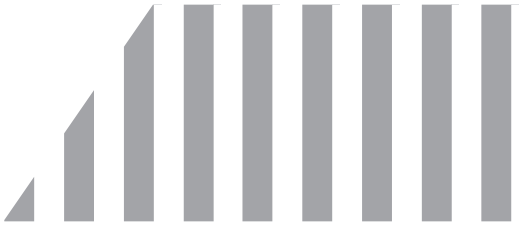
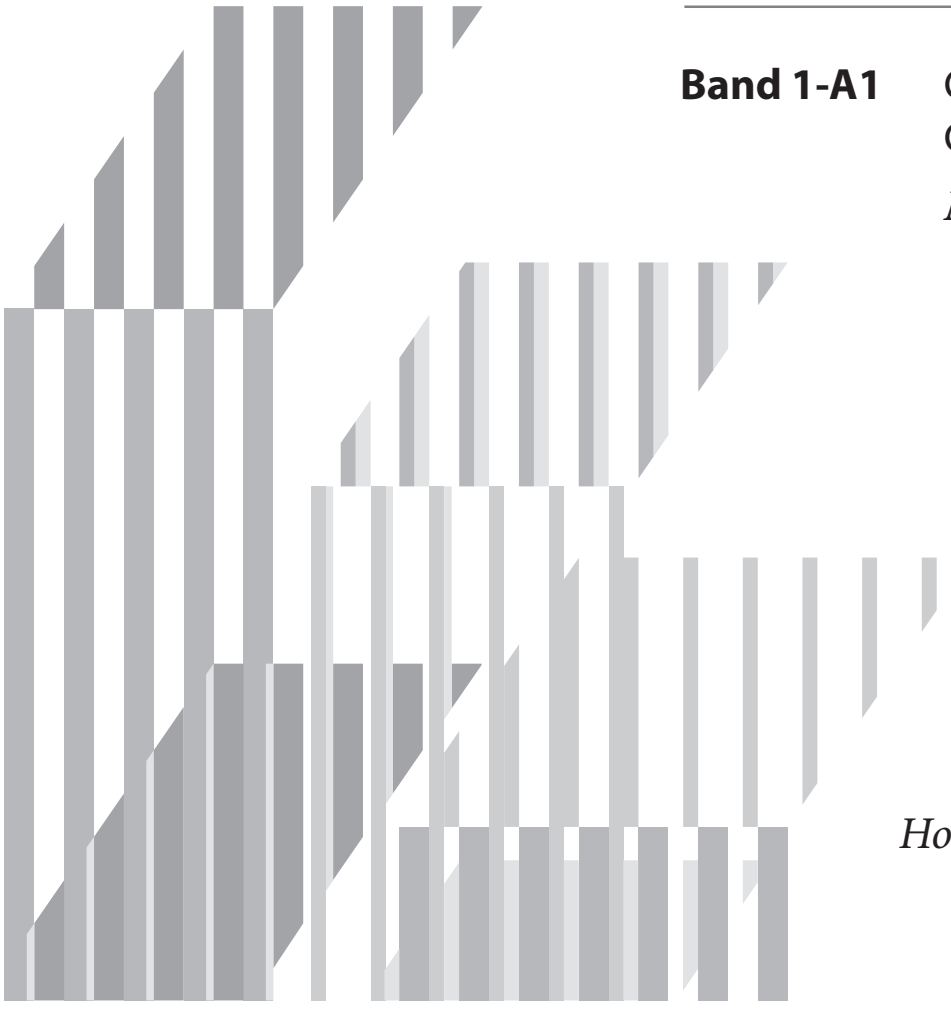
Band 1-A1



MODERNE STRUKTURIERTE PROGRAMMIERUNG

Objektorientiert und algorithmisch
ENTWERFEN | IMPLEMENTIEREN | TESTEN



Band 1-A1 Grundlagen
Gruppen
Methode

Horst van Bremen

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

Gestaltung	Katharina Schmitt
Titel- und Einbandgrafik	Monika Sylvia Gomm † 1971
Englisch-Korrektorat	David Roseveare
Verlag	HMG® Verlags-GmbH, Neue Strasse 74, 32657 Lemgo Mail-Adresse: office@hmg-verlag.de Website: www.hmg-verlag.de
Druck	Libri Plureos GmbH, Friedensallee 273, 22763 Hamburg
Version / Datum	Version 1.1.3 vom 14.2.2025
Literaturverzeichnis	siehe Kapitel „9.1 Literaturverzeichnis“ auf Seite 191 ff.
Rechtliches	siehe Kapitel „9.2 Rechtliche Hinweise“ auf Seite 193 ff.
ISBN des Paperback-Buchs	978-3-910566-21-7
ISBN der Paperback-Buchreihe	978-3-910566-00-2

Über den Autor

Horst van Bremen
Diplom 1972 an der TU Darmstadt – Elektrotechnik und Informatik
39 Berufsjahre in der IT, davon 18 als Freiberufler

IT-Systemarchitekt
Datenbankexperte

Programmiererfahrung seit 1969
Strukturierte Programmierung nach Jackson seit 1974
Freier Autor seit 2011

Vorwort zur A1-Ausgabe

Programmieren zu können, ist – nicht nur für Informatiker(innen) – zu einer Schlüsseldisziplin geworden, deren Beherrschung über Erfolg oder Mißerfolg von vielerlei Projekten entscheidet. Anfänger empfinden zunächst den intellektuellen Reiz des Programmierens als starken Antrieb, um die ersten Mißerfolge zu überwinden und etwas zu erschaffen, das – ganz ohne die Unzulänglichkeiten physischer Materialien – faszinierende Realität wird, auch wenn es selbst unsichtbar bleibt. Dieses Faszinosum bleibt auf dem Weg der Professionalisierung erhalten, mischt sich aber zunehmend mit der Furcht, das eigene Werk könne schwerwiegende, unerkannte Fehler enthalten. Es ist nicht angenehm, wenn das selbstgeschaffene Programmprodukt schon bei den ersten Tests versagt oder sogar weit später, in der Produktion, bedrohliche Konsequenzen seines Fehlverhaltens eintreten. Die Suche nach den Ursachen deckt nicht selten auf, dass die eigenen intellektuellen Grenzen enger gezogen sind, als man oder frau das gerne wahrhätte.

Diese Grenzen liegen in der menschlichen Natur. Sie können nicht abgeschafft werden, aber das ist kein Grund, in Resignation oder Zynismus zu verfallen. Es geht hier um den Umgang mit Unübersichtlichkeit und mit Komplexität, was nicht dasselbe ist. Alles, was dazu verhilft, diese Hauptursachen von Softwaredefekten zu vermeiden oder zu beherrschen, macht die Programmierprodukte fehlerfreier und damit sicherer. Eigenhändig verursachte Unübersichtlichkeit und Komplexität können durch diszipliniertes Vorgehen und den Verzicht auf vermeidbare Schnörkel eingedämmt werden, aber die in der jeweiligen Sache liegenden Ursachen für solche Schwierigkeiten sind und bleiben unabdingbar vorhanden. Ihnen gelten die von der Informatik seit Jahrzehnten entwickelten Methoden des Software Engineering.

Eine davon ist das Jackson Structured Programming (JSP), das schon seit etwa 50 Jahren existiert und nicht – wie so mancher andere Zweig am Baum der Programmiertheorien – nach einigen Jahren wieder abgestorben ist, sondern sich weiterhin einer treuen Anhängerschaft erfreut. Das JSP zeigt für eine große Klasse von Programmierproblemen den Weg zu einer fehlerfreien Lösung auf, wie anhand der zahlreichen Beispiele in den Bänden zur Modernen Strukturierten Programmierung (MSP) demonstriert wird. Es arbeitet mit einigen wenigen Grundkonstrukten, die leicht zu erlernen und dennoch imstande sind, auch in wirklich hartnäckigen Problemfällen den richtigen Weg zu weisen. Selbstverständlich kann damit nicht jede Nuss geknackt werden. Eine solche „Theorie von allem“ gibt es nicht und kann es wegen der enormen Bandbreite möglicher Aufgabenstellungen auch nicht geben. Daher werden Sie, geschätzte Leserinnen und Leser, auch zu erkennen lernen, wann JSP/MSP sinnvoll eingesetzt werden können, und wann es besser ist, sich anderen Methoden zuzuwenden.

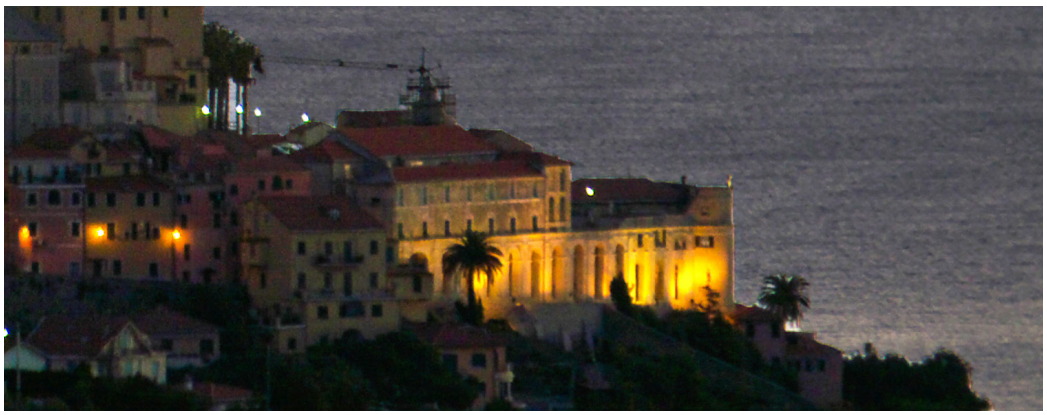
JSP und MSP verwirklichen die Prinzipien der strukturierten Programmierung, die längst als Standard bei jeglicher Softwareentwicklung gelten. Sie tun dies jedoch auf radikal andere Weise als manch anderer Lösungsansatz: Sie setzen auf den beteiligten Datenstrukturen auf und leiten daraus auf stringente Weise die Programmstrukturen her, wobei beide Arten von Strukturen mit einer einheitlichen Diagrammtechnik beschrieben werden. Die so entstandenen Programmstruktur-Skelette werden anschließend um die „Muskeln und Nerven“ – also die Operationen und Kontrollen – ergänzt, aus denen final der jeweilige Programmcode entsteht. Am Anfang steht also nicht die geniale Codierungsidee, sondern ein grafisch dargestellter Plan, der aus den Strukturen der zu verarbeitenden Daten hervorgeht.

