Kapitel 2

Grundlagen

Die theoretischen Grundlagen der Diplomarbeit werden in diesem Kapitel beschrieben. Im Kapitel 2.1 folgt neben einem kurzen geschichtlichem Prolog, eine Einführung in die Software-Messung und Software-Komplexität. Kapitel 2.2 widmet sich dem Komplexitätsmaß von McCabe, der zyklomatischen Zahl. Die zweite in der Arbeit verwendete Komplexitätsmessung, die Programmier-Leistung von Halstead wird im Kapitel 2.3 behandelt. Eine Darstellung des SAP R/2 Systems und eine Erläuterung zur Programmiersprache ABAP/4 beinhaltet das Kapitel 2.4. Die eingesetzten statistischen Analyseverfahren werden im Kapitel 2.5 veranschaulicht.

2.1 Software-Messung und Software-Komplexität

Die Software-Messung ist ein Teilgebiet in der Software-Entwicklung, in dem seit mehr als dreißig Jahren geforscht wird. Grundlegende Publikationen wurden ab Mitte der sechziger und vorwiegend in den siebziger Jahren veröffentlicht. Aufbauend auf diese frühen Arbeiten wurden weitere Ergebnisse in den achtziger und neunziger Jahren erzielt.

Eine der ersten und bedeutenderen Arbeit, die Einfluß auf die Entwicklung der Software-Messung ausübte, ist die Arbeit von [Alexander64, Fenton91] im Jahre 1964. Alexander beschreibt die Merkmale eines Design-Prozesses in einer anwendungsunabhängigen Methode. Das Ergebnis dieser Abhandlung ist, daß ein gut entworfenes Gerät aus vielen voneinander isolierten Komponenten besteht. In seinem Buch geht Alexander nicht auf Software ein; seine Ideen wurden Jahre später in Forschungen zu System-Entwurfs-Messungen wieder aufgegriffen.

Das wahrscheinlich früheste Papier zu Software-Metriken stammt von [Rubey68, Zuse95] aus dem Jahre 1968. Rubey stellt einen Rahmen von Software-Attributen vor, die als Definition von Software-Qualität gedacht sind [Ebert96]. Von [vanEmden71, Zuse95] wurde 1971 eine Dissertation veröffentlicht, die auf dem Konzept von bedingten Wahrscheinlichkeiten auf dem Formalismus der Informationstheorie basiert. Sie scheint geeignet zu sein, die Komplexität von miteinander verbundenen Systemen zu modellieren. Dies ist vergleichbar mit der Komplexität von modularisierten Programmen. Die Arbeiten von Rubey und van Emden hatten jedoch keine nennenswerten Einflüsse auf weitere Forschungen.

Eine der ältesten Messung, die heute noch eingesetzt wird, ist LoC[1]. Sie wurde in zahlreichen Publikationen von diversen Autoren, z.B. von [Conte86] beschrieben. Angewendet wurde LoC, um z.B. die Produktivität von Entwicklern oder um Programm-Charakteristiken wie Zuverlässigkeit oder Wartbarkeit zu messen [Zuse95].

Als erstes Beispiel einer vollständigen Software-Messung geben [Fenton91] und [Zuse95] die Arbeit von [Halstead77] an. Dessen Hauptidee ist, daß das Verstehen von Software ein Prozeß von mentalen Manipulationen auf Programm-Schlüsselworten darstellt. Ausgehend von diesem Gedanken definiert Halstead eine Reihe von Messungen, z.B. Programm-Länge, Programm-Größe und Programmier-Leistung, die aus dem Programmtext extrahiert werden können. Viele Veröffentlichungen in den siebziger und frühen achtziger Jahren waren von Halsteads Arbeit beeinflußt, die als ein wichtiger Meilenstein in der Geschichte der Software-Messung gesehen wird. Seit Mitte der achtziger Jahre geriet Halsteads Messung in Verruf. Ein Grund ist u.a., daß die Messungen vorwiegend auf dem Programmtext basieren, d.h. stark von der verwendeten Programmiersprache abhängen. Eine weitere Ursache sieht [Coulter83] darin, daß heute einige kognitive psychologische Ansätze als falsch angesehen werden.

Parallel zu den Programmtext-Messungen, von z.B. Halstead, entwickelten sich seit Mitte der siebziger Jahre Software-Messungen, die auf Entscheidungsstrukturen im Programm basieren. Diese Messungen werden als Kontroll-Fluß-Messungen bezeichnet. Auslöser war eine steigende Akzeptanz der *Strukturierten Programmierung*[2]. Eine klassische Kontroll-Fluß-Messung ist die *zyklomatische Komplexität* von [McCabe76], die auf der Graphentheorie beruht. Seine Arbeit ist die meist zitierteste in der Literatur. McCabe postuliert, daß eine höhere zyklomatische Komplexität ein Programm repräsentiert, das schwieriger zu lesen, zu testen und zu verstehen ist. In den folgenden Jahren wurde McCabes Idee weiterentwickelt, um die zyklomatische Komplexität zu verbessern oder zu erweitern. Kritik äußerte u.a. [Shepperd88, Fenton91], der zeigt, daß viele publizierten Experimente statistisch zweifelhaft sind und daß McCabes Messung keine besseren Voraussagen trifft als LoC.

Die achtziger Jahre können als ein Jahrzehnt des System-Designs angesehen werden. Das Konzept von System-Design-Messungen beruht auf idealen Software-Systemen mit in relativer Isolation zueinander stehenden Komponenten[3]. Die bekannteste Design-Messung ist von [Henry81]. Das Verfahren basiert auf der Messung des

[1] Die Abkürzung LoC bedeutet Line-of-Code. Sie ist jede Zeile im Programmtext, die keine Kommentar- oder Leerzeile ist, unabhängig von der Anzahl der Anweisungen oder Anweisungsfragmente. Eine Programmzeile beinhaltet alle Zeilen im Programm, die Deklarationen, ausführbare und nicht-ausführbare Anweisungen umfassen [Conte86].

[2] Eine Reihe von Richtlinien und Techniken, um Software Programme zu schreiben. Das Konzept beruht auf der Verwendung einer beschränkten Anzahl von Befehlen, um Codeblöcke mit Einfach-Eingängen und Einfach-Ausgängen zu erhalten [Yourdon79].

[3] Komponenten sind Module, Prozeduren oder Unterroutinen

Informationsflußes zwischen System-Komponenten. Die Messungen definieren die Modul-Komplexität und die Modul-Kopplung.

Ab Ende der achtziger Jahre wurden Software-Messungen für objektorientierte Umgebungen, sogenannte OO-Messungen vorgeschlagen. Die ersten Untersuchungen stammen von [Rocacher88, Zuse95]. In den neunziger Jahren folgten neben grundlegenden Arbeiten, eine Vielzahl von Veröffentlichungen über Metriken zu objektorientierten Sprachen, wie Smalltalk oder C++.

2.1.1 Was ist eine Messung?

Eine *Messung* wird in [ISO14598] als ein Prozess definiert, bei dem einem *Objekt*[4] eine Zahl oder eine *Kategorie*[5] zugewiesen wird, um ein *Attribut*[6] des Objektes zu charakterisieren. Dazu wird eine spezielle Einheit und eine Zählregel verwendet.

[Fenton91] gibt eine ähnlich formale Definition[7] an. Es sollen bei einer Messung Informationen über Attribute eines Objektes gewonnen werden. Dabei ist zu beachten, daß weder Dinge noch Attribute isoliert gemessen werden, sondern Attribute von Dingen. Eine Messung ordnet Attributen von Objekten Zahlen oder Symbole zu, um sie zu beschreiben; sie sind als eine Abstraktion zu verstehen. Diese Zuordnung muß so festgelegt sein, daß Beziehungen zwischen den Zahlen oder Symbolen die Beziehungen zwischen den Attributen widerspiegeln [Sarle95].

Ausgehend von der formalen Definition einer Messung, ist ein *Maß* nach [ISO14598] und [Fenton91] eine objektive Zuweisung einer Zahl oder eines Symbols zu einem Objekt, um ein spezielles Attribut zu charakterisieren. Eine Zuweisung bezieht sich auf eine Messungsbeschreibung. Ein Maß ist nicht nur eine Zahl, sondern eine definierte *Beschreibung*[8] für ein Objekt und das zu messende Attribut.

Voraussetzung für eine Messung ist ein Konzept über Attribute und eine Anzahl zu messender Objekten, die diese Attribute besitzen. Nach der Identifizierung der Attribute und Objekte werden Relationen gesucht, die die Attribute beschreiben. $\mathcal{M}$ ist ein Maß für ein Attribut, wenn es eine Beschreibung von einem Empirischen-Relationen-System $\mathcal{C} = (C, R)$[9] in ein Numerisches-Relationen-System $\mathcal{N} = (N, P)$[10] gibt. Ein Beschreibung weist Objekten aus C, Zahlen aus N zu. Relationen aus R

[4] Objekte sind z.B. Personen, Gegenstände, Orte wie Räume, Ereignisse wie Reisen oder Testphasen eines Projektes.

[5] Der Begriff Kategorie wird benutzt, um auf qualitative Maße von Attributen hinzuweisen, z.B. wichtige Attribute eines Software Produktes.

[6] Ein Attribut ist eine meßbare physikalische oder abstrakte Eigenschaft eines Objektes, z.B. Größe einer Person, Kosten einer Reise, syntaktische Korrektheit eines Programms.

[7] Definition: Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.

[8] Eine Beschreibung $\mathcal{M}$ wird als Messen von Attributen bezeichnet.

[9] Ein Empirisches-Relationen-System $\mathcal{C} = (C, R)$ besteht aus Relationen R, z.B. Vergleiche oder Verkettungen, die die Attribute charakterisieren und aus einer Reihe von Objekten C, z.B. Programmtext.

[10] Ein Numerisches-Relationen-System $\mathcal{N} = (N, P)$ setzt sich aus Objekten N, z.B. Zahlen und Relationen P über N, z.B. Vergleiche oder Additionen, zusammen.

werden Relationen aus P zugewiesen.

Es bestehen viele verschiedene Ansichten über den Begriff der Messung. Diese führen zu unterschiedlichen Interpretationen, was eine Messung ist und was keine ist. Die Meinungen gehen bei verschiedenen Fragen auseinander, ob z.B. ein IQ-Testergebnis ein Maß für die Intelligenz von Menschen ist oder bei einem Raum mit blauen Wänden, Blau ein Maß für die Farbe des Raumes ist [Fenton91]. Weitere aufgezeigte Probleme sind u.a. der Umgang mit Fehlertoleranzen[11], die Behandlung verschiedener *Maßstäbe*[12] und die Anwendung von Transformationen, z.B. Vergleiche der Ergebnisse von Messungen wie „größer als“ oder „gleich“.

Ein Maßstab beschreibt die Messung eines Attributes und wird mittels *zulässiger Transformationen*[13] definiert. Der Maßstab ist eine spezielle Art und Weise der Zuweisung von Zahlen oder Symbolen. Die Klasse aller zulässigen Transformationen eines Empirisches-Relationen-Systems bestimmt den *Maßstabstyp*. Das Ziel einer Messung ist, einen möglichst mächtigen Maßstab zu erhalten. Wichtige Maßstabstypen sind Nominal-, Ordinal-, Interval-, Ratio- und Absolute-Maßstäbe [Sarle95].

Ein Großteil der Messungen sind unter bestimmten Voraussetzungen vom Maßstabstyp Ordinal, einige wenige haben den Maßstabstyp Ratio [Zuse91].

- Der Ordinal-Maßstab ist nur einordnend. Er kann durch jede monotone Funktion beschrieben werden. Die Zahlen oder Symbole bezeichnen nur die relative Position der Objekte und keine Differenzen. Die erlaubten Relationen sind z.B. Gleichwertigkeit, größer oder komplexer als.

- Der Ratio-Maßstab hat neben den Eigenschaften des Ordinal Maßstabs, das Element Null und die Eigenschaft, das Verhältnis zweier Größen bezeichnende Kennziffern darzustellen. Eine neue Relation ist z.B. die Addition.

2.1.2 Der Rahmen für Software-Messungen

Um eine Messung durchzuführen, müssen, wie oben beschrieben, Objekte und Attribute, die gemessen werden sollen, identifiziert werden. Bei Software wird in drei Objektklassen differenziert, deren Attribute gemessen werden [Fenton91]:

- *Prozesse* sind Aktivitäten, die sich auf die Software beziehen; Attribute sind z.B. die Zeit eines Prozesses oder die Anzahl gefundener Fehler.

- *Produkte* sind Werkzeuge oder Dokumente, die aus den Prozessen entstehen; Attribute sind z.B. die Länge und Modularität eines Programms, die Komplexität eines Algorithmus oder die Zuverlässigkeit und Wartbarkeit von Software.

