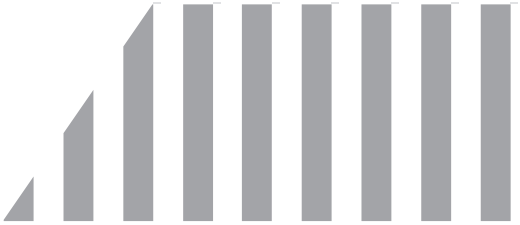
Band 1-A1

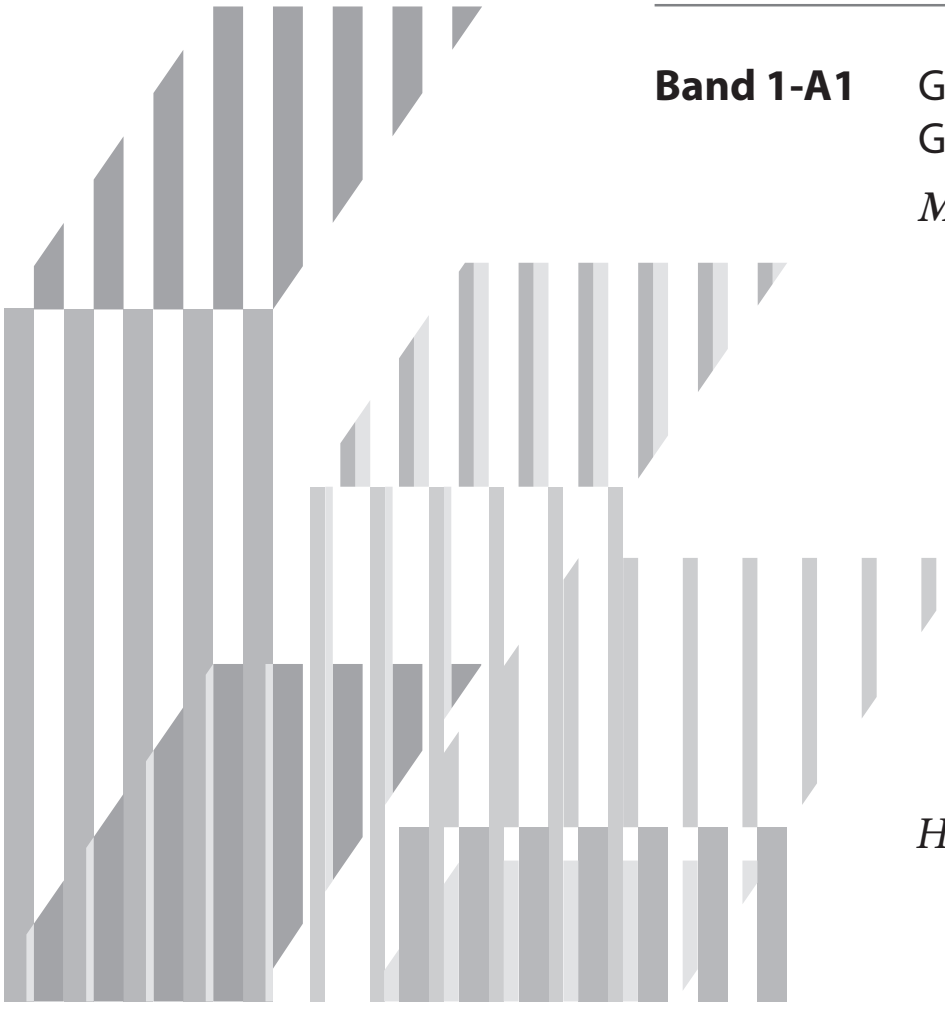


MODERNE STRUKTURIERTE PROGRAMMIERUNG

Objektorientiert und algorithmisch
ENTWERFEN | IMPLEMENTIEREN | TESTEN

2. Auflage

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	BoD · Books on Demand GmbH, Überseering 33, 22297 Hamburg, bod@bod.de
Druck	Libri Plureos GmbH, Friedensallee 273, 22763 Hamburg
Version / Datum	Version 2.1.1 vom 1.7.2026
Literaturverzeichnis	siehe Kapitel „9.1 Literaturverzeichnis“ auf Seite 191 f
Rechtliches	siehe Kapitel „9.2 Rechtliche Hinweise“ auf Seite 193 ff
ISBN des Ringbuchs	978-3-6963-6369-7

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 ersten 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

1	Einführung	1
2	JSP-Basismethode	19
3	Algorithmische Implementierungsvorbereitungen	53
4	Objektorientierte Implementierungsvorbereitungen	83
5	Blick zurück und voraus	97
6	Durchlaufanalyse	115
7	Gruppenverarbeitung I	127
8	Gruppenverarbeitung II	177
9	Anhang	191

Inhaltsverzeichnis Band 1-A1

1 Einführung 1

1.1 JSP-Literatur 3
1.2 Wo stehen wir heute? 4
1.3 Programme mit MSP konstruieren 5
1.4 ... und die Folgen 7
1.5 Die Buchreihe 8
1.6 In eigener Sache 10
1.7 Last but not least 12