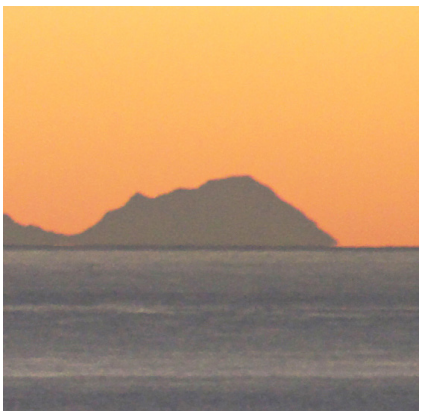
Dieser innovative Ansatz, fehlerfreie Software zu entwickeln, hat seit seiner Entstehung nichts an Kraft verloren. Nach so langer Zeit ist aber ein „Facelifting“ angebracht, das Jacksons Methode auf den Stand der Zeit bringt, wozu auch gehört, sie auf die objektorientierte Programmentwicklung anzuwenden. Der hier vorgelegte Band 1-A1 möchte Anfängern den Weg in die datenbasierte Softwareentwicklung weisen und dazu anregen, sie in der eigenen Praxis anzuwenden.

Lemgo, den 14.2.2025

Übersichtsverzeichnis Band 1-A1

Inhaltsverzeichnis Band 1-A1





01/2012 Blick auf Porto Maurizio (Imperia, Ligurien) und den Golf von Genua vor den Apuanischen Alpen

Einführung

Die Fotografie als Kunst des Sichtbaren und das Programmieren als Kunst des Unsichtbaren haben im digitalen Zeitalter nur auf den ersten Blick nichts gemeinsam. Was aber hat die frühmorgendliche Aufnahme auf der linken Seite mit dem Thema dieser Buchreihe, der Modernen Strukturierten Programmierung (MSP), zu tun – abgesehen von der beliebten Metapher vom neuen Tag, der eine neue Zeit symbolisiert?

- *Die Freude über die Schönheit unseres Planeten ist nicht mehr naiv, seit wir von der Kugelgestalt der Erde wissen und berechnen können, was wir auf der anderen Seite des Golfs von Genua in circa 180 Kilometern Entfernung sehen. Wissenschaft durchdringt unsere Wahrnehmung, also längst auch die anfängliche intuitive Auffassung des Programmierens als kreativer Akt, bei dem es hauptsächlich auf Genialität ankommt.*
- *Ein durchgehend digitaler Prozess führt von diesem Foto dahin, dass Sie, geschätzte Leserinnen und Leser, jetzt diesen Band in den Händen halten. Das hätten wir uns allesamt in früheren Jahren nicht vorstellen können, als noch auf Filmen analog fotografiert wurde und Bücher im Bleisatz per Offsetdruckverfahren entstanden. Dieser epochale Umbruch ist noch ganz nahe, vor allem wenn man bedenkt, dass das analoge Fotografieren in der ersten Hälfte des 19. Jahrhunderts entwickelt wurde und bis in unser Jahrtausend dominierte.*
- *Erst die Entwicklung leistungsfähiger digitaler Kameras, schneller Computer, sowie die digitalisierte Medienproduktion machten diesen Umbruch möglich. Die dafür erforderlichen optischen, elektronischen und mechanischen Techniken wären jedoch ohne die Fortschritte der angewandten Informatik nutzlos, auf denen auch die – für diese Buchreihe verwendeten – komplexen Programme zur Bild- und Textbearbeitung beruhen. Erst das hochdisziplinierte Zusammenwirken der Hardware mit der sie steuernden Software erbringt zuverlässig die gewünschten Ergebnisse.*
- *Die Aufnahme wurde im kameraspezifischen RAW-Format gemacht, das man sich als eine pixelgenaue Aufzeichnung der Grundfarben R(ot) - G(rün) - B(lau) und der jeweiligen Helligkeiten vorstellen kann. Ein Dienstprogramm konvertierte diese Daten in das JPEG-Format, bevor das Bild mit Photoshop® bearbeitet und dann in das Manuskript eingebunden wurde. Alle diese Verarbeitungsschritte erfordern absolute Präzision, die letztlich nur durch die strukturierte Programmierung gewährleistet werden kann.*

Zuverlässigkeit ist das entscheidende Schlüsselwort, auf dem die MSP-Buchreihe aufbaut. Lange genug war der Prozess der Softwareentwicklung ein mühsamer und fehleranfälliger Vorgang, dessen bestmögliche Bewältigung allenfalls genialen Programmierer(inne)n zugetraut wurde. Alle anderen, also auch ich, der Autor, schrieben Programme, die meistens gut funktionierten, aber halt nicht immer. Der Software-Fehlerteufel plagt unsere Branche zwar nach wie vor, aber dank der Entwicklung solider, verständlicher Methoden sind sichere und hochkomplexe Systeme möglich, ohne dass es dafür Genies bräuchte.

Die nachfolgend vorgestellte Strukturierte Programmierung ist eines der Fundamente, auf denen diese Erfolge stehen. Jedefrau und jedermann, die oder der Software herstellt, sollte sie kennen und anwenden, wo immer das sinnvoll ist. Es handelt sich hier um Basiswissen der Informatik, dessen Nutzen auch noch nach Jahrzehnten immer wieder aufs Neue bestätigt wird. Dieses Wissen droht jedoch in Vergessenheit zu geraten, wird es nicht immer mal wieder auf den Stand der Zeit gebracht. Das geschieht in diesem Band und den ihm folgenden. Die MSP-Buchreihe ist für das Selbststudium konzipiert, damit niemand für das Erlernen der strukturierten Methode auf teure Kurse angewiesen ist. Daher wird alles ausführlich erklärt, und das, zusammen mit dem Anspruch, auch komplexe Beispiele zu zeigen, erklärt den erheblichen Umfang, der sich zwangsläufig auch auf die Länge dieser Einführung auswirkt. Die Buchreihe besteht daher aus drei Staffeln mit Bänden, die paarweise zusammengehören, nämlich je ein Methoden-Band und ein Praxis-Band mit den zugehörigen Programmbeispielen.

Wohl jede Autorin und jeder Autor kennt die Selbstzweifel, wenn das erste eigene Buch sich der Fertigstellung nähert und sein künftiges Publikum ins Blickfeld rückt: Wird es überhaupt Leser(innen) finden? Wie werden die Kritiken lauten? War die viele Arbeit dafür sinnvoll aufgewendet? Diese Fragen beschäftigten mich um so mehr, weil ich bis heute über ein Jahrzehnt lang an diesem Buch und den fünf ihm folgenden Bänden arbeitete (und noch arbeite). Meine Freunde und Bekannten, die von dem Buchprojekt wissen, sowie meine Verwandschaft dürften mich wahlweise als verrückt, verbohrt, stur, einseitig, ungesellig, weltfremd und manches mehr angesehen haben. Zugegeben, das stimmt alles. Anders ist aber ein so großes Kaliber nicht zu bewältigen. Was mit einem konservativen Ansatz begann, nämlich die unschätzbar wertvolle Methode von Michael Anthony Jackson[1], vor dem Vergessenwerden zu bewahren, hat sich als eine Reise in Gefilde entpuppt, von denen selbst ich – auch nach Jahrzehnten der Arbeit mit seiner Methode – keinerlei Ahnung hatte.

Um so lange an einem Projekt zu arbeiten, ist ein starker und stetiger Antrieb unerlässlich. Dieser speist sich zum einen aus etwas, das ich nicht anders als Dankbarkeit nennen kann – Dank an M. A. Jackson, dessen Ideen einen großen Teil meines Berufslebens geprägt haben. Zum anderen denke ich an professionelle Programmierer(innen) und Informatiker(innen), sowie an Anfänger oder Umsteiger aus anderen Berufsfeldern, für die seine Methode sich als ähnlich wertvoll erweisen kann, die aber noch nichts davon wissen. Da ich ein technischer Autor bin, also kein Romancier, der sich in andere Personen versetzen kann, habe ich das geschrieben, was ich selbst gern gelesen hätte – in der Hoffnung, damit möglichst Viele zu erreichen.

Warum heisst die Buchreihe „Moderne Strukturierte Programmierung"? Für das erste Wort gibt es ein historisches Vorbild, und das stammt von Edward Yourdon[2] – ebenfalls ein Autor, dem ich viel verdanke. Er schrieb das 1989 erschienene Buch „Modern Structured Analysis" (MSA) [1][3], in dessen Vorwort er übrigens ebenfalls seine Selbstzweifel bekundete. Völlig zu Unrecht, meine ich, denn seine Idee des Event Partitioning, also der Strukturierung von Anwendungssystemen als Ensemble von Prozessen, die auf Ereignisse reagieren, befreite die Systemanalytiker vom Zwang der hierarchischen Zerlegung à la SADT[4], der in den späten 1970er-Jahren meinungsführend war. Yourdons MSA hat sich in der Praxis überzeugend bewährt. (Am Rande: „Modern" oder „Neu" lauten seit der Antike die Schlüsselworte ewiger Jugend, was bei technischen Büchern allerdings als kokett erscheint, weil sie doch früher oder später einer Überarbeitung bedürfen).

Das von M. A. Jackson entwickelte „Jackson Structured Programming" (JSP) erfreute sich ab der Mitte der 1970er Jahre zumindest ein Jahrzehnt lang großer Beliebtheit, und es besitzt auch heute noch eine treue Anhängerschaft. Das zeigt sich unter anderem darin, dass das JSP-Buch von Dr. Klaus Kilberth 2001 in der 8. Auflage erschien [2] und mittlerweile als e-Book erhältlich ist.

Die hier vorgelegte Buchreihe über die „Moderne Strukturierte Programmierung" (MSP) intendiert die Aktualisierung und Weiterentwicklung des JSP, das 1974 durch Jacksons Buch „Principles of Program Design" [3] erstmals einer breiten Fachöffentlichkeit bekannt wurde. Es hatte einen solch durchschlagenden Erfolg, weil es erstmals einen sowohl intuitiv einleuchtenden als auch theoretisch wohlfundierten Weg von der Aufgabenstellung zum fertigen Programmentwurf wies. Jacksons grundlegende Idee der Synchronisierung hierarchisch-grafischer Datenstrukturen anhand sogenannter Korrespondenzen und die daraus folgende Ableitung der – im selben Stil aufgebauten – Programmstrukturen löste auf Anhieb viele Probleme, die auch langerprobte professionelle Anwendungsentwickler immer wieder in Bedrängnis gebracht hatten. Der Vater der strukturierten Programmierung ist aber nicht Jackson, sondern Edsger Wybe Dijkstra[5].