[11] Es bestehen sogar Fehlertoleranzen bei gut verstandenen physikalischen Attributen, z.B. hängt die Größe eines Menschens von der Standposition und der Tageszeit ab.

[12] Ein Maßstab ist ein Tripel $(\mathcal{C}, \mathcal{N}, \mathcal{M})$, d.h. er beschreibt die Messung eines Attributes.

[13] Zulässige Transformationen sind Transformationen, die die relevanten Beziehungen der Messung erhalten, z.B. Änderungen der Einheit von Meter in Inch.

- *Ressourcen* sind z.B. eingesetzte Software und Hardware oder das Personal; Attribute sind z.B. Kosten von Software und Hardware oder das Alter des Personals.

Die zu messenden Attribute werden in interne und externe Attribute unterteilt:

- *Interne Attribute* eines Objektes sind solche, die aus dem Objekt an sich gemessen werden können, z.B. Größe eines Programms.

- *Externe Attribute* eines Objektes können nur unter dem Gesichtspunkt gemessen werden wie sich das Objekt auf seine Umgebung bezieht, z.B. Zuverlässigkeit oder Wartbarkeit von Software.

Im allgemeinen werden interne Attribute direkt gemessen und externe Attribute indirekt. Ein Attribut wird direkt gemessen, wenn die Messung unabhängig von anderen Attributen erfolgt. Eine indirekte Messung setzt die Messung von mindestens eines weiteren Attributes voraus.

Aus den Aktivitäten in der Software-Messung werden zwei Richtungen unterschieden [Fenton91]. Zum einen werden vorhandene Objekte bewertet, zum anderen sollen Attribute von Objekten, die noch nicht existieren, vorausgesagt werden. Die Leistungsfähigkeit dieser Voraussagen von *Software-Charakteristika*[14], darf jedoch nicht überbewertet werden.

Software-Messung ist ein Sammelbegriff für eine Vielzahl von Aktivitäten, die jeweils auf unterschiedlichen Modellen und den zugehörigen Messungen beruhen. Dazu zählen kosten- und leistungsabschätzende Messungen, Produktivitäts-Modelle, Qualitäts-Modelle, Zuverlässigkeits-Modelle, Bewertung der Performance, algorithmische und berechenbare Komplexität, strukturelle Messungen und Komplexitätsmessungen.

2.1.3 Charakteristik der Software-Komplexitätsmessung

Im Gegensatz zur Software-Messung existiert für den Begriff Komplexitätsmessung keine formale Definition. [Zuse91] versteht als *Software-Komplexität* den Schwierigkeitsgrad Software zu entwerfen, zu verstehen, zu pflegen, zu verändern, zu warten und zu testen. In [IEEE90] wird Komplexität als Grad eines Systems oder einer Komponente mit einem Design oder einer Implementierung, das schwierig zu verstehen und zu verifizieren ist, beschrieben. In der Literatur werden vorrangig zwei Arten von Software-Komplexität unterschieden: die berechenbare Komplexität[15] und die psychologische Komplexität[16] [Zuse91]. Die in der Diplomarbeit verwendeten Komplexitätsmessungen von [McCabe76] und [Halstead77] befassen sich mit der

[14] Software-Charakteristika sind z.B. Größe des spezifizierten Systems, Zuverlässigkeit und Wartbarkeit von Software.

[15] Die berechenbare Komplexität ist die Komplexität eines Algorithmus, z.B. die Anzahl von Operationen, die ein Sortieralgorithmus für die Sortierung von Daten benötigt.

[16] Die psychologische Komplexität ist die Komplexität von Programmen aus der Sicht des Entwicklers.

psychologischen Komplexität von Programmen.

Es gibt verschiedene Software-Komplexitäts-Kategorien. [Ejiogu85] geht auf fünf wichtige näher ein:

- *Strukturelle Komplexität* geht auf die Abfolge von Anweisungen in bezug auf ihre logischen Beziehungen in der Systemkomponente ein. Sie ist mehr als die Auswirkungen der Kontrollpfade in einem Software Modul und kann als globales Attribut für jedes Software Modul angesehen werden.

- *Berechende Komplexität* beschreibt den Schwierigkeitsgrad der arithmetischen und logischen Berechnungen eines Algorithmus auf eine Eingabe. Sie ist ein Attribut eines Algorithmus und kein direktes Attribut von Software.

- *Logische Komplexität* gibt den Schwierigkeitsgrad logischer Entscheidungen und Verzweigungen innerhalb des Systems an.

- *Konzeptionelle Komplexität* umfaßt die psychologische Auffassungskraft oder den Schwierigkeitsgrad in der Übernahme oder Beendigung eines Systems. Sie beinhaltet auch die Mannigfaltigkeit von Ressourcen.

- *Textuelle Komplexität* bezeichnet die statische Analyse eines Programmtextes. Diese Bedingung beeinflußt die Zuverlässigkeit.

Die Modularisierung[17] eines Programms ist eine Vorgehensweise beim Software-Entwurf. Das Ergebnis der Modularisierung ist eine hierarchische Anordnung von Modulen. Die Kriterien einer guten Modularisierung sind: Modulgeschlossenheit, Modulbindung, Modulkopplung, Modulgröße, Minimalität der Schnittstelle, Testbarkeit, Interferenzfreiheit, Verwendungszahl und Modulhierarchie [Pomberger93]. Sie sind Einflußfaktoren auf die Software-Komplexität [Zuse91]. Die Komplexität setzt sich aus Anteilen der inter-[18] und intramodularen[19] Komplexität zusammen. Die Verringerung der internen Komplexität eines Moduls führt zu einer Erhöhung der intermodularen Komplexität. Diese Komplexitätskenngrösen sollten im Zusammenhang gesehen werden, da sie sich gegenseitig beeinflussen können [Ebert96].

Die zyklomatische Komplexität von [McCabe76] beschreibt nach den Komplexitätskategorien von [Ejiogu85] die strukturelle Komplexität und das Verfahren von [Halstead77] charakterisiert berechenbare Komplexität. Die beiden Messungen sind nur auf die intramodulare Komplexität sensitiv.

Die Komplexität von Software ist ein internes Attribut aus der Objektklasse Produkt und wird direkt gemessen, d.h. direkt aus dem Programmtext ermittelt und hängt nicht von weiteren Attributen ab.

Es ist wichtig die Kriterien für den Maßstab der verwendeten Messung zu kennen.

[17] Die Modularisierung ist ein Prozeß der Zerlegung eines Programms in Einheiten, sogenannte Module. Dieser Mechanismus verbessert u.a. die Verständlichkeit eines Programms [Parnas72].

[18] Die intermodulare Komplexität beruht auf den Beziehungen von Modulen untereinander.

[19] Die intramodulare Komplexität ist die interne Komplexität eines Moduls

Dies setzt eine meßtheoretische Untersuchung der Software-Komplexitätsmessung voraus. Bezogen auf den Ordinal Maßstab werden empirische Relationen R definiert, die die Objekte C des Empirischen-Relationen-Systems $\mathcal{C} = (C, R)$ beschreiben. Werden die Objekte C, z.B. als Flußdiagramme angenommen, werden die empirischen Relationen auf den Flußdiagrammen betrachtet. Die intuitive Idee der Komplexität basiert auf den empirischen Relationen, d.h. wenn die Komplexität von Programmen, repräsentiert durch Flußdiagramme betrachtet werden soll, ist dies die Idee der Komplexität. Empirische Relationen sind binäre Relationen $\succ\!\bullet$, $\approx\!\bullet$, $\succeq\!\bullet$ definiert auf C. Für C, C$'$ $\in C$ gilt:

C $\succ\!\bullet$ C$'$	C ist komplexer als C$'$.
C $\approx\!\bullet$ C$'$	C und C$'$ sind gleich komplex.
C $\succeq\!\bullet$ C$'$	ist definiert als: C $\succ\!\bullet$ C$'$ oder C $\approx\!\bullet$ C$'$.

Es gilt für $((\mathcal{C}, \succeq\!\bullet), (\mathcal{N}, \geq), \mathcal{M})$: C $\succeq\!\bullet$ C$'$ $\Leftrightarrow \mathcal{M}(\mathrm{C}) \geq \mathcal{M}(\mathrm{C}')$ für alle C, C$'$ $\in C$.

Jedes Komplexitäts-Meßverfahren hat seine Stärken und Schwächen, die beim jeweiligen Einsatz bekannt sein müssen. Außerdem muß Kenntnis über die zu messenden Attribute existieren. [Zuse91] bemerkt, daß es keine Kriterien zur Auswahl einer Komplexitätsmessung bei einem vorliegenden Problem gibt und die Interpretation der Ergebnisse aus den Messungen schwierig ist. Probleme bereiten zum einen, daß eine Vielzahl von Methoden, die auf unterschiedlichen Ideen von Komplexität basieren, in nicht generell akzeptierten Kriterien beschrieben und bewertet sind. Zum anderen bestehen Schwierigkeiten in der Bewertung und im Vergleich der Komplexitätgrößen aus den Messungen.

Die bekanntesten und meist verwendeten Komplexitätsmessungen sind die Messungen nach [McCabe76] und [Halstead77]. Weit verbreitet, aber als Komplexitätsmessung kaum akzeptiert ist LoC [Zuse91].

2.1.4 Was wird unter intuitiver Komplexität verstanden?

Software-Komplexität ist allgemein der Schwierigkeitsgrad, Software zu entwerfen, zu verstehen, zu pflegen, zu verändern, zu warten und zu testen. Die in der Diplomarbeit verwendeten Meßverfahren von [McCabe76] und [Halstead77] messen die psychologische Komplexität von Programmen. Beide Verfahren sollen interpretiert, bewertet und die Ergebnisse miteinander verglichen werden. Dies stellt für ABAP/4[20]-Programme ein Problem dar. Es liegen keine Erkenntnisse vor, welche Ergebnisse die beiden Meßverfahren bei der jeweiligen Interpretationen liefern. Spiegeln die Resultate die intuitiven Einschätzungen wider? Können sie als Komplexitätsmaß für ABAP/4-Programme angesehen werden? Es gibt auch keine Erfahrungen wieweit die Ergebnisse miteinander verglichen werden können. Mit Hilfe der intuitiven Komplexität sollen die betrachteten ABAP/4-Programme eingeschätzt werden. Aufbauend auf diese Bewertung soll überprüft werden, ob die Ergebnisse

[20] ABAP/4 ist eine Abkürzung für Advanced Business Application Programming und ist laut SAP eine Programmiersprache der 4. Generation zur Entwicklung von Dialoganwendungen und zur Auswertung von Datenbanken.

aus den beiden Messungen vernünftig sind. Weiter soll die intuitive Komplexität eine Basis schaffen, um die Meßverfahren von McCabe und Halstead vergleichen zu können.

Intuition ist „das unmittelbare, ganzheitliche Erkennen oder Erfahren von Sachverhalten im Gegensatz zu der u.a. durch Beweis, Erklärung, Definition vermittelten diskursiven Erkenntnis. ... Charakteristisch für die Intuition ist, daß man den intuitiv eingesehenen Sachverhalt durch eine logische Analyse nicht evidenter machen kann. ... Psychologisch gesehen ist die Intuition das von einem Gefühl der Evidenz begleitete spontane und ganzheitliche Erfassen von Wirklichkeitszusammenhängen oder der Lösung wissenschaftlicher, technischer oder künstlerischer Aufgaben“ [Brockhaus89].

Abgeleitet aus der Intuition bewertet die intuitive Komplexität die psychologische Komplexität von Programmen. Sie bildet viele Facetten der Software-Komplexität ab. In ihr fließen u.a. die fünf aufgeführten Software-Komplexität-Kategorien wie strukturelle, berechenbare, logische, konzeptionelle und textuelle Komplexität ein. Weiterhin bildet sie die beschriebenen Kriterien der Modularisierung wie die Kopplung der Module, die konzeptionelle Aufspaltung in Module und die intramodulare Komplexität ab. Die intuitive Komplexität ist vom Maßstabstyp ordninal. Mit ihr können die Programme nur relativ zueinander eingeordnet werden. Relationen sind komplexer als und gleich komplex.

2.2 Messung nach McCabe: Zyklomatische Zahl

[McCabe76] führt in seinem Abstrakt *A Complexity Measure* die *zyklomatische Zahl* $v(G)$ ein, die eine quantitative Basis für die Modularisierung von Software beschreibt. Das Verfahren mißt die strukturelle Komplexität von Modulen.

McCabes Ziel ist, eine Antwort auf die Frage zu finden: „Wie kann ein Softwaresystem modularisiert werden, daß die entstehenden Module test- und wartbar sind?“. Dazu entwickelte er ein mathematisches Verfahren, das eine Basis für die Modularisierung darstellt und erlaubt Softwaremodule zu identifizieren, die schwer test- und wartbar sind.

