03. 2016).
[160] Testwell. *Testwell*. URL: http://www.testwell.fi/index.html (besucht am 01. 01. 2016).
[161] The Concord Magazine. *Charles DRAPER (1901-1987)*. 2007.
[162] James E. Tomayko. *Computers in Spaceflight: The NASA Experience*.
URL: http://history.nasa.gov/computers/contents.html (besucht am 16. 03. 2016).
[163] James Alfred Van Allen und Louis A Frank.
„Radiation around the Earth to a radial distance of 107,400 km“. In: *Nature* 183 (1959).
URL: http://www.osti.gov/scitech/biblio/4292622.
[164] Verifysoft GmbH. *Verifysoft → Verifysoft Technology GmbH*.
URL: http://www.verifysoft.com/en.html (besucht am 01. 01. 2016).
[165] Andy Verprauskus. *Ohcount download | SourceForge.net*.
URL: http://sourceforge.net/projects/ohcount/ (besucht am 03. 01. 2016).
[166] David Walden. „50th Anniversary of MIT's Compatible Time-Sharing System“.
In: *IEEE Annals of the History of Computing* 33.4 (2011), S. 84–85.
URL: http://muse.jhu.edu/journals/ahc/summary/v033/33.4.walden.html.
[167] Stephen West. *Parkes_Radio_Telescope_09.jpg (JPEG Image, 1923x2080 pixels)*.
URL: http://upload.wikimedia.org/wikipedia/commons/4/46/Parkes_Radio_Telescope_09.jpg (besucht am 26. 02. 2016).
[168] Horst Zuse. *A framework of software measurement*. New York: Walter de Gruyter, 1997.
ISBN: 9783110155877.
[169] Horst Zuse. *Resolving the Mysteries of the Halstead Measures*. 2005.
[170] Horst Zuse. *Software complexity: measures and methods*. Berlin: Walter de Gruyter, 1990.
ISBN: 9780899256405.