Publikationen über das JSP erschienen auch in Deutschland zunächst nur in englischer Sprache. Das Informatik-Kolleg der Gesellschaft für Mathematik und Datenverarbeitung[6] (GMD) in Darmstadt übersetzte und konsolidierte diese Texte, um das JSP in die dort stattfindende Ausbildung wissenschaftlich-technischer Assistent(inn)en zu integrieren. Es bot diesen Kursus aber auch den wissenschaftlichen Mitarbeitern der GMD an. Auf diesem Weg kam ich[7] als Mitglied der TR 440[8]-Systemgruppe mit dem JSP in Berührung – voller Misstrauen gegen diese angeblich revolutionäre Neuerung, die doch nur ein Aufguss bereits bekannter, untauglicher Ansätze zur strukturierten Entwicklung von Software zu sein schien. Nach dem Kursus waren diese Vorurteile allerdings komplett beseitigt und fortan gehörte das JSP zu meinem beruflichen Grundwissen.

Endnoten: siehe Seite 14

1.1 JSP-Literatur

Die im GMD-Informatik-Kolleg tätigen Autoren R. Legde und K. Truöl erarbeiteten 1977/78 eine umfangreiche JSP-Einführung mit dem Titel „Problem- und datenorientierter Entwurf strukturierter Programme“ [4]. Auf diesem äußerst nützlichen Dokument basiert die Moderne Strukturierte Programmierung in vielerlei Hinsicht. Erst 1979 erschien Jacksons Buch auf Deutsch mit dem Titel „Grundsätze des Programmentwurfs“ [5]. Diese Darstellung seiner Methode erforderte unter anderem, sich mit COBOL auszukennen, war also nicht so leicht wie [4] zu lesen und zu verstehen. Heutige Leser(innen) dürften sich darüber hinaus daran stören, dass es vorwiegend die Stapelverarbeitung thematisiert, zu der schon bald danach die Transaktionsverarbeitung und in neuerer Zeit die Web-Technologien hinzugetreten sind. Auch das Thema Datenbanken sucht man darin vergeblich – kein Wunder, steckten diese doch 1974 noch in den Kinderschuhen. Es wäre daher grob ungerecht, Jacksons bahnbrechende Gedanken aus der heutigen Perspektive zu beurteilen.

Dieselben heutigen Leser(innen) haben aber vermutlich schon eine Menge IT-Literatur konsumiert, die nach dem Prinzip arbeitet, alles in kleinste Lerneinheiten zu zerlegen und so den Weg zum Ziel möglichst eben zu gestalten. Jackson mutete seinen Leser(inne)n dagegen auch steile Anstiege und jähe Kurven zu, wenn ihm das für seine Argumentation dienlich erschien. Man muss sein Buch daher sehr aufmerksam lesen, um seine Gedankengänge zu verstehen und an seinen Einsichten teilzunehmen. Vielleicht liegt es daran, dass seine Absicht, die bis dahin üblichen Programmiertechniken als verkorkst zu entlarven und ein neues Fundament für die Programmentwicklung zu legen, nicht in dem Umfang zum Tragen kam, wie es für eine weite Verbreitung des JSP notwendig und sinnvoll gewesen wäre. Bis auf den heutigen Tag belegen zahllose Code-Fragmente, die man im Internet studieren kann, dass die Grundideen Jacksons leider weithin unbekannt geblieben sind.

Wer die Geduld aufbrachte, Jacksons Beispielfällen und seiner meist zwingenden Argumentation zu folgen, der wurde reich belohnt. Immerhin war – und ist – das JSP die einzige Software-Engineering-Methode, die es ermöglicht, auf einem plausiblen Weg von den Datenstrukturen zur Programmstruktur zu gelangen. Für schnelle Leser(innen) ist das Buch jedoch leider völlig ungeeignet. Insbesondere fehlt eine übersichtliche Zusammenfassung (die für die Mitte des Bands 5 dieser Buchreihe geplant ist). Man muss sich daher die relevanten Aspekte der Methode über viele Seiten verstreut zusammensuchen und kann nicht sicher sein, alles Wesentliche wirklich erfasst zu haben. Positiv ist dagegen, dass Jackson zahlreiche Beispiele anführte, um seine Ideen zu erklären. Das JSP ist ja eine auf Abstraktion basierende Modellierung, was die Gefahr in sich birgt, sich dem Zauber des Abstrakten hinzugeben und darüber die ganz realen Programmierprobleme und -pannen aus dem Blickfeld zu verlieren.

COBOL ist ein Akronym, das am Ende der 1950er Jahre aus der Bezeichnung „Common Business-Oriented Language“ gebildet wurde. Diese krude Programmiersprache war all denjenigen fremd, die – wie ich – aus der Welt von ALGOL, FORTRAN, BCPL und diverser Assemblersprachen kamen. Unglücklicherweise – für unsereinen – war dieses alte COBOL jedoch Jacksons wichtigstes Darstellungswerkzeug. Es lieferte ihm die speicherinternen Datendefinitionen und war die Grundlage für zahlreiche Codebeispiele. Erstaunlicherweise hat Jackson die von ihm entwickelte grafische Sprache, die Struktogramme, nicht so intensiv in den Vordergrund seiner Argumentation gestellt, wie sie es meines Erachtens verdient hätte. Zudem fehlt dem von ihm vorgeschlagenen Pseudocode namens Schematische Logik die erforderliche Eleganz – bei hohem Schreibaufwand. Somit hatte sein Buch aus meiner Sicht einige didaktische Mängel, was der Verbreitung des JSP ausserhalb der kommerziellen Programmierung vermutlich nicht förderlich gewesen ist. Die damaligen objektorientierten Sprachen wie zum Beispiel Simula und Smalltalk spielten in Jacksons Buch keine Rolle. Das JSP wurde vermutlich auch deshalb als eine rein algorithmische Entwurfsmethode aufgefasst, obwohl, genau genommen, das alte COBOL – heute COBOL I genannt – den damaligen algorithmischen Sprachen weit unterlegen war.

1.2 Wo stehen wir heute?

Seit Jacksons englischem Buch [3] sind fünfzig Jahre vergangen, in denen sich die IT weltweit zu einer der führenden Branchen entwickelt hat. Damit einher ging einerseits eine enorme Differenzierung der Plattformen und Anwendungsfelder, andererseits eine damals unvorstellbare Steigerung der Leistungsfähigkeit von Computersystemen. Jede(r) kann heute ein Vielfaches der Speicherkapazität und der Rechenleistung eines TR 440- oder IBM-System/360[9] in ihrem/seinem Smartphone mit sich herumtragen – allerdings ohne damit Hunderte Benutzer gleichzeitig online zu bedienen und parallel dazu aufwendige Stapelverarbeitungen auszuführen. Der größte Teil der Smartphone-Rechenleistung wird für dessen grafische Oberfläche benötigt.

Währenddessen fand ein beeindruckender Siegeszug der Objektorientierung statt, dem der Großteil zahlreicher Innovationen zu verdanken ist, die für ihre Nutzer mittlerweile als unverzichtbare Errungenschaften gelten. Dieser Aufschwung war nur auf der Basis massiver methodischer Anstrengungen möglich, die unter den OO-Kürzeln OOA (Object-Oriented Analysis), OOD (Object-Oriented Design) und OOP (Object-Oriented Programming) zusammengefasst werden können. Ohne diese theoretische Basis konnten die heutigen komplexen OO-Systeme nicht entwickelt werden.

Die alte algorithmische Programmierung bildet zwar immer noch das Rückgrat der datenbank- und transaktionsbasierten Hochleistungssysteme, aber diese gehören überwiegend zum Gebiet der Mainframes, deren Verbreitung auf Großorganisationen beschränkt ist. In diesem Sektor der IT ist bedauerlicherweise eine erhebliche Verwilderung der guten Sitten bei der Programmentwicklung und –wartung festzustellen, wie ich es in meiner eigenen Praxis oftmals erlebt habe. Aus den Auseinandersetzungen um die „einzig wahre" Softwaretechnologie in den 1970er Jahren ging nämlich keine einzige Methode als Sieger hervor, auch das JSP nicht. Zu divergent waren schon damals die Anforderungen an eine solche „Theorie von Allem", und zu wenig flächendeckend konnte sich die strukturierte Programmierung – in welcher Ausprägung auch immer – etablieren. Die Folgen: Wildwuchs, Zynismus, Methodenaversion, Bastlermentalität, bestenfalls formale – und institutionell erzwungene – Strukturierung mittels Programmgeneratoren.

In diesen vier Dekaden versanken auch die alten Namen und Abkürzungen: Michael Jackson, das war doch ein berühmt-berüchtigter Pop-Sänger, und JSP bedeutet JavaServer Pages (alt) beziehungsweise Jakarta Server Pages® (neu), nicht wahr? Interessiert sich überhaupt noch jemand ausser einer kleinen Gruppe nostalgischer Alt-Programmierer(innen) für die strukturierte Programmierung? Ist sie nicht rettungslos dem algorithmischen Denken verpflichtet, das angesichts der Erfolge der Objektorientierung als eine verflossene Entwicklungsphase gilt, geradezu als Zeit der Programmier-Dinosaurier? Jede(r) kennt doch diese Tiere, speziell die Pflanzenfresser: massiger Körper, kleiner Kopf, folglich wenig Intelligenz. Diese beleidigende Parallele erscheint zudem als wohlverdient angesichts der Fehlschläge, die wegen der allzu leicht entstehenden Undichtigkeiten in algorithmisch aufgebauten Programmen vielerorts aufgetreten sind. Die Objektorientierung verspricht dagegen, vor allem durch die obligate Kapselung von Daten und den ihnen zugeordneten Methoden in Klassen beziehungsweise Objekten, weit besser strukturierten Code zu ermöglichen. Das ist eine gute Nachricht.

Die schlechte Nachricht lautet: Auch mit den besten objektorientierten Sprachen und Methoden ist es anscheinend immer noch eine Kunst, ein Programm so zu schreiben, dass es gut gestaltet ist und seine Aufgabe einwandfrei löst, also keinen Unfug macht und auch nicht abstürzt. Kunst kann zwar imitiert, aber nicht gelehrt werden. Jemand ist ein – werdender – Könner, oder ist es nicht. Kunst ist vor allem ein Prozess, dessen Ergebnis weder vorhersagbar noch personenunabhängig ist. Dieses Paradigma, dem der Typus des freien Programmierkünstlers entspricht, ist das exakte Gegenteil des Ingenieursgedankens, dass Programme wie andere technische Erzeugnisse konstruiert werden können. Konstruktion ist keine Kunst. Sie ist angewandte Wissenschaft.