Der Autor erklärt die Entscheidungslogik eines Algorithmus zum Ausgangspunkt seines Komplexitätsmaßes. Die logische Struktur von Software Modulen wird veranschaulicht, indem sie durch gerichtete Graphen dargestellt wird. Die Basis für die Transformation in einen Graphen und die Anwendung von Rechenverfahren bildet die Graphentheorie [Berge73]. McCabe mißt die Anzahl möglicher Pfade durch ein Modul. Als Pfade werden voneinander unabhängige geschlossene Regelkreise einer Entscheidungsstruktur verstanden.

McCabe geht zur Messung der Komplexität von Software von dessen Struktur aus, indem er die Komplexität eines Moduls gleich der möglichen Pfade durch ein Modul setzt. Dadurch ist die Komplexitätsmessung unabhängig von der Programmtextformatierung und nahezu unabhängig von der Programmiersprache, solange gleiche fundamentale Entscheidungstrukturen genutzt werden.

2.2.1 Theorie

Die von McCabe vorgestellte Komplexitätsmessung ist ein Verfahren, daß die Anzahl der Pfade in einem Programm mißt. Das Komplexitätsmaß kann direkt aus dem Programmcode ermittelt werden.

Die Komplexitätsmessung definiert die Basispfade, die in Kombination alle möglichen Pfade im Programm generieren, d.h. sie beinhaltet die Summe aller linear unabhängigen Pfade eines Moduls.

Zyklomatische Zahl

Die zyklomatische Zahl $v(G)$ eines Graphen G mit e Kanten, n Knoten und p miteinander verbundenen Modulen ist definiert durch:

$$v(G) \quad = \quad e - n + 2p \tag{2.1}$$

Das von McCabe aufgestellte Theorem lautet, daß in einem geschlossenen Graph G die zyklomatische Zahl gleich der maximalen Anzahl linear unabhängiger Regelkreise ist.

Jedes Programm, das mit einem Graphen in Verbindung gebracht wird, hat einen Eingangs- und einen Ausgangsknoten. Jeder Knoten korrespondiert mit einem Programmfragment, bestehtend aus einer sequentielle Abfolge von Befehlen oder aus Verzweigungen. Dieser Graph heißt klassisch Programm-Kontroll-Graph. Zum Beispiel hat der folgende Beispielgraph G den Eingangsknoten a und den Ausgangsknoten f.

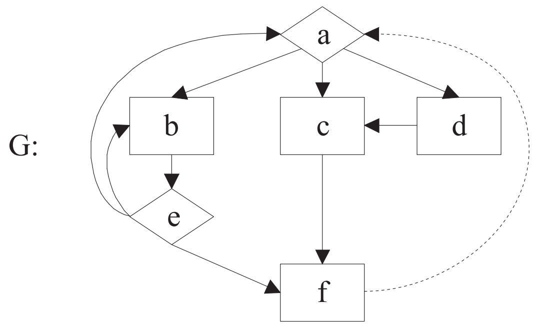

Abbildung 2.1: Beispielgraph G

Eigenschaften des Graphen G

Die Eigenschaften des Graphen G lauten:

- Der Graph G muß genau einen Eingangs- und Ausgangsknoten haben, von dem aus jeder andere Knoten erreicht werden kann oder der für jeden anderen Knoten erreichbar ist.
- Der Graph G muß geschlossen sein, d.h. es existiert eine Kante zwischen Eingangs- und Ausgangsknoten, um formal die Graphentheorie anzuwenden. Bei der zyklomatischen Zahl zählt diese Kante nicht; im Beispielgraph in Abbildung 2.1 die gestrichelte Linie.
- Jeder Knoten des Graphen G korrespondiert zu einem Programmtextfragment, das eine Sequenz oder eine Verwzeigung darstellt, d.h. die Knoten repräsentieren berechnende Programmfragmente, verzweigen oder führen den Steuerfluß zusammen.
- Die Kanten stellen den Kontrollfluß zwischen den Knoten dar.
- Jeder Knoten muß mindestens eine Ausgangskante besitzen.
- Jeder Verzweigungsknoten muß genau eine Bedingung enthalten.

Nach McCabe ist die Komplexität eines Moduls gleich der Anzahl der möglichen Pfade in dem Modul. Pfade sind voneinander unabhängige geschlossene Regelkreise der Entscheidungsstruktur in G. Ein Regelkreis ist dadurch gekennzeichnet, daß sein Anfangsknoten gleich dem Endknoten ist.

Der Beispielgraph G in Abbildung 2.1 hat folgende unabhängige Regelkreise:

(abefa), (beb), (abea), (acfa), (adcfa).

Als Basispfade gelten die Pfade:

(abef), (abeabef), (acf), (adcf).

Die zyklomatische Komplexität des Beispielgraphen G ist:

$$v(G) \;=\; e - n + 2p \;=\; 9 - 6 + 2 \;= 5.$$

Kontroll-Graphen der Basiskonstrukte

Die minimale Anzahl von Konstrukten, die in der strukturierten Programmierung benötigt werden, sind:

- Sequenzen
- Bedingte Ausführungen: If-Anweisungen
- Iterationen: While- und Repeat-Schleifen

Die Kontrollgraphen und deren Komplexität der oben aufgeführten Konstrukte sind in Abbildung 2.2 dargestellt:

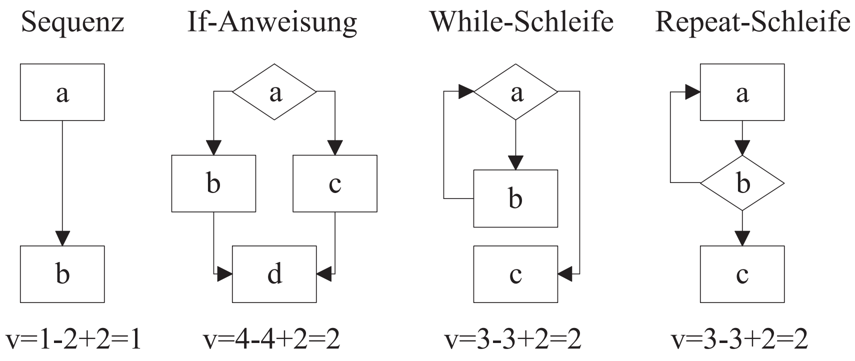

Abbildung 2.2: Basiskonstrukte und ihre Komplexität nach McCabe

Eigenschaften der zyklomatischen Komplexität

McCabe gibt folgende sechs Eigenschaften für die zyklomatische Komplexität an:

1. $v(G) \geq 1$.
2. $v(G)$ die ist maximale Anzahl der linear unabhängigen Pfade in G; sie wird auch die Größe der Basispfade genannt.
3. Das Einfügen oder Löschen von Sequenzen in G hat keinen Einfuß auf $v(G)$.
4. G hat nur einen Pfad, wenn gilt: $v(G) = 1$.
5. Beim Einfügen einer neuen Verzweigung, wie einer bedingten Ausführung, auch Fallunterscheidungen genannt oder Iteration in G, erhöht sich $v(G)$ um die entsprechende Einheit.
6. $v(G)$ hängt nur von der Entscheidungsstruktur von G ab.

Durch die zyklomatische Komplexität soll erreicht werden, daß bei der Softwareentwicklung eine Trennung in Module anhand der zyklomatischen Zahl erfolgt und nicht nach der physikalischen Größe des Programmtextes. McCabe schlägt eine zyklomatische Zahl von zehn als vernünftiges Limit für Module vor. Dadurch bleiben die Module überschaubar, erlauben ein Testen aller unabhängigen Pfade und sind einfach zu modifizieren. Im Fall von verketteten Fallunterscheidungen, macht das Limit keinen Sinn.

Vereinfachung

Für strukturierte Programme, die keine Modulen beinhalten, schlägt McCabe eine vereinfachte Komplexitätsberechnung vor. Die Intention ist, keine gerichteten Graphen erstellen zu müssen, sondern die zyklomatische Komplexität direkt aus den syntaktischen Konstrukten der Programme zu ermitteln.

Die zyklomatische Komplexität von strukturierten Programmen ist gleich der Anzahl von Verzweigungen π plus eins:

$$v(G) \quad = \quad \pi + 1. \tag{2.2}$$

Verzweigungen sind für McCabe bedingte Ausführungen und Iterationen. Liegen binär verkettete Verzweigungen vor, wird die zyklomatische Komplexität um die Anzahl der Binäroperatoren der Verzweigung erhöht.

Der Beispielgraph G in Abbildung 2.1 besteht aus zwei Verzweigungen mit jeweils einer binären Verkettung:

$$v(G) \quad = \quad \pi + 1 \quad = \quad 2 + 2 + 1 \quad = \quad 5.$$

Modularisierung

Durch die Modularisierung wird die logische Komplexität der Aufgabenstellung in Teilaufgaben geringere Komplexität zerlegt, so daß strukturelle Komplexität im Programm entsteht.

Für ein Programm C, das in k Module C_i, $1 \leq i \leq k$ unterteilt ist, gilt für jedes Modul C_i mit e_i Kanten und n_i Knoten folgende Eigenschaft:

$$\begin{aligned} v(C) \quad &= \quad e - n + 2p \quad = \quad \sum_{i=1}^{k} e_i - \sum_{i=1}^{k} n_i + 2k \\ &= \quad \sum_{i=1}^{k} (e_i - n_i + 2) \quad = \quad \sum_{i=1}^{k} v(C_i) \end{aligned}$$

Wird ein Modul C_i in einem Programm r-mal aufgerufen, fließt es r-mal in die zyklomatische Komplexität $v(C)$ ein, d.h. es wird wie r verschiedene Module behandelt.

Strukturiertheit

Die Quantität logischer Entscheidungen wird mittels der zyklomatischen Komplexität gemessen, ebenso ist die Qualität der Logik ein signifikanter Faktor in der Softwareentwicklung.

Strukturierte Programmierung vermeidet schwer wartbaren „Spaghetticode“, indem die Verwendung von Kontrollstrukturen begrenzt wird. Die zugelassenen syntaktischen Konstrukte, auch D-Strukturen genannt, sind einfach analysier- und zerlegbar. Sie sind in Abbildung 2.2 mit ihrer zyklomatischen Komplexität dargestellt. Die Stärke dieser Programmierung liegt in der einfachen Auflösung von komplexen Strukturen. Derartige Zerlegungen erleichtern die Modularisierung von Software, da die Basiskonstrukte Komponenten mit Eingangs- und Ausgangsknoten darstellen.

Voraussetzung für das Messen der Strukturiertheit und Komplexität von Programmen ist eine vorherige Definition der unstrukturierten Konstrukte. Ein unstrukturiertes Programm besitzt nach McCabe mindestens zwei der folgenden Eigenschaften:

- Sprung aus einer oder in eine Schleife
- Sprung aus einer oder in eine Fallunterscheidung

Bei der Generierung aller möglichen Kombinationen der vier unstrukturierten Konstrukte, können diese auf vier Basis-Typen reduziert werden:

1. Sprung aus einer und in eine Schleife
2. Sprung aus einer Schleife und aus einer Fallunterscheidung
3. Sprung in eine Schleife und in eine Fallunterscheidung
4. Sprung aus einer und in eine Fallunterscheidung

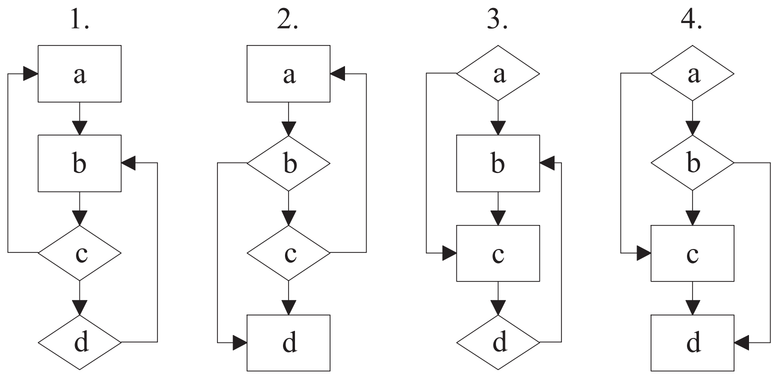

Abbildung 2.3: Basistypen unstrukturierter Konstrukte

Die zyklomatische Komplexität eines unstrukturierten Programms ist mindestens drei.

Bei Graphen unstrukturierter Programme ist es nicht möglich, diese in Teilgraphen mit Eingangs- und Ausgangsknoten zu zerlegen. Ein strukturiertes Programm kann zu einem Programm mit der Komplexität eins zurückgeführt werden; siehe Abbildung 2.4.