2 JSP-Basismethode 19

2.1 Vorbemerkungen 19
2.2 Das erste JSP-Beispiel 20
2.3 Der intuitive Ansatz 22
2.4 Konstruieren anstatt Erfinden 23
2.5 Strukturen der Ein- und Ausgabedaten identifizieren 24
2.5.1 Vom konkreten Eingabebeispiel zur abstrakten Eingabestruktur 24
2.5.2 I = Iteration 25
2.5.3 S = Selection (Auswahl) 26
2.5.4 Blöcke ohne S oder I 26
2.5.5 Heikle Ratschläge 26
2.5.6 Sonderfall Leerzeile 28
2.5.7 Unbemerkte Implikationen 28
2.5.8 Vom konkreten Ausgabebeispiel zur abstrakten Ausgabestruktur 29
2.5.9 Inkompatible Strukturblöcke 31
2.6 Korrespondenzen zwischen den Datenstrukturen ermitteln 31
2.7 Programmstruktur ohne Operationen ableiten 32
2.8 Programmstruktur mit Operationen ergänzen 34
2.9 Nachbemerkungen zu den Operationen 43
2.10 Kontrollen überprüfen und präzisieren 44
2.11 Ausgabe-Lösungsalternative 46

3 Algorithmische Implementierungsvorbereitungen 53

3.1 Flussdiagramm erstellen 54
3.2 Schematische Logik oder Pseudocode schreiben 57
3.3 Jacksons Schematische Logik 58
3.4 BDL-Pseudocode 63
3.4.1 Exkurs zum Hintergrund von BDL 63
3.4.2 Die BDL-Syntax 69
3.5 Anmerkungen zum Pseudocode 74
3.6 Nassi-Shneiderman-Diagramme zeichnen 75

4 Objektorientierte Implementierungsvorbereitungen 83

4.1 JSP objektorientiert? 83

4.2 Klassen und Objekte bestimmen 85
4.3 Bewertung der Ergebnisse 90
4.4 Namenskonventionen 90

5 Blick zurück und voraus 97

5.1 Und das soll alles sein? 97
5.2 Wie geht es weiter? 99
5.2.1 Durchlaufanalyse als Sicherheitsnetz 100
5.2.2 Der Klassiker: (Rang-)Gruppenverarbeitung 101
5.2.3 Fallstudie zur Ranggruppenverarbeitung 101
5.3 Mischen: Das unerschöpfliche Thema 103
5.3.1 Misch-Standardfälle 103
5.3.2 Mischen von drei und mehr Beständen 104
5.3.3 Multidimensionales Mischen 104
5.3.4 Abgleich von Bilddateien mit ihren Kopien 104
5.3.5 Exkurs: Generierung von Change-Kommandos 105
5.3.6 Abgleich von Dateien in beliebig tief gestaffelten Verzeichnissen 105
5.3.7 Ende und neuer Anfang 106
5.4 Backtracking und Programminversion 107
5.4.1 Backtracking in drei Lektionen 108
5.4.2 Programminversion 108
5.4.3 Mischen mit Plausibilitätsprüfung 109
5.5 Zusammenfassung 110
5.6 JSP/MSP und relationale Datenbanktechniken 110
5.7 Geschafft! 112

6 Durchlaufanalyse 115

6.1 Durchlaufanalyse = Ablaufsimulation 115
6.2 Zusammenfassung 123
6.3 Problembehandlung 123
6.4 Testkontrolle 124

7 Gruppenverarbeitung I 127

7.1 Sonett-Typ-Erkennung 128
7.2 Sonett-Eingabestruktogramme 128
7.3 Generalisierung kontra Spezialisierung 132
7.4 Überlegungen zu den Kontrollen 136
7.5 Finale Struktogramme 139
7.6 Operationen 143
7.7 Italienisches Sonett: Beispiel und Struktur 145
7.8 Durchlaufanalyse 146
7.8.1 Sonderfall: Nach sechs Absätzen folgen weitere Zeilen 153
7.8.2 Sonderfall: Dateiende nach dem ersten Absatz 160
7.8.3 Sonderfall: Leerzeile(n) + Dateiende nach dem ersten Absatz 160
7.8.4 Vorsicht Falle! 163
7.8.5 Schlussbemerkungen zur Durchlaufanalyse 165
7.9 Pseudocode 166
7.10 Implementierungs-Varianten in COBOL II und REXX 168

7.10.1 COBOL II 168
7.10.2 REXX 170
7.11 Implementierung in C und Java 174

8 Gruppenverarbeitung II 177

8.1 Struktogramme von Ein- und Ausgabe 179
8.2 Programmstruktur, Operationen und Kontrollen 180
8.3 Programmstruktur mit Operationen 181
8.4 Durchlaufanalyse 182
8.4.1 Initialphase 182
8.4.2 Folgesatzverarbeitungen 183
8.4.3 Verarbeitung der ersten Duplikate 185
8.4.4 Verarbeitung des Gruppenendes und des Folgesatzes 187
8.4.5 Schlussphase 188
8.5 Implementierung in C und Java 188
8.6 Der geplatzte Traum 189

9 Anhang 191

9.1 Literaturverzeichnis 191
9.2 Rechtliche Hinweise 193
9.2.1 Geschützte Bezeichnungen 193
9.2.2 Haftungsausschluss 204
9.2.3 Zitate 204
9.2.4 Autoren im Literaturverzeichnis 205
9.2.5 Copyrights 205
9.2.6 Bild- und Grafiknachweis 206
9.3 Versionsgeschichte zu 1.1.3 206
9.4 Versionsgeschichte zu 2.1.1 206