1.3 Programme mit MSP konstruieren

Was ist also modern an der Modernen Strukturierten Programmierung? Sie baut auf dem Jackson Structured Programming auf und verfolgt dieselbe Absicht: Programme sollen nicht erfunden, sondern konstruiert werden. Hierfür integriert die MSP-Buchreihe die folgenden Aspekte:

- Durch ein Facelifting des JSP und die intensive Verwendung von Struktogrammen strebt die MSP an, diese Software-Engineering-Methode auch ausserhalb der kommerziellen Programmierung zu etablieren. Anstelle von COBOL oder Jacksons Schematischer Logik dient ein moderner Pseudocode als Beschreibungssprache, der auch eine flexible Syntax für Datendefinitionen umfasst. Neu ist auch die Nutzung der Struktogramme für die Durchlaufanalyse vor der Implementierung, mit der Entwurfsfehler schon frühzeitig aufgedeckt werden können – ein veritabler Schreibtischtest.
- An vielen, teilweise ziemlich komplexen Beispielen zeigt die MSP, dass auf der Basis von M. A. Jacksons Struktogrammen mit Operationen und Kontrollen sowohl – wie bisher – algorithmische als auch – neuerdings – objektorientierte Implementierungen hergeleitet werden können. Für letztere ist ein zusätzlicher OO-Entwurfsschritt erforderlich, dessen Verlauf durch die vorangehende JSP-Modellierung gesteuert und damit weithin vorhersagbar wird. Das eröffnet die Möglichkeit, existierende algorithmische Programme zunächst mit Struktogrammen zu modellieren und dann in objektorientierte Programme zu transformieren – eine strategisch wichtige Basis für die Migration von Altsystemen auf neue Plattformen. Zugleich werden Barrieren abgebaut, die bislang manche Entwickler(innen) davon abhielten, die jeweils beste Lösung für ein Programmierproblem sowohl auf der algorithmischen als auch auf der objektorientierten Seite zu suchen. Noch wichtiger ist mir allerdings, dass werdende professionelle Anwendungsentwickler zahlreiche Anregungen erhalten, wie sie das JSP für ihre eigene Praxis zum Tragen bringen und dabei ein hohes qualitatives Niveau erreichen können.
- Die Entwicklung der modernen, GO-TO-freien Programmiersprachen brachte es mit sich, dass es schwierig oder sogar unmöglich zu werden schien, damit Jacksons Problemlösungsverfahren „Backtracking" und „Programminversion" anzuwenden. Die MSP zeigt einen Konstruktionspfad auf, der es ermöglicht, in *jeder* modernen Programmiersprache diese beiden Verfahren anzuwenden.
- Viele Software-Engineering-Methoden scheiterten daran, dass sie nicht miteinander kompatibel waren und zudem mit dogmatischer Härte gelehrt wurden. Zwanghaftes Entwerfen, das starren Mustern verpflichtet ist, schadet jedoch der Qualität der Ergebnisse und schöpft das Potential des JSP nicht aus. Die MSP bettet Jacksons Methode daher in ein Umfeld ein, das von vielfältigen, konkurrierenden Ansätzen durchdrungen ist. Dabei geht es darum, einerseits den Kern der Methode zu verdeutlichen, andererseits das JSP so flexibel wie möglich anzuwenden. Starre Begrifflichkeiten und Regeln, die zu unnötigen Umwegen oder künstlichen Anpassungen an eine verabsolutierte Methodik führen, werden daher aufgegeben. Ein Beispiel dafür ist die Einbindung der Automatentheorie dort, wo eine JSP-konforme Ableitung des Codes – scheinbar oder tatsächlich – als krampfhafte Übung erscheint.
- Die Aktualisierung des JSP und die neuen methodischen Elemente, welche die MSP einbringt, bedeuten auch, neue grafische Elemente vorzuschlagen, die für mehr Übersichtlichkeit sorgen, Irrtümer vermeiden helfen, oder die Qualität der Dokumentation erhöhen. Durchgehend liegt der Akzent darauf, die bei jedem Entwurf anfangs noch weichen Beschreibungselemente zu festen, präzisen Definitionen weiterzuentwickeln, die den Implementierungen zugrunde gelegt werden. Das heißt, sukzessive den Interpretationsspielraum der Programmierer(innen) und damit die deutungsbedingten Irrtumsmöglichkeiten einzuschränken. In diesem Kontext verabschiedet sich die MSP auch von fragwürdigen Thesen aus der JSP-Entstehungsgeschichte, die es manchen Modellierern möglich machten, die Last ihrer Fehlentscheidungen oder ihrer Bequemlichkeit den Programmierer(inne)n aufzubürden.

- Die teilweise extreme Arbeitsteilung in der IT führt vor allem bei Neulingen zu einer verzerrten Wahrnehmung des Entwurfsprozesses. Die MSP-Buchreihe nimmt sich daher die Freiheit, anhand sorgfältig ausgewählter Beispiele den gesamten Weg von der Idee bis hin zu Implementierung und Test darzustellen. Die strukturierte Vorgehensweise wird dabei in mehreren Facetten gezeigt: als Entwicklung aus einem Guss (der traditionelle Weg), als inkrementelle Vorgehensweise, als schrittweise Weiterentwicklung, als Erstellung von Service-Modulen für nicht-JSP-basierten Code, als Revision unbefriedigender Lösungen, als Korrektur eines Irrwegs oder auch als Migration auf eine andere Programmierplattform.
- Die meisten Beispiele wurden in C oder Java® programmiert und getestet, nur eines in COBOL II. In vielen Fällen wird zunächst die algorithmische Realisierung gezeigt, und dann der Übergang zu einer objektorientierten Lösung im Detail vollzogen. Teilweise blieb es bei einer der beiden Alternativen, um den Umfang der Buchreihe nicht unnötig aufzublähen, oder um spezielle Möglichkeiten zu nutzen. Java-Programme dienen zum Beispiel dazu, Mails aufzubauen und zu versenden, oder die Verzeichnisstrukturen des File-Systems auszuwerten. Der Umfang der Beispielprogramme, die zudem nicht jeden interessieren dürften, machte die Auslagerung ihres Codes, der zugehörigen Beschreibungen und des größten Teils der Testdokumentation in den den jeweils zweiten Band der drei Buchstaffeln erforderlich. Diese Aufteilung ermöglicht es, die jeweils zusammengehörenden Bände nebeneinanderzulegen, um die Übertragung des Entwurfs in den Code zu studieren.
- Das JSP beruhte technisch auf sehr einfachen Datenorganisationen, vorwiegend auf sequentiellen Dateien. Die Entwicklung von Datenbanksystemen begann erst damals und es ist bislang kein Ende abzusehen. Dabei entstand eine enorme Vielfalt von technischen Lösungen, die bei großen und mittleren Rechnern vor allem auf die Interaktion mit Transaktionssystemen hin ausgelegt wurden. Dort treten Konkurrenz- und Konsistenzprobleme auf, die für die verlässliche Funktionalität der Anwendungssysteme zwingend gelöst werden mussten. Dagegen spielen konkurrierende Zugriffe auf Desktop-Systemen und erst recht auf portablen Rechnern keine wesentliche Rolle. Auf diesen Maschinen haben sich hauptsächlich relationale Datenbanksysteme durchgesetzt, während auf IBM®-Großrechnern neben dem relationalen DB2® auch das ältere IMS/DB® und dessen quasi-relationaler Konkurrent ADABAS® im produktiven Einsatz sind.

  Der datenorientierte Entwurf nach Jackson muss natürlich je nach Datenbanksystem adaptiert werden, um dessen Zugriffsformen aufzugreifen, die Aufteilung der bislang kontinuierlichen Verarbeitung in Segmente – Transaktions- oder COMMIT-Intervalle – zu berücksichtigen, und beispielsweise für Restarts die erforderlichen Vorkehrungen zu treffen. Einerseits ist es offensichtlich unmöglich, in dieser Buchreihe das gesamte Spektrum der modernen Datenbanksysteme auf seine JSP-Relevanz hin zu untersuchen und die Konsequenzen für die Modellierung der Daten- und Programmstrukturen zu diskutieren. Andererseits wurde wegen der hohen Bedeutung, die insbesondere die RDBMS – Relational Database Management Systems – für die heutige Anwendungsentwicklung besitzen, diese Thematik in der zweiten und dritten Staffel dieser Buchreihe aufgegriffen.

Ihr Eindruck von alledem als Leser(in) mag sein, dass Sie sich durchaus positiv angesprochen, aber vom Umfang der MSP-Buchreihe abgeschreckt fühlen. In der Tat muss ich zugeben, dass mehrere Beispiele sich als weit raumfordernder erwiesen haben, als es ursprünglich geplant und abzusehen war. Der Spitzenreiter ist sicherlich der internationale Schul-Sportwettbewerb mit dem abschließendem Versand der Ergebnisse per E-Mail. Gerade diese Fallstudie belegt jedoch nicht nur eindrucksvoll die Ausweitung des JSP auf das objektorientierte Gebiet, sondern ermöglicht auch zahlreiche Einsichten in die damit einhergehende Anpassung an die durch Java definierte Basismaschine – ein realitätsnahes Beispiel für eine Migration auf eine neue Plattform.

Alles in allem deckt die MSP-Buchreihe nicht nur den gesamten Umfang des JSP ab, wie ihn Michael A. Jackson definiert hat, sondern sie führt auch vielfältige Modifikationen und Ergänzungen ein, die zusammengenommen das Attribut „modern" rechtfertigen – von den umfangreichen Implementierungs- und Testbeispielen mal völlig abgesehen. Das heißt jedoch auch, dass viele Programmierthemen, welche die IT seit

damals entdeckt hat, nicht berücksichtigt werden konnten. Es wären einfach zu viele, um sie alle zu würdigen. Ausserdem hat es sich in Jahrzehnten meiner JSP-Anwendung auf sehr verschiedenen Gebieten der Informatik gezeigt, dass das JSP deshalb keiner Erweiterung bedurfte – von den Datenbank-Aspekten mal abgesehen.