Als ein Maß für die Strukturiertheit von Programmen definiert McCabe die essentielle Komplexität $ev(G)$, d.h sie gibt den Mangel an Struktur eines Programms wieder.

$$ev(G) \quad = \quad v(G) - n \tag{2.3}$$

Die essentielle Komplexität wird soweit wie möglich aus dem reduzierten Graph G ermittelt, wobei n die Anzahl der Teilgraphen mit Eingangs- und Ausgangsknoten ist. Es gilt: $1 \leq ev(G) \leq v(G)$. Die essentielle Komplexität eines strukturierten Programms ist gleich eins.

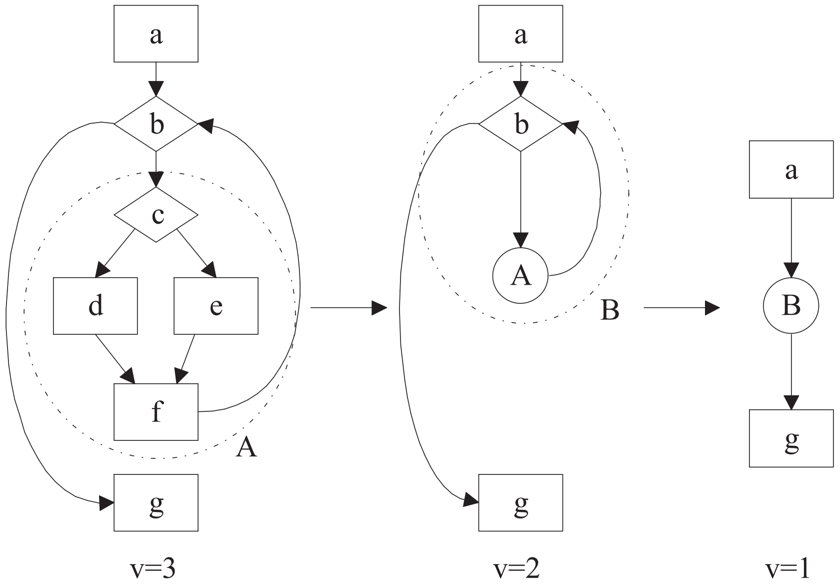

Abbildung 2.4: Beispiel einer Graph-Reduktion

2.2.2 Charakteristiken

Die zyklomatische Komplexität von McCabe ist ein weitverbreitetes und akzeptiertes Verfahren in der Komplexitätsmessung, das eine Reihe von Stärken hat, aber auch einige Schwächen aufweist.

Die Stärken des Verfahrens sind nach [Li87] und [Zuse91]:

- Die Komplexitätsmessung ist unabhängig von der Programmformatierung und nahezu unabhängig von der Programmiersprache, da die Messung von der Struktur des Programms ausgeht. Dieses gilt nur, wenn gleiche fundamentale Entscheidungsstrukturen vorliegen.
- Mit Hilfe der zyklomatischen Zahl kann eine klare Programmstrukturen entworfen werden, d.h. Software-Programme können derart modularisiert werden, daß die entstehenden Module gut test- und wartbar sind.
- Durch das Aufstellen der Basispfade wird ein strukturiertes Testen von Modulen unterstützt. Die zyklomatische Komplexität ermöglicht die Angabe der minimalen Anzahl notwendiger Tests, die zyklomatische Zahl selbst [McCabe96].
- Die Entscheidungskonstrukte sind einfach zu interpretieren, d.h. für alle Entscheidungsstrukturen und unstrukturierten Konstrukte können leicht die entsprechenden Kontroll-Graphen der Basiskonstrukte ermittelt werden.
- Der Kontroll-Graph zu einem Software Programm kann problemlos, auch automatisiert per Programm, aufgestellt werden. Die zyklomatische Zahl ist dann einfach aus dem Kontroll-Graph abzuleiten.

- Die zyklomatische Zahl kann direkt aus dem Programmtext generiert werden, indem die Verzweigungen gezählt werden.

- Mit der zyklomatischen Komplexität kann die Komplexität von Modulen kontrolliert werden.

Neben den oben aufgezeigten Stärken, weist die zyklomatische Komplexität einige Schwächen auf:

- Das Verfahren kann keine Mehrfachaufrufe von Modulen abbilden, d.h. wird ein Modul r-mal aufgrufen, wird es nach McCabe wie r unterschiedliche Module behandelt und abgebildet [Roth87].

- Die zyklomatische Komplexität ist kein Maß für sequentielle Programmfragmente oder Module; die zyklomatische Zahl ist unabhängig von der Anzahl Befehlen immer eins [Li87, Weyuker88, Zuse91].

- Die zyklomatische Zahl ist nicht sensitiv auf die Reihenfolge von Anweisungen, d.h. die Reihenfolge von Anweisungen übt keinen Einfluß auf die Komplexität des Moduls aus [Weyuker88].

- McCabe macht zum einen keinen Unterschied zwischen durch Binäroperationen verketteten Verzweigungen und verschachtelten Verzweigungen. Zum anderen können Verschachtelungen durch die zyklomatische Komplexität nicht dargestellt werden [Roth87].

- Verzweigungen wie bedingte Ausführungen und Iterationen werden von McCabe nicht unterschieden; sie haben die gleiche zyklomatische Komplexität [Li87, Zuse91].

- Das McCabesche Verfahren ermöglicht keine Abbildung von Modulbeziehungen. Es zeigt weder den Zusammenhang der Module noch die Verbindung dieser zueinander auf [Li87, Zuse91].

2.3 Messung nach Halstead: Länge, Größe, Leistung

Das von [Halstead77] in *Elements of Software Science* beschriebene Meßverfahren quantifiziert Modul-Komplexität direkt vom Programmtext mit Schwerpunkt auf berechende Komplexität. Die Meinungen über das Verfahren reichen von „unzuverlässig und undurchsichtig“ [Jones94, vanDoren97b] bis „eine starke Messung für die Wartbarkeit“ [Oman91, vanDoren97b].

Halsteads Ziel war, ein möglichst einfaches Meßverfahren zu entwickeln, das auf eine große Anzahl von Programmiersprachen anwendbar ist. Dazu werden Regeln zur Identifizierung von Operatoren und Operanden festgelegt, um die Komplexität direkt aus den Operatoren und Operanden im Modul oder Programm zu bestimmen [vanDoren97b].

Die Grundmessungen von Halstead sind Programm-Länge N, Programm-Größe V, Informations-Inhalt I und Programmier-Leistung E. Die Berechnung diese Messungen ist einfach. Schwierigkeiten bestehen bei der Definition der Regeln zur Identifizierung von Operatoren und Operanden. In einigen Fällen ist es nicht immer eindeutig, ob es sich um einen Operatoren oder einen Operanden handelt [SMLab97, vanDoren97b]. Mit großem Auhwand geschieht die eigentliche Messung, d.h. die Ermittlung der Anzahl von Operatoren und Operanden im Programmtext.

Halsteads Komplexitätsmessung, eine der ältesten Meßverfahren, wird aufgrund einiger Schwächen selten eingesetzt. Der Hauptkritikpunkt ist, daß die Messungen eher lexikalische und textuelle Komplexität anzeigen, als struktuelle oder logische. Wenn diese Schwächen bekannt sind, liefert Halsteads Messung gute Ergenisse.

2.3.1 Theorie

Halsteads Meßverfahren basiert auf den algorithmischen Charakteristika eines Algorithmus, Moduls oder Programms. Es kann direkt auf dem Programmtext durchgeführt werden, indem die Anzahl von Operatoren und Operanden ermittelt wird.

Die Kritik, daß Halstead eher lexikalische und textuelle Komplexität anzeigt, kann auch als ein Vorteil angesehen werden. Wenn im Programmcode ein hohes Verhältnis von berechnenden zu verzweigenden Anweisungen herrscht, kann die Komplexität besser abgebildet werden, als z.B. mit der zyklomatischen Komplexität von [McCabe76]. Berechnende Anweisungen sind Programmschlüsselworte in sequentieller Folge, ohne strukturelle Eigenschaften. Dagegen beeinflussen verzweigende Anweisungen wie bedingte Ausführungen und Iterationen die Struktur eines Moduls oder Programms.

Basiseigenschaften

Um meßbare Merkmale eines Algorithmuses zu bestimmen, sind nach Halstead Basismetriken Voraussetzung, die direkt vom statischen Ausdruck des Algorithmus in jeder Sprache zu entwickeln sind.

Bei jedem implementierten Algorithmus in jeder Sprache ist es möglich Operatoren und Operanden festzulegen. Operanden sind die benutzten Variablen und Konstanten, die von Operatoren verwendet und verändert werden. Schwierigkeiten bei der eindeutigen Identifizierung von Operatoren und Operanden sollen mit Hilfe von aufgestellten Regeln beseitigt werden. Wichtig ist die Konsistenz, d.h. die exakte und konstante Zähltechnik.

Halstead gibt nur wenig Hilfestellung bezüglich der oben genannten Identifizierungsproblematik: Operatoren sind Zuweisungen wie z.B. :=, Vergleiche wie z.B. $<, =, >$ und Rechenoperationen wie z.B. $+, -, /, \cdot$. Alle Schlüsselworte einer Sprache gelten als Operatoren. Dies sind Kontrollstrukturen und sequentielle Anweisungen. Klammernde Operatoren zählen als ein Operator. Kommentare und deklarative

Schlüsselworte sind keine Operatoren. Operanden bilden Konstanten, deklarierte Variablen und Programm- und Modulnamen.

Allgemein können als Operatoren Metazeichen und Schlüsselworte, die Einfluß auf Operanden und auf den Programmablauf nehmen, angesehen werden. Im Einzelfall muß in Abhängigkeit der Implementierungssprache und dem aktuellen Kontext eine Entscheidung fallen, ob das Metazeichen oder Schlüsselwort als Operator gezählt wird oder nicht.

Meß- und zählbare Eigenschaften eines Algorithmus bestehen aus:

η_1	=	Anzahl eindeutiger Operatoren im Programmtext
η_2	=	Anzahl eindeutiger Operanden im Programmtext
N_1	=	Anzahl aller Operatoren im Programmtext
N_2	=	Anzahl aller Operanden im Programmtext
$f_{1,j}$	=	Anzahl des Auftretens des j-ten vorkommenden Operators mit $1 \leq j \leq \eta_1$
$f_{2,j}$	=	Anzahl des Auftretens des j-ten vorkommenden Operanden mit $1 \leq j \leq \eta_2$

Es gelten folgende Beziehungen zwischen den oben aufgeführten Definitionen:

$$N_1 = \sum_{j=1}^{\eta_1} f_{1,j}$$

$$N_2 = \sum_{j=1}^{\eta_2} f_{2,j}$$

Vokabular η

Das Vokabular η eines Programms ist definiert durch:

$$\eta = \eta_1 + \eta_2 \tag{2.4}$$

Aufgrund der Möglichkeit der Deklarationen neuer Funktionen existiert für η_1 keine obere Grenze.

Programm-Länge N

Für die Programm-Länge N gilt:

$$\begin{aligned} N &= N_1 + N_2 \\ &= \sum_{i=1}^{2} \sum_{j=1}^{\eta_i} f_{i,j} \end{aligned} \tag{2.5}$$

Die Programm-Länge N, abhängig vom Vokabular η, besitzt eine untere und obere Schranke.

Die untere Schranke wird durch das Vokabular η mit $\eta \leq N$ festgelegt.

Für die Bestimmung der oberen Schranke wird folgende Voraussetzung getroffen. Das Programm der Länge N wird in Unterausdrücke der Länge η unterteilt und besteht dann aus $\frac{N}{\eta}$ Ausdrücken der Länge η. Damit eine obere Schranke für N existiert, dürfen im Programm keine zwei Unterausdrücke der Länge η vorkommen, die gleich sind. Die Bedingung, daß das Programm nicht aus identischen Unterausdrücken der Länge η bestehen darf, ist sinnvoll. Den Unterausdrücken können dadurch eindeutige Namen zugeordnet werden, die eine mehrfache Auswertung verhindern. Wird ein Unterausdruck der Länge η mehr als einmal benötigt, bekommt dieser einen eindeutigen Namen und das Vokabular η erhöht sich um Eins.

Die Anzahl möglicher Kombinationen von η Ausdrücken, die η-mal zugeordnet werden, beträgt: η^η. Ein Programm besteht aus maximal η^η Ausdrücken der Länge η.

$$N \leq \eta^{\eta+1}$$

ist die obere Schranke von N.

Unter der Annahme, daß das Vokabular η aus Operatoren η_1 und Operanden η_2 besteht, die sich abwechseln, ergibt sich als obere Schranke:

$$N \leq \eta \cdot \eta_1^{\eta_1} \cdot \eta_2^{\eta_2}$$

Die obere Schranke bezieht sich nicht nur auf das betrachtete Programm mit N-Elementen, sondern auch auf alle möglichen Programmteile. Die Familie aller möglichen Programmteile, "Power Set" genannt, besteht aus 2^N Elementen.

Für die berechnete Länge $\widehat{N}$ einer Implementierung eines Algorithmus gilt:

$$\begin{aligned} 2^N &= \eta_1^{\eta_1} \cdot \eta_2^{\eta_2} \\ \widehat{N} &= \log_2(\eta_1^{\eta_1} \cdot \eta_2^{\eta_2}) \\ &= \log_2 \eta_1^{\eta_1} + \log_2 \eta_2^{\eta_2} \\ &= \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \end{aligned} \tag{2.6}$$

Eine Differenz zwischen der gemessenen Länge N und der berechneten Länge $\widehat{N}$ kommt aufgrund einer Vielzahl von Möglichkeiten der Implementierung eines Algorithmus zustande.

Programm-Größe V

Neben der Länge N ist ein weiteres wichtiges Programm-Kriterium die Größe V.

Die Hauptanforderung an die Größen-Metrik ist, daß sie unabhängig von der verwendeten Sprache, d.h. unabhängig von der Syntax der Sprache sein muß. Das Problem wäre sonst, daß bei einer Umsetzung eines Programms von einer Sprache in eine andere, sich die Programmgröße sonst ändern würde. Das muß die Metrik abbilden.

Für die Lösung wird vorausgesetzt, daß die Operatoren und Operanden als binäre Zahlen oder Bits repräsentiert werden. Das Ergebnis ist eine minimale Länge für die Repräsentation der längsten Operatoren und Operanden. Die Länge ist nur noch von der Anzahl der Elemente im Vokabular η abhängig.

Die Größe V wird definiert als:

$$\begin{aligned} V &= N \log_2 \eta \\ &= (N_1 + N_2) \log_2(\eta_1 + \eta_2) \end{aligned} \tag{2.7}$$

Diese Interpretation gibt die Programm-Größe in der Einheit Bits an.

Potentielle-Größe V^*

Die Potentielle-Größe V^* ist die kürzeste oder prägnanteste Form, in der ein Algorithmus präsentiert werden kann. Sie setzt die frühere Existenz einer Sprache voraus, in der die Operatoren bereits als Unterprogramme deklariert sind. Die Implementierung in dieser Sprache erfordert nur eine Namenszuweisung der Operanden für Argumente und Ergebnisse.

Die Potentielle-Größe ist definiert durch:

$$V^* = (N_1^* + N_2^*) \log_2(\eta_1^* + \eta_2^*)$$

In ihrer minimalen Form sind weder für Operatoren noch für Operanden Wiederholungen erforderlich. Es gilt:

$$N_1^* = \eta_1^* \quad ; \quad N_2^* = \eta_2^*$$

$$V^* = (\eta_1^* + \eta_2^*) \log_2(\eta_1^* + \eta_2^*) \tag{2.8}$$

Programm-Level L

Um den Programm-Level L zu messen, muß auf quantitative und meßbare Terme reduziert werden, d.h. es soll die Differenz von Programmerkmalen zwischen verschiedenen Ausprägungen eines Algorithmuses gemessen werden. Der Level der Implementierung ist wichtig, da er die Schreibleistung des Entwicklers, die Fehlerneigung und die Verständlichkeit des Programms beeinflußt. Voraussetzung für eine geeignete Metrik ist die Unterscheidung zwischen dem Level des Programms und dem Level der Sprache.

Der Programm-Level L einer Implementierung eines Algorithmuses ist definiert als:

$$L = \frac{V^*}{V} \tag{2.9}$$

Die prägnanteste mögliche Ausprägung eines Algorithmuses hat den Level $L = 1$; für andere Implementierung gilt: $L \leq 1$.

Daraus ergibt sich das Phänomen, daß eine potentielle Sprache am einfachsten angewendet werden kann. Dieses setzt aber voraus, daß jede mögliche Prozedur bereits implementiert ist. Da die Anzahl der möglichen Prozeduren unendlich ist, ist auch die Kenntnis aller Prozeduren unmöglich.

Der Programm-Level L wirkt sich doppelt auf die Verständlichkeit von Programmen aus:

- Eine, die Sprache gut beherrschende Person versteht schneller und einfacher den im höheren Level implementierten Algorithmus.
- Eine weniger vertraute Person benötigt zum Verständnis einen niedrigeren Level und eine höhere Programmgröße V.

Zusammengefaßt, verändert sich für Personen, die eine Sprache gut beherrschen, die Schwierigkeit des Verstehens invers zum Programm-Level L.

Berechneter Programm-Level $\widehat{L}$

Wird der Programm-Level $\widehat{L}$ aus der Implementierung eines Algorithmuses berechnet, muß vorausgesetzt werden, daß Operatoren und Operanden einen eigenen Einfluß auf den Programm-Level L ausüben. Es gilt zum einen, je größer die Anzahl eindeutiger Operatoren η_1, desto geringer ist der Implementierungslevel:

$$L \sim \frac{\eta_1 *}{\eta_1}$$

Zum anderen gilt, je mehr Operanden wiederholt werden, desto geringer ist der Implementierungslevel:

$$L \sim \frac{\eta_2}{N_2}$$

Für den berechneten Programm-Level $\widehat{L}$ gilt:

$$\widehat{L} = \frac{\eta_1^*}{\eta_1}\frac{\eta_2}{N_2} \tag{2.10}$$

Informations-Inhalt I

Der Informationsgehalt eines Algorithmuses ändert sich bei der Implementierung in verschiedenen Sprachen nicht. Halstead bezeichnet mit dem Informations-Inhalt I ein Maß für den inneliegenden Inhalt eines Algorithmus unabhängig der Sprache.

$$\begin{aligned} I &= \widehat{L}V \\ &= \frac{2}{\eta_1}\frac{\eta_2}{N_2}(N_1 + N_2)\log_2(\eta_1 + \eta_2) \end{aligned} \tag{2.11}$$

Der Informations-Inhalt I korreliert hoch mit dem Potential Volume V^*. Da V^* unabhängig von der Sprache ist, in der der Algorithmus implementiert wurde, ist auch I unabhängig. Dies bedeutet, daß I nur steigt, wenn auch die Komplexität des Problems steigt.

Programmier-Leistung E

Die Programmier-Leistung I ist beschränkt auf die mentale Aktivität, um einen erdachten Algorithmus in eine Sprache, die der Entwickler beherrscht, zu implementieren. Unter der Voraussetzung, daß das Konzept der Programmier-Leistung wie beschrieben, beschränkt ist, geben die bisherigen Metriken und Konzepte einen Einblick in den Programmierprozeß und stellen den Rahmen für die Quantifizierung der Programmier-Leistung E dar.

Die Beziehung zwischen den bisherigen Metriken und der mentalen Aktivität des Entwicklers bilden die Grundlage für die Definition der Programmier-Leistung E. Halstead unterteilt die Herleitung der Programmier-Leistung E in sechs aufeinander aufbauende Schritte:

1. Annahme 1: Die Implementierung jedes Algorithmuses besteht aus N Selektionen vom Vokabular mit η Elementen.

2. Annahme 2: Jede Selektion vom Vokabular η erfolgt auf einer nicht zufälligen Basis. Es sind $\log_2 \eta$ Vergleiche zur Auswahl jedes Elements nötig.

3. Aus den Annahmen 1 und 2 resultiert, daß für eine Programmentwicklung $N \log_2 \eta$ mentale Vergleiche erforderlich sind.

4. Mit der definierten Programm-Größe V mit $V = N \log_2 \eta$ folgt aus 3., daß die Größe V die Anzahl der mentalen Vergleiche ist, um ein Programm zu entwickeln.

5. Jeder mentale Vergleich bedarf einer Anzahl elementarer mentaler Entscheidungen, wobei diese Anzahl die Messung der Schwierigkeit einer Aufgabe ist. Der Programm-Level L ist reziprok zur Programm-Schwierigkeit.

6. Die Programm-Größe V zeigt die Anzahl mentaler Vergleiche an Die durchschnittliche Anzahl elementarer mentaler Entscheidungen ist $\frac{1}{L}$ für jeden mentalen Vergleich. Aus den beiden Voraussetzungen kann geschlossen werden, daß die Gesamtanzahl der elementaren mentalen Entscheidungen, gebraucht wird, um ein Programm zu entwickeln. Diese Gesamtanzahl wird Programmier-Leistung E genannt:

$$E = \frac{V}{L} \quad \text{mit } L = \frac{V^*}{V} \tag{2.12}$$

$$= \frac{V^2}{V^*} \tag{2.13}$$

Die Programmier-Leistung E beschreibt, daß die mentale Leistung, die zur Implementierung eines Algorithmus zu einer gegebenen Potentiellen-Größe V^* benötigt wird, mit dem Quadrat der Programm-Größe V steigt. Hieraus ergibt sich, daß im verteilten Programmieren durch die Modularisierung die Programmier-Leistung E reduziert werden kann.

Sprachlevel λ

Unter Beibehaltung der Implementierungssprache und einer Veränderung des Algorithmuses ist folgende Beziehung zwischen dem Programm-Level L und der Potentiellen-Größe V^* erkennbar: steigt V^*, sinkt L proportional. Dementsprechend ist das Produkt aus L und V^*, der Sprachlevel λ für jede Sprache konstant.

$$\begin{aligned} \lambda &= LV^* \quad \text{mit } V^* = LV && (2.14) \\ &= L(LV) \; = \; L^2V && (2.15) \end{aligned}$$

Der Sprachlevel λ steigt mit Sprachen von steigendem Funktionsumfang.

2.3.2 Modularitätsbetrachtung

Es kann nicht gewährleistet werden, daß sich modularisierte Programme ähnlich wie unmodularisierte verhalten, da $\widehat{N}$ nicht linear ist. Hierzu gibt Halstead ein Gegenbeispiel an: Voraussetzung:

$$\begin{aligned} \eta = 16 \quad &\text{mit} \quad \eta_1 = \eta_2 = 8 \\ N \;\; &= 48 \quad \widehat{N} \end{aligned}$$

Das Programm wird in zwei Teile A und B mit gleicher Länge $N_A = N_B = 24$ zerlegt, wobei alle Operatoren und Operanden des ganzen Programms jeweils in beiden Teilen sind.
Es gilt:

$$\widehat{N}_A = \widehat{N}_B = 8\log_2 8 + 8\log_2 8 = 48$$

Es ergibt sich:

$$\widehat{N}_A + \widehat{N}_B = 96$$

anstatt:

$$N = 48$$

Auch für den Fall, daß in beiden Teile keine gleichen Operatoren und Operanden sind, mit:

$$\begin{aligned} \eta_{1_A} = \; \eta_{1_B} &= 4 \\ \eta_{2_A} = \; \eta_{2_B} &= 4 \end{aligned}$$

gilt:

$$\widehat{N}_A = \widehat{N}_B = 4\log_2 4 + 4\log_2 4 = 16$$

Hier ergibt sich:

$$\widehat{N}_A + \widehat{N}_B = 32$$

anstatt:

$$N = 48$$

Für das Vokabular η hingegen, spielt es keine Rolle, ob das Programm unmodularisiert oder modularisiert ist.

2.3.3 Charakteristiken

Das Meßverfahren von Halstead ist ein erstes Beispiel einer vollständigen Software-Messung. Es besitzt eine Reihe von Schächen, weist aber auch einige Stärken auf.

Die Stärken Software-Messung nach Halstead sind:

- Das Meßverfahren ist unabhängig von der Programmformatierung und ist auf eine große Anzahl von Programmiersprachen anwendbar. Die Sprache darf nur aus Operatoren und Operanden bestehen. Regeln müssen zur Identifizierung von Operatoren und Operanden existieren.

- Das Verfahren basiert auf algorithmischen Charakteristiken eines Algorithmuses. Die Messungen sind sensitiv auf sequentielle Anweisungen, die keinen Einfluß auf die Struktur eines Moduls haben. Sequentielle Anweisungen sind z.B. Zuweisungen, Berechnungen und die Ausgabe auf den Bildschirm [Weyuker88].

- Die Bestimmung der Operatoren und Operanden erfolgt direkt aus dem Programmtext. Jede Metrik ist direkt aus dem Programmtext ermittelbar, d.h. einfach zu berechnen [vanDoren97b].

Der Kritikpunkt, daß das Meßverfahren von Halstead nur lexikalische und textuelle Komplexität abbildet, d.h. sensitiv gegenüber berechender Komplexität ist, kann auch als Stärke angesehen werden. Dies ist abhängig von der Zielsetzung der Komplexitätsmessung.

Bemängelungen am Verfahren von Halstead sind:

- Halstead hat einige Annahmen der Software-Wissenschaft falsch auf Ergebnisse von kognitiven psychologischen Studien angewendet. Nach [Coulter83] gibt es sich widersprechende Sachverhalte z.B. die Suche im menschlichen Gedächtnis und die Programmier-Leistung E.