1.4 ... und die Folgen

Jede Medizin hat Risiken und Nebenwirkungen. Das gilt auch für „Medikamente gegen schlechte Software", also für alle Methoden, die darauf abzielen, den Entwurfsprozess an objektiven Kriterien statt an nebulöser Könnerschaft auszurichten. Das primäre Risiko besteht bei JSP/MSP darin, damit Probleme lösen zu wollen, für die der datenorientierte Ansatz nicht geeignet ist, zum Beispiel in der Robotik – aber das ist im Allgemeinen leicht zu erkennen. Zwar erweitert die MSP den Lösungsraum um die Modellierung von Programmstrukturen, die nicht aus Datenstrukturen abgeleitet werden (können), aber damit geht die Führungswirkung der korrespondenzgesteuerten Verschmelzung von Daten-Struktogrammen verloren. Dennoch sind solche nicht-konservativen Entwürfe der ad-hoc-Programmierung meines Erachtens durchaus vorzuziehen, weil sie es ermöglichen, die so entstandenen Programmstrukturen auf einem guten qualitativen Niveau zu untersuchen – auch und gerade um Fehler aufzudecken. Ob dieser Nutzen es rechtfertigt, den Aufwand der JSP/MSP-Modellierung zu treiben, kann nur von Fall zu Fall entschieden werden.

Damit eng verwandt ist das Risiko des Denkens in starren Kategorien. Ist man erst einmal an eine Entwurfsmethode gewöhnt, liegt es nahe, neue Probleme nur noch durch die Brille dieser Methode zu betrachten. Schlimmstenfalls wird daraus ein Prokrustesbett[10]. In diese Falle sind viele Software-Engineering-Ansätze getreten, die ihre Anhänger auf einen strikten Regelgehorsam verpflichteten und neben sich keine Konkurrenz duldeten.

Wenn JSP/MSP die richtige Methode für Ihr Programmierproblem ist, dann gibt es ein soziales Risiko: Sie könnten zum Aussenseiter werden. Solange Ihr Management bis hinab zum Projektleiter entweder andere Methoden oder gar keine favorisiert, wobei Letzteres im Bereich der algorithmischen Programmierung leider die Regel ist, fallen Sie möglicherweise als Abweichler auf. Anstatt nur die geforderten Dokumente und danach direkt den Programmcode zu produzieren, verschwenden Sie aus der Sicht Anderer, auch Ihrer Kollegen, Ihre Zeit mit Struktogrammen. Der Erfolg wird Ihnen zwar später recht geben, aber bis dahin müssen Sie mit dem Argwohn derer leben, die Ihr Vorgehen nicht verstehen oder Ihre Präferenz nicht teilen – und des Öfteren ausserdem nichts dazulernen wollen, sondern mit dem Erreichten zufrieden sind.

Auch die Nebenwirkungen sind nicht harmlos: Sie verändern sich. Anstatt sich freudig auf die Produktion von Code zu stürzen und Ihr Ego durch großartige Konstruktionen zu bestätigen, die womöglich niemand sonst richtig zu würdigen weiß, gehen Sie einen zunächst als mühsamer erscheinenden Weg. Sie zeichnen Datenstrukturen und ermitteln Korrespondenzen, bauen Programmstrukturen auf, definieren die Auswahl- und Schleifenkontrollen, erstellen die Liste der Operationen und ordnen diese den richtigen Stellen im Programm-Struktogramm zu. Wenn Sie ein OO-Programm entwerfen, schließen Sie den dafür erforderlichen Entwicklungsschritt an und runden Ihre Arbeit mit einem Klassendiagramm ab – möglicherweise, bevor Sie eine einzige Zeile Code geschrieben haben. Es ist auch ein exploratives Vorgehen denkbar, bei dem Sie die Entwurfsschritte iterativ durchlaufen und zwischendrin immer wieder programmieren, gemäß dem Motto von Grady Booch[11] „Design a little, program a little" in seinem Buch „Object-Oriented Design with Applications" [6].

Auf jeden Fall verliert das Programmieren dadurch an Reiz. Der eigentliche Entwurf findet im Vorfeld statt und das Programmieren dient „nur noch" dazu, ihn umzusetzen beziehungsweise die Umsetzbarkeit Ihrer Ideen zu prüfen. Der Suchtfaktor des Erschaffens neuer Welten als Programmierer(in) fällt dadurch – leider – weitgehend weg. Zugleich verändert das disziplinierte, methodische Vorgehen Ihre Arbeitsweise insgesamt. Wo vorher einige Skizzen und möglichst unpräzise Vorgaben Ihnen freie Hand ließen, werden Sie nun bessere Analyseergebnisse erwarten oder gegebenenfalls vorhandene Lücken selbst zu schließen versuchen, damit

Ihr Entwurf überhaupt fertiggestellt werden kann. Sie werden kritischer und sind weniger bereit, analytische Schlampereien zu akzeptieren.

Rundweg positiv ist die Nebenwirkung, dass Sie weniger Angst verspüren werden. Schon während des Entwurfs, der Programmierung und dem Test sind Sie entspannter, weil es nun nicht mehr hauptsächlich auf Ihr Können als Entwickler(in) ankommt, sondern darauf, alles richtig modelliert zu haben. Anstatt sich zu sorgen, ob Sie wirklich alle denkbaren Datenkonstellationen geprüft haben und ob sämtliche Pfade in Ihrem Programm häufig genug korrekt durchlaufen wurden, gibt Ihnen JSP/MSP die Sicherheit, alle relevanten Aspekte beachtet zu haben. Das bislang Undenkbare tritt ein: Sie geben ein Programm in Produktion und es läuft dauerhaft fehlerfrei.

Auch in der adaptiven Wartung, also der Weiterentwicklung, entfalten sich die Vorteile des strukturierten Programmierens: Anstatt im Code nach denjenigen Stellen suchen zu müssen, auf welche die neuen Anforderungen sich auswirken, und dabei auf gar keinen Fall etwas zu übersehen, greifen Sie auf die Dokumentation zurück. Sie machen sich wieder mit den Entwurfsdokumenten vertraut, identifizieren dort die Auswirkungen dieser Anforderungen, erstellen neue Versionen der Struktogramme, sowie der begleitenden Listen von Kontrollen und Operationen und ordnen diese Änderungen den korrespondierenden Code-Partien zu. Nach der Umsetzung finden die Tests auf der Basis der bisherigen beziehungsweise der erweiterten Datenkonstellationen statt. Wenn Sie bei alledem das Richtige getan haben, wird die neue Programmversion ebenso fehlerfrei wie die alte arbeiten.

„Zero defects" ist das eigentliche Ziel aller Software-Entwicklungsmethoden. Ein fehlerfreies Programm enthält nicht nur den korrekten Code, der die Anforderungen umsetzt, sondern es ist auch gut gegliedert und – intern wie extern – dokumentiert. Mit JSP/MSP steht Ihnen nun eine mächtige Zero Defects-Methode zur Verfügung, die sich nicht absolut setzt, sondern sich mit anderen Ansätzen verträgt. Aufgrund meiner jahrzehntelangen positiven Erfahrungen mit dieser Methode, die mir oftmals meine berufliche Existenz gerettet hat, wünsche ich Ihnen viel Freude beim Lernen und noch mehr Erfolg bei der Anwendung in Ihrer Praxis.

1.5 Die Buchreihe

Wie schon erwähnt, besteht die Buchreihe aus drei Staffeln mit paarweisen Bänden[12]. Was ist darin enthalten?

Im Band 1 geht es um die Grundlagen der Methode und um (Rang-)Gruppenverarbeitungen, sowie um Link-Listen. Er beginnt mit einem sehr einfachen Beispiel und steigert sich bis zum automatischen E-Mail-Versand der Ergebnisse des dort entworfenen fiktiven Internationalen Schulsportwettbewerbs. Dabei kommen nahezu ausschließlich „traditionelle" Aspekte der Methode zur Anwendung, die allerdings zum Teil auf ungewohnte Weise dazu dienen, funktionstüchtige Programme sowohl algorithmisch als auch objektorientiert zu entwerfen. Der Übergang zwischen diesen beiden Formen wird ausführlich erklärt und demonstriert. Auch das Thema „Test" spielt darin eine große Rolle, und zwar vor *und* nach der Implementierung. Ergänzend wird eine Java-Standard-Zugriffsschicht für sequentielle Dateien entworfen.

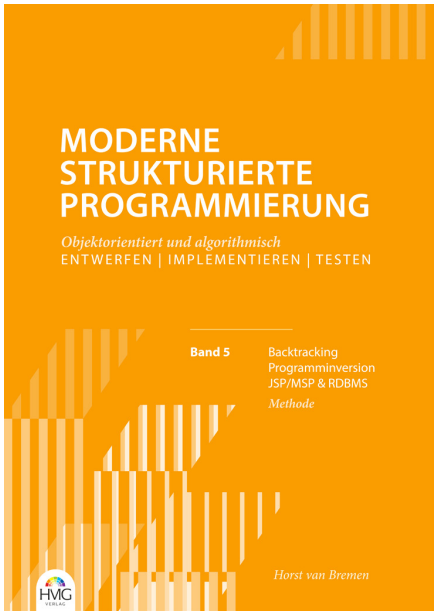
Nur im Paperback-Format wurden die Bände 1 und 2 in die Schwierigkeitsniveaus A1 und A2 geteilt. Wer sich noch nie zuvor mit der Strukturierten Programmierung befasst hat, sollte sich mit dem Niveau A2, also den Ranggruppen- und Link-Listen-Verarbeitungen, Zeit lassen. Keine Bange! Es wird alles detailliert erklärt. Auf B1 und B2 (in den Bänden 3 und 4) folgen C1 und C2 (in den Bänden 5 und 6). Die Analogie zu den Sprachniveaus wurde gewählt, um zu verdeutlichen, dass der Weg der Buchreihe vom elementaren Einstieg für Anfänger bis zu einem hohen Maß an Beherrschung von Methode und Praxis für Profis führt.

Der Band 2 enthält alle Beispielprogramme, mit denen die Entwürfe im Band 1 realisiert wurden. Das erste und einfachste Beispiel wurde in COBOL II, C und Java implementiert, während die anderen Beispiele meist in C *und* in Java – als prototypische Exponenten der algorithmischen und der objektorientierten Welt – programmiert wurden. Nur das letzte Beispiel, der E-Mail-Versand, ist allein in Java geschrieben, weil ein C-Programm sehr umständlich wäre und keinen Erkenntnisgewinn böte.

Der Band 3 behandelt das Mischen in vielerlei Konstellationen. Oft stehen die zu mischenden Daten keineswegs passend sortiert zur Verfügung. Dann ist deutlich mehr Kreativität als im Fall synchron sortierter Daten erforderlich, der gleichwohl ebenso detailliert erklärt wird. Mit dem Mischen von relational organisierten Daten einschliesslich Restartfähigkeit wird der Schwierigkeitsgrad B1 erreicht. Er steigt auf B2 mit dem Abgleich von Bilddateien, die einerseits auf Memory Chips und andererseits auf Festplatten oder ähnlichem gespeichert und in Ordnern organisiert sind. Schließlich geht es darum, im Rahmen des Copy Management ganze Ordnerhierarchien gegeneinander abzugleichen, um Differenzen zu erkennen und sie für die zielgerichtete Backup-Aktualisierung zu verwenden. Die strukturierte Hilfsmethode der Programminversion wird erstmals verwendet, aber noch nicht detailliert untersucht.

Der Band 4 zeigt wiederum die zu den Entwürfen gehörenden Implementierungen. Die Programme im Band 4 sind in C und in Java geschrieben, wobei die Java-Programme das relationale Datenbanksystem MySQL nutzen, wo nötig. Auf der Website des Verlags wird die Verfügbarkeit der Hardcover- und Paperback-Bände rechtzeitig angekündigt. Das gilt ebenso für die Bände 5 und 6.

Nun verbleiben noch drei Themenkomplexe, nämlich die bereits erwähnten Hilfsmethoden einerseits, und die spezielle Beziehung der Strukturierten Programmierung zur Datenbankprogrammierung andererseits.



Der Band 5-C1 geht die Themen Backtracking und Programminversion systematisch an, bevor im Band 5-C2 in Top Down-Manier der Bezug zwischen der Systemanalyse und der relationalen Datenbankprogrammierung aus JSP/MSP-Sicht aufgebaut wird. Der Hardcover-Band 5 vereint C1 und C2. Anhand einer MySQL-Implementierung der Datenbestände wird der Entwurf von Programmen für den Internationalen Schulsportwettbewerb im relationalen Umfeld demonstriert, final sogar anhand einer restartfähigen Verarbeitung. Dieser Band ist definitiv nicht für Anfänger geeignet, wird aber für Fortgeschrittene und Profis – hoffentlich – von großem Nutzen sein.

Der Band 6(-C1/C2) versammelt alle Beispielprogramme, die aus den Entwürfen im Band 5(-C1/C2) entstanden. Sie sind in C *oder* in Java – also nicht in beiden Sprachen – geschrieben, wobei die Datenbankprogramme wie im Band 4-B1 in Java implementiert wurden. Der Grund hierfür ist, dass das C-API (Application Programming Interface) von MySQL nicht sonderlich attraktiv ist und deutlich mehr – langweiligen – Code als das Java-API erfordert. Zudem ist der Konfigurierungsaufwand für die Konnektivität beim C-API erheblich.

1.6 In eigener Sache

Ich möchte nun noch einige – mir wichtige – Anmerkungen machen:

- Alles Schriftliche dient der Kommunikation. Im Fall eines Buches ist es die Kommunikation mit seinen Leser(inne)n, die kein Autor allesamt kennen und deren Vorlieben oder Abneigungen er daher nicht berücksichtigen kann. Diese Kommunikation findet im Allgemeinen indirekt mittels des Textes und der Grafiken statt, die für sich selber sprechen sollen. In etlichen Fällen erschien es mir aber als angebracht, Sie als Leserin oder Leser direkt anzusprechen. Das ist nicht als Kumpelhaftigkeit gemeint, sondern wird von

mir unter anderem dann verwendet, wo es eher um Wahrnehmungen und Meinungen, als um Tatsachen geht (wobei es dazwischen eine beachtliche Grauzone gibt).

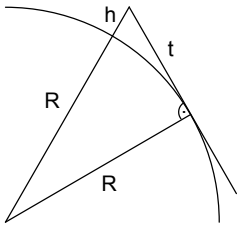
- Wie das Wort „Autor" im vorangehenden Abschnitt zeigt, verwende ich oft nur die männliche Form. Fehler werden von Programmierern gemacht, nicht von Programmiererinnen, die natürlich ebenso wenig unfehlbar sind. Die Autorin Elke Heidenreich soll einen Rundfunkbeitrag einmal folgendermaßen begonnen haben: „Liebe Zuhörer und Zuhörerinnen an den Radioapparaten und Radioapparatinnen", was ihr angeblich einen Shitstorm eintrug. Diese witzige Idee lässt sich leider nicht anhand von Fundstellen im Web belegen, ist ihr aber zuzutrauen. Sie karikierte damit vermutlich die häufig total verkrampften Bemühungen um eine geschlechtsneutrale oder zumindest nicht männlich dominierte Schreibung, was keineswegs gut ankam. Vor dem Hintergrund der weiterhin andauernden Diskussionen um dieses Thema, also um das korrekte Gendern, sind sprachverändernde Bemühungen meiner Ansicht nach zum Scheitern verdammt. Um es Ihnen zu ersparen, sich mit angeblich politisch korrekten, aber unlesbaren Bezeichnungen gequält zu fühlen, habe ich es mir ziemlich einfach gemacht. Bei allen diesbezüglich empfindlichen Seelen möchte ich mich dafür vorab entschuldigen.
- Zudem lässt sich meine kulturelle Prägung als Deutscher nicht verbergen. Warum auch? Jede(r) kann, darf und sollte auf ihre oder seine Kultur stolz sein und sie leben. Diese Prägung manifestiert sich – vielfach unabsichtlich – in Stil und Inhalt. Ebenso wenig wie im vorangehenden Abschnitt ist das als Ausschließung Anderer gemeint. Ganz im Gegenteil habe ich mehrfach internationale Aspekte einbezogen, was ich als erhebliche Bereicherung empfinde. Die Grafiken sind – bis auf eine französische Genealogie – durchgehend in englischer Sprache beschriftet, damit sie nicht bei jeder Übersetzung bearbeitet werden müssen. Das Englische ist die Lingua Franca der IT, und Michael A. Jackson ist Brite.
- Trotz extremer Sorgfalt und mehrfachen, intensiven Überprüfungen des Texts, der Grafiken und der Programme kann ein solches Monumentalwerk nicht fehlerfrei sein. Es ist möglich, dass Sie auf Dinge stoßen, die wirklich falsch und nicht bloß unverständlich sind, was auch schon schlimm genug wäre. Für Hinweise darauf habe ich immer ein offenes Ohr. Erkannte Fehler werden korrigiert und unzulängliche Beschreibungen werden verbessert, sobald ich davon weiß.
- Man kann es nicht allen recht machen. Für manche Anhänger der OO-Glaubensrichtungen (pardon!) grenzt es möglicherweise an ein Sakrileg, dass ausgerechnet auf der Basis einer längst im Treibsand der IT-Geschichte untergegangenen Methode der Versuch unternommen wird, sowohl eine Brücke zwischen der algorithmischen und der OO-Programmierung zu bauen als auch Java-Programme direkt aus JSP-Modellen herzuleiten. Programmierer(innen), die ihre Arbeit zu verlieren drohen, weil die von ihnen jahrzehntelang gewarteten Altsysteme durch neue OO-Software abgelöst werden, können diese Brücke dagegen hoffentlich als einen gangbaren Weg in eine bessere berufliche Zukunft begrüßen.
- Zwangsläufig kommen im Fließtext und in den Grafiken viele englische Worte vor. In letzteren und in den Programmen dominiert die radikale Kleinschreibung, aber im Fließtext kollidiert diese mit der deutschen Groß- und Kleinschreibung. Daher habe ich mich mit dem Englisch-Korrektor und Übersetzer David Roseveare für den Fließtext auf folgende Regeln geeinigt: a) Die englischen Namen von Strukturblöcken werden in Anführungszeichen zitiert. Abkürzungen darin werden gegebenenfalls mit Klammersetzung ausgeschrieben. b) Andere englische Worte werden wie deutsche Worte groß oder klein geschrieben. c) Englische Worte werden nicht „genitiviert" oder anderweitig an deutsche Grammatikregeln angepasst. d) Sie werden untereinander auch nicht mit Bindestrichen verbunden, ausser bei bekannten Ausnahmen wie „object-oriented". Wenn englischen Worten ein deutsches Wort folgt, das nach deutschen Regeln mit einem Bindestrich angeschlossen werden muss, dann geschieht das auch (siehe obiges Beispiel: Top Down-Manier).
- Entspannen Sie sich. Nehmen Sie alles, was Sie hier lesen, mit Humor.

1.7 Last but not least

… bin ich es Ihnen noch schuldig, die kryptische Bemerkung über die Wahrnehmung und die Wissenschaft hinsichtlich des eingangs gezeigten Fotos aufzulösen. Diese Aufnahme ist in mehrfacher Hinsicht aussergewöhnlich:

- Die darauf erkennbaren Berge sind fast immer unsichtbar, weil ein Dunstschleier über dem Golf von Genua sie verbirgt. Nur dadurch, dass kalte Luft kaum Wasserdampf aufnehmen kann, sind solche optischen Überreichweiten ausnahmsweise möglich. Zehn Tage vorher war von Porto Maurizio (Imperia) aus tagsüber auch die ungefähr 140 Kilometer entfernte nördliche Spitze von Korsika, das Cap Corse, sichtbar, wobei sogar Einzelheiten zu erkennen waren.
- Das Original dieser Aufnahme, die mit einem starken Teleobjektiv gemacht wurde, ist nicht ganz so beeindruckend, weil die untere, dunklere Partie darauf nur wenige Details zu erkennen erlaubt. Die von der Kamera gewählte Belichtung hebt zwar sehr schön das – schon ins Orange tendierende – Morgenrot hervor, aber die Tonwertspreizung im Original war zu extrem, um auch den Stadtteil im Vordergrund gut erkennen zu können. Daher wurde in Photoshop mittels des magnetischen Lassos der dunkle Bereich ausgewählt und mit der Funktion „Auto-Kontrast" aufgehellt. Erstaunlich viele Details, die zuvor im Dunkeln versunken waren, traten dadurch ans Licht. Auch dies ist ein schönes Beispiel für die digitalen Techniken, die uns erst seit nicht allzu langer Zeit zur Verfügung stehen.
- Wohl jede(r), die oder der sich einmal am Meer gefragt hat, wie weit denn der Horizont entfernt sei, wird rätseln, „wieviel Berg" es auf diesem Bild eigentlich zu sehen gibt. Es sind ja circa 180 Kilometer vom Aufnahmestandort zu den dort gezeigten Apuanischen Alpen, die erst durch Koordinatenberechnungen identifiziert werden konnten. Deren höchste Erhebung, der Monte Pisanino (1946 m) befindet sich fast ganz links in der Bergkette. Ganz rechts scheint eine Landspitze zu sein, aber das täuscht.