- Die Festlegung der Regeln zur Identifizierung von Operatoren und Operanden ist schwierig. Halstead gibt keine eindeutige Abgrenzung zwischen Operator und Operand an [SMLab97, vanDoren97b]. [Halstead77] nennt nur einige Beispiele, für die Zuordnung der Operatoren. Eine genaue Definition fehlt. Die Identifizierung der Operanden ist eindeutig: nur Variablen. Halstead geht nicht auf die Behandlung von z.B. Konstanten, Zahlenliterale und Modulnamen ein.

- Die Ermittlung der Anzahl eindeutiger Operatoren η_1 und Operanden η_2 sowie die Gesamtanzahl der Operatoren N_1 und Operanden N_2 in einem Modul oder Programm ist aufwendig. [vanDoren97b] schreibt, daß der Einsatz eines sprach-abhängigen Parser, der für jede Sprache aufgestellt werden kann, die Bestimmung unterstützt.

- Das Leistungsmaß E bildet nicht die Eigenschaft ab, daß sich ein Programm aus einfacheren Modulen zusammensetzt, d.h. die relative Betrachtung der Komplexität von Modulen. Voraussetzung ist, daß die Komplexität der einzelnen Module nicht größer als die Komplexität des Programms ist [Weyuker88].

- Die Halstead-Metriken sind nicht sensitiv auf die Reihenfolge von Anweisungen, d.h. die Reihenfolge von Anweisungen übt keinen Einfluß auf die Komplexität des Moduls aus [Weyuker88].

- Halsteads Messungen beanspruchen allgemeingültig zu sein und bilden kein besonderes Ziel ab. Dies ist konträr zur heutigen Auffassung von Software-Messungen: die Stellung der Frage nach dem Ziel der Messung ist erforderlich [Fenton91].

- Das Meßverfahren bildet keine verschachtelte Fallunterscheidungen oder Iterationen ab [Bezbroz97].

- Halstead unterscheidet keine Verzweigungen wie bedingte Ausführungen und Iterationen. Eine Differenzierung könnte über eine unterschiedliche Gewichtung erfolgen, die Halstead nicht vornimmt [Bezbroz97].

2.4 R/2 System der SAP AG

Das System R/2 der SAP AG ist eine integrierte Anwendungssoftware für die betriebliche Informationsverarbeitung. Sie deckt viele betriebswirtschaftliche Funktionen, z.B. Vertrieb, Kostenrechnung, Buchhaltung, Materialwirtschaft und Produktion ab. Die Zielsetzung ist die geschlossene Bearbeitung von Geschäftsvorfällen. Dazu müssen Informationsverarbeitung und Organisation eine Gesamtheit darstellen. Diese Gesamtheit ist über eine integrierte Anwendungssoftware erreichbar, die alle Funktionsbereiche miteinander verbindet. Die Konzeption des SAP-Systems basiert auf der Idee, daß Mengen- und Wertfluß eine Einheit bilden. Alle Funktionen, z.B. Buchungen, werden direkt durchgeführt und stützen auf eine gemeinsame Datenbank. Alle Benutzer arbeiten mit gleich aktuellen Informationen. Das System kommuniziert zum Betriebssystem, Datenbanksystem und Teleprocessing-Monitor über einheitliche Schnittstellen [SAP92b].

Die Aufgaben der oben aufgeführten unterschiedlichen betrieblichen Bereiche werden von verschiedenen Systemkomponenten wahrgenommen. Es besteht eine hohe Integration zwischen den einzelnen Systemkomponenten. Das Fundament bildet die Basis-Komponente, die die Grundlage für die enge Kopplung der anderen anwendungsspezifischen Komponenten schafft. Dies vermindert die Redundanz und vereinfacht die Kommunikation zwischen den Systemkomponenten. Die Basis vereinheitlich die Bedienung des Systems und liefert die R/2-Entwicklungsumgebung als ein Instrument zur Erweiterung des Systems. Zu den Technologien der Basis-Komponente, die alle Anwendungssysteme nutzen, gehören unter anderem [SAP90]:

- ABAP/4 ist eine Programmiersprache für die Anwendungsentwicklung bei SAP und deren Kunden , neben der Programmierung in Assembler.
- Ein Dialog-Programm bietet die Möglichkeit eine bestimmte Transaktion[21] durchzuführen.

2.4.1 Ablaufsteuerung im R/2

Im SAP-System sind alle Systemkomponenten vom Betriebssystem abgeschirmt. Die Schnittstellen zum Betriebs- und Datenbanksystem sind zentral in der Basis-Komponente definiert. Dadurch erfolgt kein Ablauf im R/2-System ohne Steuerung durch die Basis. Die Aufgabenverteilung erfolgt auf verschiedene Typen von Tasks, z.B.

- erfaßt und verprobt die Dialogtask Daten und zeigt sie am Terminal an.
- führt die Verbuchungstask Datenbankänderungen durch.

Die oben genannte Aufgabenverteilung auf verschiedene Tasks ist ein Architekturmerkmal des R/" Systems. Der Erfassung von Daten und die Aufnahme in die Datenbank wird hierdurch zeitlich entkoppelt. Der Grund liegt in der Organisation der Datenbankzugriffe von vielen parallelen Benutzersitzungen, um ein gleichmäßiges Antwortverhalten zu garantieren. Durch die Auslagerung wird eine sequentielle Verbuchung erreicht [SAP90].

2.4.2 Programmiersprache ABAP/4

Die Programmiersprache ABAP/4 ist in den frühen achtziger Jahren von SAP konzipiert worden und war ursprünglich nur für die interne Verwendung gedacht. Heute wird ABAP/4 zur Anwendungsentwicklung von SAP selbst und deren Kunden eingesetzt. ABAP/4 ermöglicht eine Programmierung im betriebswirtschaftlichen Umfeld. Sie ist eine Sprache mit u.a. folgenden Eigenschaften:

- ABAP/4 erlaubt strukturierte Programmierung:
  - Daten müssen vor ihrer Verwendung deklariert sein.
  - Üblichen Kontrollstrukturen, z.B. bedingte Ausführungen und Iterationen sind in der Sprache enthalten.
  - Konzepte für die Modularisierung von Programmen sind vorhanden.

[21] Eine Transaktion umfaßt einen logisch abgeschlossenen Vorgang im R/2-System. Ein Vorgang ist z.B. die Ausführung eines Programms oder die Änderung von Kundendaten.

- ABAP/4 ist eine interpretierende Sprache:
  - Programme werden soweit wie möglich partiell ausgewertet, d.h. die syntaktische Korrektheit ist garantiert. Es erfolgt keine Berechnung von Markendefinitionen bei Sprüngen.
  - Das Ergebnis dieser semantischen Analyse ist fest eingebaut und durch die Aufrufstruktur der Funktionen repräsentiert.
- Sprachumfang von ABAP/4 ist auf den Einsatz im Rahmen betrieblicher Informationssysteme zugeschnitten:
  - Die Manipulation der Inhalte externer Dateien sowie der Zugriff auf Datenbanken wird unterstützt.
  - Es können Bildschirmeingaben und -ausgaben realisiert werden.
  - ABAP/4 ist ereignisorientiert.
  - ABAP/4 enthält deklarative, operationale, steuernde Sprach- und Zeitpunktelemente.

Der Sprachumfang von ABAP/4 beträgt etwa 300 Schlüsselworte. Neben den Schlüsselworten beinhaltet ein ABAP/4-Programmtext Operanden, in Form von Konstanten und Variablen. ABAP/4 ist eine formatfreie Sprache; Leerzeichen zwischen den Worten oder Zeilenvorschübe tragen keine Sematik. Ein ABAP/4-Programm besteht aus einzelnen Sätzen, die mit einem Punkt abgeschlossen werden. Ein typischer Satzaufbau ist ein Wechsel von Schlüsselworten und Operanden. Sätze, die mit demselben Schlüsselwort beginnen, können zu einem Kettensatz zusammengefaßt werden [SAP90].

SAP unterscheidet zwei Arten von ABAP/4-Programmen:

- ABAP/4-Reports

  lesen und analysieren Daten aus Datenbanktabellen, ohne die Datenbank zu verändern. Das Ergebnis eines Reports hat die Form einer Liste, die auf dem Bildschirm oder auf einem Drucker ausgegeben wird. Ein ABAP/4-Report ist ein Programm, das einen Report auf Basis von einer oder mehreren Datenbanktabellen erzeugt. Reports werden von logischen Datenbanken[22] unterstützt.

- Dialogprogramme

  sind als Modulpools[23], die Dialogmodule enthalten, organisiert. Jedes Dynpro[24], das aus einer Bildschirmmaske und dessen Ablauflogik besteht, basiert

[22]Eine logische Datenbank ist eine Methode, mit deren Hilfe ein ABAP/4-Report Daten lesen und verarbeiten kann. Die logische Datenbank stellt ein Selektionsbild zur Verfügung.

[23]Zu den wichtigsten Bereichen eines Arbeitsgebietes gehört der Modulpool. Er enthält alle für diesem Bereich relevanten Programmmodule. Auf den Modulpool können alle Entwickler des Bereichs zugreifen.

[24]Dynpro heißt Dynamisches Programm. Es ist ein SAP-spezischer Begriff für eine Bildschirmmaske und dessen Ablaufsteuerung.

auf genau einem ABAP/4-Dialogprogramm. Die Ablauflogik eines Dynpros enthält Aufrufe an die ABAP/4-Dialogbausteine.

Selektionen und logische Datenbank

Besonderheiten von Reports sind Selektionen und logische Datenbanken. Selektionen sind Eingabemöglichkeiten von Parametern und können aus einzelnen Werten, u.a. auch aus Intervallen, Vergleichen, gesuchten Texten und mehreren alternativen Bedingungen bestehen. Die logische Datenbank repräsentiert eine spezielle Sichtweise auf die Daten, die in einer oder mehreren physischen Datenbanken gespeichert sind. Die logische Datenbank ist ein spezielles Leseprogramm für die Datenbeschaffung und versorgt einen Report mit einer hierarchisch organisierten Menge von Tabelleneinträgen. Die Reihenfolge, in der die Daten geliefert werden, hängt vom Aufbau der logischen Datenbank ab. Nach jedem Lesen eines Datensatzes stellt ihn die logische Datenbank in den vom Report dafür definierten Arbeitsbereich. Dort kann der Datensatz weiter bearbeitet werden [SAP90].

Im SAP-System sind Stammdaten[25] in mehrstufigen Datenstrukturen gespeichert. Ein Stammsatz setzt sich aus mehreren Segmenten und Tabellen zusammen. Ein Segment oder eine Tabelle ist ein Art Formular, das die logisch zusammengehörigen Informationen zu einem Objekt in Form von Feldern enthält. Jedes Feld eines Segments oder einer Tabelle repräsentiert eine Eigenschaft des Objekts. Die Beziehungen der einzelnen Objekte zueinander werden durch die Hierarchie des Segmente oder Tabellen dargestellt. Die Hierarchie stellt eine logische Sicht auf die Daten dar. Diese Sichtweise wird im SAP-System als logische Datenbank bezeichnet [SAP95].

Verarbeitungslogik eines Reports

Die Programmiersprache ABAP/4 ist eine ereignisorientierte Sprache. Die Anweisungen eines Reports werden nicht sequentiell in der Reihenfolge ihres Auftretens im Report abgearbeitet, sondern durch Zeitpunktsprachelemente. Sie fassen Anweisungen zu Verarbeitungsblöcken zusammen. Zeitpunktsprachelemente werden durch sogenannte Ereignisschlüsselworte abgebildet. Innerhalb eines Verarbeitungsblocks werden die Anweisungen sequentiell ausgeführt oder folgen einer Steuerung durch entsprechende Steuerungsschlüsselworte. Dieses wird interne Steuerung genannt. Ein Ereignisschlüsselwort leitet stets einen neuen Verarbeitungsblock ein, der automatisch durch ein neues Ereignis oder durch Definition eines Unterprogramms abgeschlossen wird.

ABAP/4-Programme werden nie als Ganzes, sondern nur in Teilen in Abhängigkeit der Ereignisse ausgeführt. Die Reihenfolge der einzelnen Verarbeitungsblöcke hängt vom Eintreten der externen Ereignisse ab. Dieses ist die externe Steuerung. Ereignisse werden entweder durch andere ABAP/4-Programme oder durch interaktive Anwendereingaben erzeugt. Andere ABAP/4-Programme sind z.B. System- oder

[25] Stammdaten sind Daten, die über einen längeren Zeitraum unverändert bleiben. Sie beschreiben im weitesten Sinne Personen und Gegenstände, z.B. Kunden-, Lieferanten- und Materialdaten.

Anwendungsprogramme. Das Eintreten von Ereignissen ist auch an die Selektion und die logische Datenbank gekoppelt.

ABAP/4-Prozessor und Report

Um einen Report handelt es sich, wenn in ihm logische Datenbanken verwendet werden. Eine Sammlung von Verarbeitungsblöcken, die beim Eintreten bestimmter Ereignisse ausgeführt werden, wäre ein Beispiel. Im SAP-System hängen Report und logische Datenbanken, über die, die im Report benötigte Daten eingelesen werden können, eng zusammen. Im Report selbst wird nur angegeben, wie die deklarierten Daten verarbeitet und weiter aufbereitet werden sollen. Dieses wird durch ein enges Zusammenspiel von ABAP/4-Prozessor und Report erreicht. Das Navigieren über die Datenbank und die anschließende Weiterverarbeitung übernimmt der ABAP/4-Prozessor, gesteuert durch die aktuelle Selektion und die logische Datenbank. Im Report ist nicht definiert, wie Informationen aus der Datenbank selektiert werden, sondern wie sie weiter bearbeitet werden.

Der ABAP/4-Prozessor stellt das Rahmenprogramm dar. Der Report beinhaltet die Verarbeitungsvorschriften, die beim Eintreten bestimmter Ereignisse durchgeführt werden. Das Zusammenspiel wird durch Ereignisschlüsselworte im Report erreicht, die die einzigen zulässigen Verbindungen zwischen ABAP/4-Prozessor und Report darstellen.

Schlüsselwort-Kategorien

Es werden vier Kategorien von ABAP/4-Schlüsselworten unterschieden:

1. Deklarative Schlüsselworte

   deklarieren die Datenobjekte, die global im Report angesprochen werden können, auch über verschiedene Ereignisse hinweg. Eine Ausnahme sind lokale Datenobjekte in Unterprogrammen und Funktionsbausteinen[26].

2. Ereignisschlüsselworte

   geben Verarbeitungszeitpunkte an. Dabei werden zwei Gruppen von Ereignissen unterschieden: Ereignisse während des Durchlaufs der logischen Datenbank und Ereignisse während der Erstellung einer Bildschirmausgabe oder Liste.

3. Steuerungsschlüsselworte

   formen aus den elementaren Operationen Kontrollstrukturen. Sie steuern den Ablauf innerhalb eines Verarbeitungszeitpunktes.

4. Operationale Schlüsselworte

[26] Funktionsbaustein sind in ABAP/4 spezielle Unterprogramme mit einer klar abgegrenzten Schnittstelle. Für die Validation existiert eine eigene Testumgebung und die Verwaltung erfolgt in einer eigenen Funktionsbibliothek.

führen mit den deklarierten Daten bei bestimmten Ereignissen unter gewissen Bedingungen bestimmte Verarbeitungen durch.

Ereignisse

Die einem Ereignis zugeordneten, steuernden und operationalen Schlüsselworte sowie Ausführungen von Unterprogrammen beziehen sich immer auf den aktuellen Verarbeitungszeitpunkt, dem Ereignis.

Die Verarbeitung eines Reports beginnt mit dem Ereignis `INITIALIZATION`. Hier können Vorschlagswerte für Selektionen angegeben werden. Es folgt der Zeitpunkt `START-OF-SELECTION`. Er wird zu Beginn der Datenselektion, vor dem Durchlaufen der logischen Datenbank ausgeführt. Die eigentliche Verarbeitung, das Einlesen der Daten erfolgt mit dem Ereignis `GET <tabelle>`. Alle Felder aus der Tabelle können in der nachfolgenden Verarbeitung angesprochen werden. Das Ereignisschlüsselwort mit einem Zusatz `GET <tabelle> LATE` wird dann ausgeführt, wenn im hierarchischen Durchlauf der Tabelleneinträge, alle untergeordneten Tabellen gelesen und verarbeitet sind. Im anschließenden Ereignis `END-OF-SELECTION` werden die gelesenen Daten weiterverarbeitet.

Während der Anzeige können durch interaktive Eingriffe weitere Verarbeitungen gestartet werden. Diese Eingriffsmöglichkeiten sind speziell im Programm eingeräumt. Dazu stehen, je nach beschrittenem Weg, drei weitere Ereignisse `AT PF<nn>`, `AT LINE-SELECTION` und `AT USER-COMMAND` zur Verfügung. Der `AT LINE-SELECTION` Zeitpunkt wird prozessiert, wenn in der Anzeige eine Position ausgewählt und durch eine Taste bestätigt wurde. Über die Funktionstasten können mit `AT PF<nn>` aus der Anzeige heraus weitere Verarbeitungen ausgeführt werden. Die dritte Variante ist, durch einen eingegebenen Funktionscode die Verarbeitungen für das Ereignis `AT USER-COMMAND` anzustoßen.

Die Ereignisse `TOP-OF-PAGE` und `END-OF-PAGE` gestalten die Kopf- und Fußzeilen einer Seite. Sie werden jeweils zu Beginn oder am Ende einer Seite prozessiert.

Jede Anweisung in einem ABAP/4Programm ist Bestandteil eines Verarbeitungsblocks oder eines Unterprogramms. Es ist nicht zwingend, daß in einem Report eins der angeführten Ereignisse vorkommen muß. In diesem Fall bilden alle Anweisungen den Verarbeitungsblock `START-OF-SELECTION`, sofern sie nicht im Deklarationsblock eines Unterprogramms sind. Dieses gilt auch für alle Anweisungen, die nicht nach einem Ereignis stehen. Sie sind automatisch Bestandteil des Standard-Ereignisses `START-OF-SELECTION`.

Steuernde Schlüsselworte

In ABAP/4 existieren eine Reihe von Kontrollstrukturen, die innerhalb des Verarbeitungszeitpunktes, Einfluß auf den Ablauf des Verarbeitungsblocks nehmen.

Kontrollstrukturen sind bedingte Ausführungen, Iterationen und Gruppenwechsel[27]. Weitere Einflußmöglichkeiten beim Durchlaufen von logischen Datenbanken oder Iterationen können durch unstrukturierte Konstrukte vorgenommen werden.

Eine einfache bedingte Ausführung ist die `IF`-Abfrage. Mit der `CASE`-Konstruktion stehen mehrere Anweisungen zur Auswahl. Einfache Iterationen in ABAP/4 sind die `DO`-, `FOR`- und `WHILE`-Schleife. Eine interne Tabelle[28] wird mit einer `LOOP`-Schleife verarbeitet. Innerhalb dieser Schleife stehen die Anweisungen `AT FIRST`, `AT NEW`, `AT END`, `AT LAST` und `ON CHANGE OF` für Gruppenwechsel zur Verfügung. Vorzeitige Abbrüche von Datenbank- und Schleifendurchläufen erfolgen durch die Konstrukte `CHECK`, `EXIT`, `STOP`, `REJECT` und `LEAVE`.

2.4.3 Modularisierung in ABAP/4

Ein Modul ist eine abgeschlossene Folge von Sprachelementen mit einer klar definierten Schnittstelle. Jedes Modul hat einen eindeutigen Namen [Yourdon79]. [Pomberger93] gibt zur Definition eines Moduls die folgenden vier Eigenschaften an:

1. In einem Modul sind Operationen und Daten zusammengefaßt, um eine abgeschlossene Aufgabe zu realisieren.
2. Zur Außenwelt kommuniziert ein Modul nur über eindeutig spezifizierte Schnittstellen.
3. Um ein Modul in ein Programmsystem zu integrieren darf keine Kenntnis über das innere Arbeiten des Moduls erforderlich sein.
4. Die Korrektheit eines Moduls muß ohne Kenntnis seiner Eingliederung in ein Programmsystem nachprüfbar sein.

In ABAP/4 existieren zwei Möglichkeiten den Programmtext in Unterroutinen auszulagern: Unterprogramme und Funktionsbausteine. Unterprogramme sind nur unter den Voraussetzungen Module, solange die vier oben aufgeführten Eigenschaften eingehalten werden. Dies ist in ABAP/4-Programmen nicht immer der Fall. Unterprogramme benötigen z.B. keine definierte Schnittstelle, da sie direkt auf die deklarierten Variabeln im Hauptprogramm zugreifen. Dagegen können Funktionsbausteine ohne Einschränkung als Module angesehen werden. Sie sind abgeschlossen, haben eine klar abgegrenzte Schnittstelle und besitzen eine eigene Testumgebung.

Die Modularisierung ist ein Prozeß, bei dem die Aufgaben eines Programms in abgeschlossene Teilaufgaben, die von Modulen wahrgenommen werden, zerlegt wird. Um Fehler bei der Modularisierung zu vermeiden, sollten in ABAP/4 die Module durch

[27] Bei einem Gruppenwechsel werden Gruppen für eine Auswertung abgegrenzt. Ein Gruppenwechsel hängt von der Änderung des Schlüsselfeldes ab. Ist eine Gruppe abgearbeitet beginnt eine neue, es findet ein Gruppenwechsel statt.

[28] Interne Tabellen werden wie Variablen in einem Report deklariert. Sie sind auch nur in dem Report gültig.

Funktionsbausteine beschrieben werden. Die Verwendung von Unterprogrammen kann zu Problemen bei der Integration in das Programm und bei der Nachprüfbarkeit der Korrektheit des Moduls führen. Der Grund ist, daß in einem Unterprogramm global im Programm deklarierte Variable verwendet werden können. Diese beiden Hauptprobleme sind durch die Eigenschaften der Funktionsbausteine ausgeschlossen.

2.4.4 ABAP/4-Beispiel-Programme

Die ABAP/4-Beispiel-Programme sind in sequentielle (S), nicht-sequentielle (N) und strukturierte (U) Programme sowie in modularisierte (M) und nicht-modularisierte (UM) Programme unterteilt. Jedes ABAP/4-Programm hat eine eigene Bezeichnung erhalten, die diese Einteilung widerspiegelt. Ein nicht-sequentielles und unmodularisiertes Programm trägt z.B. die Bezeichnung: N-UM-003.

Die Einordnung wurde gewählt, um die Charakteristiken der beiden Meßverfahren von [McCabe76] und [Halstead77] herausstellen zu können: Die zyklomatische Komplexität beschreibt die strukturelle Komplexität. Halstead geht auf die berechende Komplexität ein.

1. Der unterschiedliche Fokus der Verfahren zwischen sequentiellen und nicht-sequentiellen Programmen wird hervorgehoben. Dies beschreibt [Weyuker88] mit ihrer zweiten Eigenschaft für Komplexitätsmessungen: eine Messung muß sensitiv genug sein, um Programme nicht nur in wenige Komplexitätsklassen einzuteilen. Halsteads Messung kann die zweite Eigenschaft von Weyuker erfüllen. Die zyklomatische Komplexität von McCabe entspricht nicht diesen Anforderungen.

2. Die Auswirkung von unstrukturierten Konstrukten auf die Komplexität wird charakterisiert. McCabe hebt die Messung der unstrukturierten Konstrukte hervor. Halstead nimmt keine Unterscheidung zwischen sequentiellen und (un)strukturellen Konstrukten vor.

3. Der Effekt einer Modularisierung der Programme auf die Komplexität wird dargestellt. Mit Hilfe der zyklomatischen Zahl können klare Programmstrukturen entworfen und Programme modularisiert werden.

2.5 Statistische Analyseverfahren

Verschiedene statistische Analyseverfahren werden zur Überprüfung der Ergebnisse der Messungen eingesetzt. Zu untersuchen ist, ob die Meßverfahren von McCabe und Halstead die Komplexität von ABAP/4-Programmen charakterisieren. Mittels der Korrelations- und Regressionsanalyse erfolgt die Validation der Zusammenhänge zwischen einer intuitiven Einschätzung und der Meßergebnissen anhand einer ABAP/4-Beispiel-Auswahl. Eine kurze Einführung in diese Analyseverfahren ist im Kapitel 2.5.1 zu finden. Die Messung nach Halstead soll mit Hilfe der Faktorenanalyse vereinfacht werden. Es wird ein Faktor gesucht, der die Beziehungen der vier Basisgrößen beschreibt. Die wichtigsten Aspekte der Faktorenanalyse werden im Kapitel 2.5.2 behandelt.

2.5.1 Korrelations- und Regressionanalyse

Die Korrelations- und Regressionanalyse sind flexible und häufig eingesetzte statistische Analyseverfahren. Sie untersuchen stochastische Zusammenhänge zwischen Zufallsvariablen anhand einer Stichprobe. Es sollen Beziehungen erkannt und erklärt werden. Mittels der Korrelationsanalyse werden Abhängigkeitsmaße und Vertrauensbereiche geschätzt und Hypothesen geprüft. Ein wichtiges Abhängigkeitsmaß ist der Korrelationskoeffizient ρ. Dieser wird druch den Stichprobenkorrelationskoeffizienten r geschätzt. r ist ein Maß für die Stärke des stochastischen Zusammenhangs. Es erfolgt eine Unterscheidung u.a. in den einfachen, multiplen und partiellen Korrelationskoeffizienten [Sachs74].