Die geografischen und geometrischen Aspekte sollen nun näher betrachtet werden. Zunächst geht es um die Entfernung zum Horizont. Da 180 Kilometer im Verhältnis zum Erdumfang von circa 40.000 Kilometer viel zu wenig ist, um damit eine brauchbare Grafik zu zeichnen, soll zunächst ein weit kleinerer, kugelförmiger Himmelskörper ohne Atmosphäre betrachtet werden, wobei der Kreisabschnitt zu einem Großkreis um dessen Kern gehört:



$$R^2 + t^2 = (R + h)^2$$
$$= R^2 + 2Rh + h^2$$
$$\Leftrightarrow \quad t^2 = 2Rh + h^2$$
$$\Leftrightarrow \quad t = \sqrt{2Rh + h^2}$$
$$\Leftrightarrow \quad t \approx \sqrt{2Rh}$$

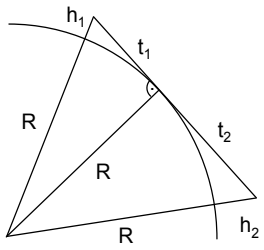
Anmerkung: h^2 kann vernachlässigt werden, wenn $h << R$ gilt, was auf der Erde der Fall ist.

Wenn eine Person von h = 1,60 m Augenhöhe am Rand eines Meeres oder Sees steht, ergibt sich ihre Sichtweite t anhand des mittleren Erdradius R von 6.371 Kilometer zu 4,5 km, wobei der Einfluss der Atmosphäre nicht berücksichtigt ist, die bekanntlich oberflächennahe Strahlen beugt.

Aus der Höhe des Aufnahmestandorts des eingangs gezeigten Fotos, nämlich h = 160 m, errechnet sich somit eine Sichtweite von 45 km bis zu dem Punkt, wo die Tangente t die Wasseroberfläche streift. Diese Höhe ist zwar 100 * 1,6 m, aber wegen der Wurzelfunktion vergrößert sich die Sichtweite nur um das Zehnfache.

Abbildung 1.1: *Ableitung der Tangentenlänge t aus dem Kugelradius R und der Betrachterhöhe h*

Da die Aufnahme auch die Berge jenseits der Sichtweite t zeigt, muss die Grafik erweitert werden:



Weil t_1 = 45 km ist, muss t_2 = 135 km sein, denn $t_1 + t_2$ = 180 km. Um die Sichtweite von 45 km zu verdreifachen, muss h_2 neun mal so groß wie h_1 (160 m) sein, also 1.440 m. Auf einem Himmelskörper ohne Atmosphäre wären also vom Monte Pisanino nur 506 m zu sehen, und das auf eine so große Entfernung!

Abbildung 1.2: *Geometrische Konstellation bei Überreichweiten*

Nun kommen uns zwei Effekte zur Hilfe, der eine wirklich und der andere scheinbar. Der erste ist die terrestrische Refraktion[13]. Darunter versteht man den Effekt, der oberflächennah verlaufende Lichtstrahlen in der Atmosphäre zum Betrachter hin beugt. Genauer gesagt, handelt es sich um den Krümmungsradius der Lichtstrahlen im Verhältnis zur Erdkrümmung, der durchschnittlich circa 13% beträgt. Die Refraktion ist der Grund, warum Sonne und Mond beim Auf- oder Untergang überhaupt zu sehen sind, obwohl sie sich in dieser Zeit, geometrisch gesehen, hinter dem Horizont befinden.

Zitat aus dem genannten Wikipedia-Artikel: »Die Refraktion variiert sehr stark, sie hängt von der aktuellen Dichteschichtung der Atmosphäre ab, genauer vom Gradienten der Feuchtigkeit, der Temperatur und des Druckes der Luft.«

Da die exakten Werte dieser drei Parameter zum Zeitpunkt der Aufnahme nicht bekannt sind, können 13% als Annäherung dienen. Auf der Gesamtstrecke von 180 km beträgt die Krümmung der Erdoberfläche 160 m + 1.440 m = 1.600 m. 13% davon entsprechen 208 m. Die Apuanischen Alpen waren folglich unter dieser Annahme erst ab circa 1.230 m Höhe sichtbar. Alles darunter, also die niedrigeren Berge, sowie die Hügel und das Flachland, bleibt unsichtbar. Das erklärt den Landspitzen-Effekt rechts im Bild.

Circa 710 Meter Rest-Berghöhe des Monte Pisanino sind immer noch nicht sonderlich beeindruckend. Hier kommt der zweite Effekt ins Spiel. Er lässt Sonne und Mond in der Nähe des Horizonts für den Betrachter deutlich größer als am Zenit erscheinen, was am Aufnahmetag auch für die Apuanischen Alpen galt: Die weit entfernte Bergkette wirkte beeindruckend nahe. Hierzu gibt es unter anderem die folgende, ziemlich ernüchternde Fundstelle bei Wikipedia:

> »Die **Mondtäuschung** ist eine optische Täuschung, durch welche der Mond und die Sonne in Horizontnähe größer erscheinen als bei höherem Stand am Firmament. Für diesen insbesondere beim Aufgang des Vollmondes bekannten Effekt gibt es keine physikalische oder astronomische Erklärung. Die Ursache dieses wahrnehmungspsychologischen Phänomens ist nicht endgültig geklärt.«[14]

Die Kamera ließ sich nicht täuschen und machte daher ein Bild, das die tatsächlichen Größenverhältnisse zeigt.

Endnoten

1 **Michael Anthony Jackson** (* 1936) ist ein britischer Informatiker und arbeitet als unabhängiger Berater in London, England und bei AT&T Research, Florham Park, NJ, USA. Er ist Gastprofessor an der Open University in Großbritannien.

Jackson studierte an der Oxford University und war Kommilitone von Tony Hoare. In den 1970er Jahren entwickelte er die Softwareentwicklungsmethode Jackson Structured Programming (JSP), in den 1980er Jahren entwickelte er zusammen mit John Cameron eine Methode zur Systementwicklung (Jackson System Development (JSD)). In den 1990er Jahren entwickelte er die Problem Frames, einen Ansatz um Probleme zu zerlegen und zu strukturieren.

[Seitentitel: Michael Anthony Jackson
Herausgeber: Wikipedia – Die freie Enzyklopädie.
Autor(en): Wikipedia-Autoren, siehe Versionsgeschichte
Datum der letzten Bearbeitung: 23. Oktober 2019, 11:38 UTC
Versions-ID der Seite: 193387996
Permanentlink: https://de.wikipedia.org/w/index.php?title=Michael_Anthony_Jackson&oldid=193387996
Datum des Abrufs: 13. Juni 2022, 15:30 UTC]

2 **Edward Nash Yourdon** (April 30, 1944 – January 20, 2016) was an American software engineer, computer consultant, author and lecturer, and software engineering methodology pioneer. He was one of the lead developers of the structured analysis techniques of the 1970s and a co-developer of both the Yourdon/Whitehead method for object-oriented analysis/design in the late 1980s and the Coad/Yourdon methodology for object-oriented analysis/design in the 1990s.

[Page name: Edward Yourdon
Author: Wikipedia contributors
Publisher: Wikipedia, The Free Encyclopedia.
Date of last revision: 29 July 2022 14:41 UTC
Date retrieved: 16 August 2022 15:23 UTC
Permanent link: https://en.wikipedia.org/w/index.php?title=Edward_Yourdon&oldid=1101143734
Primary contributors: revision history statistics
Page Version ID: 1101143734]

3 Textziffern in eckigen Klammern verweisen auf das Kapitel „18.1 Literaturverzeichnis".

4 Die **Structured Analysis and Design Technique** (**SADT**) ist eine Methode zur Softwareentwicklung, die 1977 von Douglas T. Ross publiziert wurde.

[…]

Basiselement der Modellierung ist die Funktion, dargestellt als Rechteck, die Eingangsgrößen (engl. Input, Pfeile von links) in die Ausgangsgrößen (engl. Output, Pfeile nach rechts) transformiert. Für die Ausführung werden Mechanismen (engl. Mechanisms, Pfeile von unten) benötigt, zum Beispiel Ressourcen. Weitere Einflüsse oder Störgrößen (Control) werden als Pfeile von oben dargestellt.

Mehrere Funktionen können hintereinander ausgeführt werden, in der eine Ausgangsgröße der einen Funktion mit der Eingangsgröße der anderen verbunden wird.

Die schrittweise Modellierung kann top-down erfolgen. Eine Funktion wird hierbei hierarchisch zerlegt, d. h. eine Funktion einer Ebene, wird in der nächsten Ebene als Verschaltung mehrerer Teilfunktion dargestellt.

[Wikipedia -Seitentitel: Structured Analysis and Design Technique
Datum der letzten Bearbeitung: 15. Juli 2019, 15:46 UTC
Versions-ID der Seite: 190456980
Permanentlink: https://de.wikipedia.org/w/index.php?title=Structured_Analysis_and_Design_Technique&oldid=190456980
Datum des Abrufs: 13. Juni 2022, 15:20 UTC]

[Hervorhebung des Autors: Die Quellen von Wikipedia-Zitaten werden ab hier ohne Herausgeber / Autor(en) aufgelistet, weil diese Angaben konstant sind. Dem Wort „Seitentitel" wird daher „Wikipedia-" vorangestellt.]

5 **Edsger Wybe Dijkstra** (* 11. Mai 1930 in Rotterdam; † 6. August 2002 in Nuenen) war ein niederländischer Informatiker. Er war der Wegbereiter der strukturierten Programmierung. 1972 erhielt er den Turing Award für grundlegende Beiträge zur Entwicklung von Programmiersprachen.

[Wikipedia-Seitentitel: Edsger W. Dijkstra
Datum der letzten Bearbeitung: 29. April 2022, 07:04 UTC
Versions-ID der Seite: 222457618
Permanentlink: https://de.wikipedia.org/w/index.php?title=Edsger_W._Dijkstra&oldid=222457618
Datum des Abrufs: 13. Juni 2022, 15:37 UTC]

6 Die **GMD – Forschungszentrum Informationstechnik GmbH** war eine zwischen 1968 und 2001 bestehende deutsche Großforschungseinrichtung für angewandte Mathematik und Informatik.

[…]

Die Gründung erfolgte am 23. April 1968 in Bonn unter dem Namen Gesellschaft für Mathematik und Datenverarbeitung (GMD). Damit sollte das Konzept der Großforschung, das sich im Bereich der Kernenergie bewährt hatte, auf die damals noch Datenverarbeitung genannte Informatik übertragen werden. Hierzu wurde das Institut für Instrumentelle Mathematik (IIM) an der mathematischen Fakultät der Universität Bonn ausgebaut als Großforschungseinrichtung des Bundes mit einer Minderheitsbeteiligung des Landes Nordrhein-Westfalen.