Durch die Regressionsanalyse wird einer beobachteten Punktwolke eine Regressionsgleichung angepaßt. In der Gleichung $y = \alpha + \beta \cdot x$ ist die abhängige Zufallsvariable y die Zielgröße und die vorgegebene Variable x die Einflußgröße. Die Parameter α und β werden aus den Stichprobenwerten geschätzt.

Korrelationsanalyse

Mit Hilfe der Korrelationsrechnung kann geprüft werden, ob zwischen zwei Merkmalen x und y ein Zusammenhamg besteht oder nicht. Es wird revidiert, ob bei zunehmenden x-Werten die y-Werte in statistisch gesicherten Ausmaß ansteigen oder zurückgehen. Dieses Abhängigkeitsmaß, der Korrelationskoeffizient r kann einen Wert zwischen -1 und $+1$ ausweisen:

- Positiver Korrelationskoeffizient: bei steigenden x-Werten steigen auch gleichzeitig die y-Werte

- Negativer Korrelationskoeffizient: bei zunehmenden x-Werten gehen die y-Werte zurück

Je näher der Korrelationskoeffizient r bei -1 oder $+1$ liegt, desto intensiver ist der Zusammenhang zwischen den beiden Merkmalen. Bei einem Korrelationskoeffizient $r = 0$ besteht zwischen den beidem Größen kein Zusammenhang. Aus der Höhe des Koeffizienten kann die statistische Sicherheit abgelesen werden. Eine einfache Korrelation bezieht sich nur auf die Merkmale x, y und eine partielle Korrelation schaltet Auswirkungen eines weiteren Merkmals oder mehrerer anderer Merkmale auf den Zusammenhang zwischen x und y-Werte aus. Die multiple Korrelation mißt die Abhängigkeit von y von mehreren Merkmalen x_i [Renner81].

Einfache lineare Korrelation

Die Anzahl der Proben beträgt n. Für die Freiheitsgrade FG gilt: $FG = n - 2$. Der Korrelationskoeffizient wird beschrieben druch:

$$r = \frac{n \cdot \sum xy - \sum x \cdot \sum y}{\sqrt{\mid n \cdot \sum x^2 - (\sum x)^2 \mid \cdot \mid n \cdot \sum y^2 - (\sum y)^2 \mid}}$$

Der Korrelationskoeffizient r kann auch berechnet werden, wenn es sich bei den x-Werten um eine unabhängige Größe handelt, die sich in logisch numerischer Reihefolge bringen läßt.

Partielle Korrelation

Die Berechnung einer partiellen Korrelation ist notwendig, wenn mehr als zwei Merkmale in einem Zusammenhang zueinander stehen. Es kann nur dann ein stichhaltiger Korrelationskoeffizient r für die Beziehung zwischen jeweils zwei Merkmalen erhalten werden, wenn die überlagernden Beziehungen zum dritten oder zu weiteren Merkmalen ausgeschaltet sind oder wenn die weiteren Merkmale konstant gehalten werden. Die Berechnung des partiellen Korrelationskoeffizienten ermöglicht diese rechnerische Konstanthaltung.

Es wird die partielle Korrelation behandelt, bei der ein Merkmal konstant gehalten wird. Bei drei Merkmalen, die miteinander in Beziehung stehen, können drei einfache Korrelationskoeffizienten berechnet werden: r_{ab}, r_{ac}, r_{bc}.

Diese einfachen Korrelationskoeffizienten r bilden die Voraussetzung für die Berechnung des partiellen Korrelationskoeffizienten $r_{ab \cdot c}$, d.h. des Korrelationskoeffizienten für die Beziehung zwischen den Merkmalen a und b, wobei das Merkmal c rechnerisch konstant gehalten wird:

$$r_{ab \cdot c} = \frac{r_{ab} - r_{ac} \cdot r_{bc}}{\sqrt{(1 - r_{ac}^2) \cdot (1 - r_{bc}^2)}}$$

$$r_{ac \cdot b} = \frac{r_{ac} - r_{ab} \cdot r_{bc}}{\sqrt{(1 - r_{ab}^2) \cdot (1 - r_{bc}^2)}}$$

$$r_{bc \cdot a} = \frac{r_{bc} - r_{ab} \cdot r_{ac}}{\sqrt{(1 - r_{ab}^2) \cdot (1 - r_{ac}^2)}}$$

Wenn bei der Überprüfung eines Zusammenhangs zwischen zwei Merkmalen, Beziehungen zu einem anderen Merkmal oder mehreren anderen Merkmalen bekannt sind, dann ist Berechnung der partiellen Korrelationkoeffizienten anzuwenden, um nicht zu unrichtigen Ergebnissen zu kommen.

Multiple Korrelation

Lautet die Frage, in welcher Weise ein Merkmal zugleich von anderen Merkmalen abhängt, dann wird die multiple Korrelation eingesetzt. Die drei einfachen Korrelationskoeffizienten sind gegeben. Der multiple Korrelationskoeffizient wird berechnet durch:

$$r_{a \cdot bc} = \sqrt{\frac{r_{ab}^2 + r_{ac}^2 - 2 \cdot r_{ab} \cdot r_{ac} \cdot r_{bc}}{1 - r_{bc}^2}}$$

Regressionsanalyse

Eine lineare Korrelation für den Zusammenhang zwischen den Merkmalen x und y kann durch eine Regressionsgerade dargestellt werden, die durch folgende Gleichung definiert ist:

$$y = \eta \cdot x + \alpha$$

Die Größen β und α werden wie folgt berechnet:

$$\beta = \frac{n \cdot \sum xy - \sum x \cdot \sum y}{n \cdot \sum x^2 - (\sum x)^2}$$

$$\alpha = \frac{\sum y - \beta \cdot \sum x}{n}$$

2.5.2 Faktorenanalyse

Die Hauptziele der Faktorenanalyse sind [Ueberla68]:

- die Ableitung hypothetischer Größen, sogenannte Faktoren aus einer Menge beobachteter Variablen.
- Die Faktoren sollten möglichst einfach sein und die Beobachtungen genau beschreiben und erklären.
- Die ermittelten Größen bzw. Faktoren sollten hinsichtlich ihrer Zahl möglichst klein und hinsichtlich ihres strukturellen Aufbaus und Zusammenhangs möglichst einfach sein.

Die Faktorenanalyse stellt die Frage nach der einfachsten Struktur, um vorliegende Daten genügend genau zu reproduzieren und zu erklären. Die beobachteten Korrelationen zwischen Variablen werden als Ausdruck einer nicht beobachteten Größe, eines Faktor angesehen, von dem aus Korrelationen in einfacher Weise berechnet werden können.

Eigenschaften der Faktoren sind:

- Faktoren stehen hinter beobachteten Größen und sind direkt nicht zugänglich.
- Faktoren sind nichtmeßbare Einflußgrößen, die im Hintergrund stehen und erst durch die Analyse ermittelt werden.
- Faktoren sind hypothetisch, eine aus den Daten abgeleitete Konstruktion, eine mathematische Größe, die beobachteten Korrelationen gerecht werden, indem sie sich aus Faktoren ableiten lassen.

Die Faktorenanalyse ermittelt hypothetische Faktoren. Außerdem macht sie eine differenzierte Hypothese über die Struktur des Zueinander der Variablen und Faktoren möglich, ohne vorher eine bestimmte Struktur anzunehmen oder kennen zu müssen. Das Ergebnis der Faktorenanalyse wird von der Anlage der Untersuchung bestimmt.

Die Faktorenanalyse wird in Bereichen angewendet, in denen die Manipulation der beobachteten Daten nicht möglich ist. Die Analyse behandelt folgende Fragestellungen:

- Was ist die einfachste lineare Hypothese der Struktur, die hinter der Vielfalt miteinander korrelierender Variablen steht?
- Wie viele und welche hypothetische Größe oder Faktoren werden benötigt, um die beobachteten Beziehungen zwischen den Variablen möglichst genau zu reproduzieren und zu erklären?
- Wie lassen sich große Datenmengen auf möglichst einfache Konzepte reduzieren?

Korrelationsmatrix und Faktor

Die Voraussetzung für die Faktorenanalyse ist, daß mehrere gemessene Variable, eng zusammenhängen und stark miteinander korrelieren. Es kann gefolgert werden, daß die Variablen weitgehend dasselbe besagen. Zwei Annahmen bestehen:

1. Die Variablen bestimmen sich einander wechselseitig.
2. Eine dritte Größe, die sich nicht direkt messen läßt, prägt sich in den Variablen aus.

Das faktorenanalytische Modell geht von der 2. Annahme der Folgerung aus: Das Meßbare ist nur eine Erscheinungsform von Größen, die im Hintergrund stehen und die nicht direkt gemessen werden können.

Das Ziel ist die Suche, ob aus den beobachteten Variablen sich eine Größe isolieren läßt, ein sogenannter Faktor, der die beobachteten Zusammenhänge klärt. Der Faktor ist die mathematische Größe, die aus den Beobachtungen abgeleitet ist.

Es wird eine Korrelationsmatrix aus m beobachteten Objekten mit n Eigenschaften aufgestellt. Es besteht eine Korrelation zwischen je zwei Eigenschaften eines Objektes. Die Faktorenanalyse stellt die Frage, ob es eine Größe gibt, ein Faktor, der in Verbindung mit geeigneten Rechenregeln die beobachteten Korrelationen möglichst genau reproduziert. Dieser Faktor mit den verbundenen Rechenregeln ist zunächst hypothetisch, da er nur erlaub die Korrelationsmatrix zu reproduzieren. Die Zahlenwerte des Faktors erlauben die rechnerisch-formale Erklärung der beobachteten Korrelationen.

Ablauf der Faktorenanalyse

Voraussetzungen der Faktorenanalyse sind:

- Die Beziehung zwischen zwei Variablen wird durch einen Korrelationskoeffizient ausgedrückt.
- Die Anordnung aller Korrelationskoeffizienten zwischen den Variablen erfolgt in einer Korrelationsmatrix.
- In dieser Matrix sind alle relevanten Informationen über die Beziehungen der Variablen einschließlich störender Einflüße enthalten.
- Die Faktorenanalyse geht von der Korrelationsmatrix aus.
- Aus der Analyse der Korrelationsmatrix erhält man hypothetische Größen, sogenannte Faktoren, die bestimmte Beziehungen zu Variablen ausweisen.

Aus der Datenmatrix $Y = (y_{ij})$ wird eine standardisierte Ausgangsmatrix Z erzeugt. Die Ausgangsmatrix Z ist die Grundlage für die Korrelationsmatrix R. Sie entählt die Beziehungen zwischen den Variablen, die durch Korrelationskoeffizienten ausgedrückt werden. Die Anordnung alle Korrelationskoeffizienten zwischen den Variablen erfolgt in einer Korrelatonsmatrix $R = (r_{ik})$.

Bei dem Kommunalitätensproblem werden neue Elemente in die Diagonale der Matrix R eingesetzt, sogenannte Kommunalitäten. Die entstehende Matrix heißt reduzierte Matrix R_h. Die Schätzverfahren für die Kommunalitäten sind u.a. der höchste Korrelationskoeffizient, das Quadrat des multiplen Korrelationskoeffizienten und die Iteration.

Die reduzierte Korrelationsmatrix R_h wird in Faktoren aufgelöst, die sie möglichst genau zu reproduzieren gestatten. Ist ein einziger Faktor nicht ausreichend, um die Korrelationsmatrix R_h genügend genau zu reproduzieren, wird auf Basis der Residualmatrix R_1[29] ein weiterer Faktor extrahiert. Dieses Verfahren wiederholt sich iterativ solange, bis die Korrelationsmatrix möglichst genau reproduziert ist. Methodem sind u.a. die Hauptkomponenten- und Zentroidmethode.

Das Ergebnis des Faktorenproblems ist das Faktorenmuster mit den Faktorenladungen A. Eine anschließende Rotation wird nötig, weil die Faktorenextraktion keine eindeutiges Ergebnis bringt, sondern unendlich viele äquivalente Lösungen.

Aus dem rotierten Faktorenmuster V werden die Faktorenwerte geschätzt. Die Faktorenwerte sind Meßwerte eines bestimmten Objektes in bezug auf einen bestimmten Faktor. Die Faktorenwerte gehen einen Schritt weiter als die erhaltenen Faktoren, die die Struktur hinter den Varibalen beschreiben. Jedes Objekt hat einen Meßwert für jede Variable und es können jedem Objekt Werte für jeden Faktor zugeordnet werden. Allgemein sind due Faktorenladungen die Merkmalsparameter eines Modells und die Faktorenwerte sind Parameter, die den einzelnen Objekten zugeordnet werden.

[29] Sie entsteht aus der Differenz der reduzierten R_h und reproduzierten Matrix R^+