[…]

Im März 1995 erfolgte eine Umbenennung. Auf Initiative des Bundesministeriums für Bildung und Forschung wurde die GMD in den Jahren 2000 bis 2001 gegen den Widerstand der Mitarbeiter, die eine breite Unterstützung durch die regionale Politik erfuhren, mit der Fraunhofer-Gesellschaft fusioniert. […] Die Einrichtung ist im Fraunhofer-Institutszentrum Schloss Birlinghoven aufgegangen.

[Wikipedia-Seitentitel: GMD-Forschungszentrum Informationstechnik
Datum der letzten Bearbeitung: 12. Mai 2021, 10:49 UTC
Versions-ID der Seite: 211872283
Permanentlink: https://de.wikipedia.org/w/index.php?title=GMD-Forschungszentrum_Informationstechnik&oldid=211872283
Datum des Abrufs: 13. Juni 2022, 15:43 UTC]

7 Damals noch unter meinem Geburtsnamen Gomm

8 **TR 440** (gesprochen: T-R-4-40) ist die Bezeichnung des von AEG-Telefunken, Fachbereich Informationstechnik, aus dem „Telefunken-Rechner TR 4" weiterentwickelten Großrechners. AEG-Telefunken lieferte 1969 den ersten TR 440 an das Deutsche Rechenzentrum. Als der TR 440 herauskam, war er der schnellste Rechner, der je in Europa entwickelt worden war. Bis im Jahr 1974 wurden insgesamt 46 Anlagen vom Typ TR 440 gebaut.

Das Gesamtsystem aus Hardware, BS 3 und Programmiersystem wurde auch unter dem Namen TNS 440 (Teilnehmer-System 440) vermarktet.

[…]

Die möglichen Nachfolgesysteme waren weit weniger benutzerfreundlich als das TNS 440; insbesondere gegen die Beschaffung von 7.700-Rechnern mit BS2000® gab es große Widerstände trotz der von Siemens angebotenen Umstellungshilfen. Die Wertschätzung für das TNS 440 bei den Nutzern ging wesentlich auf dessen Systemsoftware zurück: Das Betriebssystem bietet eine Virtuelle Speicherverwaltung mit Speicherschutz und Mehrfachzugriff, das Programmiersystem eine flexible, übersichtliche Kommandosprache, eine gute Ausstattung an Programmiersprachen (einschließlich Sprachverknüpfung) und Programmbibliotheken, sowie innovative Testhilfen für die Programmentwicklung.

[Wikipedia-Seitentitel: TR 440

Datum der letzten Bearbeitung: 20. April 2022, 08:39 UTC
Versions-ID der Seite: 222219658
Permanentlink: https://de.wikipedia.org/w/index.php?title=TR_440&oldid=222219658
Datum des Abrufs: 13. Juni 2022, 15:51 UTC]
[Hervorhebung d. A.: Das Deutsche Rechenzentrum in Darmstadt ging zum 01.01.1973 in der GMD auf.]

9 **System/360** oder kurz **S/360** bezeichnet eine Großrechnerarchitektur der Firma IBM aus dem Jahre 1964. Zuvor wurden von IBM die 700/7000 series gebaut. Dem System/360 folgte das im Jahr 1970 angekündigte System/370.

[…]

Es war eines der erfolgreichsten Computersysteme aller Zeiten. Die von IBM veröffentlichten Spezifikationen erlaubten es außerdem auch anderen Anbietern, Peripheriegeräte für das System/360 anzubieten, häufig kostengünstiger. Umgekehrt begannen andere Firmen wie Amdahl und Univac zum 360-System kompatible Computer zu entwickeln. Ebenso wie die Hardware wurde mit dem System/360 auch die Software vereinheitlicht und kompatibel gemacht.

Die Hauptarchitekten waren Frederick P. Brooks, Gerrit Blaauw, Gene Amdahl. Die Gesamtleitung hatte Bob O. Evans. Beteiligt war auch u. a. Erich Bloch. Brooks, Bloch und Evans erhielten dafür 1985 die National Medal of Technology and Innovation.

[Wikipedia-Seitentitel: System/360
Datum der letzten Bearbeitung: 2. Mai 2022, 16:31 UTC
Versions-ID der Seite: 222558995
Permanentlink: https://de.wikipedia.org/w/index.php?title=System/360&oldid=222558995
Datum des Abrufs: 13. Juni 2022, 15:57 UTC]

10 **Prokrustes** (altgriechisch Προκρούστης (..), deutsch ‚Ausstrecker') war ein Riese aus der griechischen Mythologie, Beiname des Polypemon oder Damastes, eines attischen Räubers in der Umgegend von Eleusis und Sohn des Poseidon.

[…]

In seiner Weltgeschichte berichtet der altgriechische Geschichtsschreiber Diodor (1. Jahrhundert v. Chr.) Folgendes über den Unhold und Wegelagerer Prokrustes:

Prokrustes bot Reisenden ein Bett an, aber in manchen Sagen zwang er auch Wanderer, sich auf ein Bett zu legen. Wenn sie zu groß für das Bett waren, hackte er ihnen die Füße bzw. überschüssige Gliedmaßen ab; waren sie zu klein, hämmerte und reckte er ihnen die Glieder auseinander, indem er sie auf einem Amboss streckte.

Prokrustes wurde von Theseus auf seiner Wanderung nach Athen als letzter der Bösewichte am Kephisos erschlagen.

[…]

Als Prokrustesbett oder Bett des Prokrustes bezeichnet man redensartlich eine Form oder ein Schema, wohinein etwas gezwungen wird, das dort eigentlich nicht hineinpasst.

[Wikipedia-Seitentitel: Prokrustes
Datum der letzten Bearbeitung: 4. Februar 2022, 12:37 UTC
Versions-ID der Seite: 219858623
Permanentlink: https://de.wikipedia.org/w/index.php?title=Prokrustes&oldid=219858623
Datum des Abrufs: 13. Juni 2022, 16:01 UTC]

11 **Grady Booch** (* 27. Februar 1955 in Texas) ist ein amerikanischer Informatiker. Er gilt als Pionier auf dem Gebiet des modularen und objektorientierten Softwareentwurfs und der Klassenbibliotheken (Ada, C++). 1977 schloss er als Bachelor of Science an der United States Air Force Academy ab, 1979 erlangte er den Titel Master of Science an der University of California.

Seit 1980 ist er Chief Scientist der Firma Rational Software in Santa Clara (Kalifornien). Zusammen mit Ivar Jacobson und James Rumbaugh spricht man auch von den Drei Amigos (Begründer der Unified Modeling

Language). Nach der Übernahme von Rational Software durch IBM arbeitet er bei IBM und ist seit 2003 IBM Fellow.

[Wikipedia-Seitentitel: Grady Booch
Datum der letzten Bearbeitung: 25. April 2016, 04:18 UTC
Versions-ID der Seite: 153784355
Permanentlink: https://de.wikipedia.org/w/index.php?title=Grady_Booch&oldid=153784355
Datum des Abrufs: 16. August 2022, 15:58 UTC]

12 Im Oktober 2024 wurden die Paperback-Bände 1 und 2 in die Schwierigkeitsniveaus A1 und A2 aufgeteilt. Die Hauptthemen der A1-Ausgaben sind „Grundlagen" und „Gruppen". In diesem Kontext wurde auch das Titel-Layout der Buchreihe verändert. Die A2-Ausgaben werden mit „Ranggruppen" und „Link-Listen" bezeichnet. Die Hardcover-Bände blieben inhaltlich unverändert. Es werden dort bei der geplanten Neuauflage lediglich alle vier Hauptthemen auf dem Titel aufgeführt und im Band 1 das Unterkapitel zu den Nassi-Shneiderman-Diagrammen ergänzt. Die zweite und die dritte Staffel werden analog zur ersten Staffel aufgebaut sein. Dort soll es ebenfalls jeweils zwei Hardcover-Bände und vier Paperback-Bände geben, die den Schwierigkeitsniveaus B1, B2, C1 und C2 entsprechen.

13 Als **terrestrische Refraktion** (auch Strahlenbrechung oder atmosphärische Refraktion genannt) wird die Brechung eines Lichtstrahls in der untersten Erdatmosphäre bezeichnet. Diese entsteht durch die Änderung des Brechungsindexes der Luft entlang des Strahlverlaufs infolge der mit der Höhe abnehmenden Luftdichte und bewirkt eine bogenförmige Krümmung des Strahls, die bei genaueren Vermessungen oder im physikalischen Labor als Korrektion („Reduktion") an jedem gemessenen Vertikalwinkel angebracht werden muss. Diese Strahlkrümmung beträgt durchschnittlich 13 % der Erdkrümmung und erhöht die horizontale Sichtweite geringfügig.

[Wikipedia-Seitentitel: Terrestrische Refraktion
Datum der letzten Bearbeitung: 2. Mai 2022, 17:22 UTC
Versions-ID der Seite: 222560036
Permanentlink: https://de.wikipedia.org/w/index.php?title=Terrestrische_Refraktion&oldid=222560036
Datum des Abrufs: 13. Juni 2022, 16:08 UTC]

14 Erste Hinweise auf das Phänomen der Mondtäuschung finden sich auf Tontafeln aus den königlichen Bibliotheken von Niniveh und Babylon (6. Jahrhundert v. Chr.). Ptolemäus (ca. 150 n. Chr.) vermutete fälschlicherweise vergrößernde Eigenschaften der Atmosphäre. Alhazen (Abu Ali al-Hasan ibn al-Haitham, 965 bis ca. 1040) stellte fest, dass der Mond sowohl am Horizont als auch im Zenit die gleiche Größe hat, und schrieb bereits vom abgeflachten Firmament […] als Ursache der Wahrnehmungstäuschung. Auch Leonardo da Vinci, Johannes Kepler und René Descartes beschäftigten sich mit der Mondtäuschung. Seit über 100 Jahren wird diese optische Täuschung von der wissenschaftlichen Wahrnehmungspsychologie untersucht. Dennoch ist das Phänomen noch immer nicht eindeutig geklärt, es bleiben Widersprüche bei den unterschiedlichen Erklärungsansätzen. Die derzeit anerkanntesten und von vielen Experimenten untermauerten Erklärungen sind die der falsch eingeschätzten Entfernung mit dem abgeflachten Firmament und das Prinzip der Vergleichsobjekte.

[Wikipedia-Seitentitel: Mondtäuschung
Datum der letzten Bearbeitung: 16. Mai 2022, 21:32 UTC
Versions-ID der Seite: 222937687
Permanentlink: https://de.wikipedia.org/w/index.php?title=Mondt%C3%A4uschung&oldid=222937687
Datum des Abrufs: 13. Juni 2022, 16:14 UTC]