
Structured Design

Datenstrukturen und Algorithmen entwerfen und anwenden

Inhaltsverzeichnis

1	Edward Yourdon 1944-2016.....	2
2	Einleitung.....	3
2.1	Ziel des strukturierten Entwurfs.....	3
3	Abstraktion	4
3.1	Funktionale Abstraktion	4
3.2	Datenabstraktion.....	5
4	Modularisierung	6
4.1	Eigenschaften eines Moduls.....	6
4.2	Äussere und innere Sicht des Moduls	7
4.3	Ziele der Modularisierung	7
5	Modellnotationen des "Structure-Chart"	7
5.1	Namensgebung und Darstellung der Module	7
5.2	Aufruf eines Moduls, eines Speichers oder eines externen Gerätes	8
5.3	Parameterübergabe	8
5.4	Eigenschaften des Structure-Charts	9
6	Modul-Spezifikation	10
6.1	Modulkopf	10
6.2	Beispiel einer Modulspezifikation	12
7	Kopplung versus Kohäsion.....	13
7.1	Kopplung.....	13
7.2	Zusammenhalt (Kohäsion).....	20
8	Zusammenfassung.....	25

1 Edward Yourdon 1944-2016



He is one of the ten most influential men and women in the software field, according to the December 1999 issue of *Crosstalk: The Journal of Defence Software Engineering*; in June 1997, he was inducted into the Computer Hall of Fame, along with such notables as Charles Babbage, Seymour Cray, James Martin, Grace Hopper, Gerald Weinberg, and Bill Gates. An internationally recognized consultant and lecturer, he is the author/ co-author of more than two dozen books.

Ed is widely known as the lead developer of the structured analysis/design methods of the 1970s, and was a co-developer of the Yourdon/Whitehead method of object-oriented analysis/design and the popular Coad/Yourdon OO methodology in the early 1990s. He has worked in the computer industry for 38 years, beginning when Digital Equipment Corporation innocently risked the downfall of Western civilization by hiring him as a starry-eyed undergraduate student in 1964 to write the FORTRAN math library for the PDP-5 and the assembler for the popular PDP-8 minicomputer. During his career, he has worked on over 25 different mainframe computers, and was involved in a number of pioneering computer technologies such as time-sharing operating systems and virtual memory systems.

After stints with DEC and GE, a small consulting firm and a few years as an independent consultant, Ed founded his own consulting firm, YOURDON Inc., on April Fool's Day of 1974, in order to provide consulting and educational services in state-of-the-art software engineering technology and project management techniques. Over the next 12 years, the company grew to a staff of over 150 people, with offices throughout North America and Europe; as CEO of the company, he oversaw an operation that trained over 250,000 people around the world in analysis/design methodologies that ensured delivery of high-quality systems on time, and under budget. YOURDON Inc. was sold in 1986 and eventually became part of CGI, the French software company that is now part of IBM. The publishing division, YOURDON Press (now part of Prentice Hall), has produced over 150 technical computer books on a wide range of software engineering topics; many of these "classics" are used as standard university computer science textbooks.

Ed is the author of over 550 technical articles. He has also authored or co-authored 26 computer books since 1967, including a mediocre novel on computer crime and a damn good book for the general public entitled *Nations At Risk* (which garnered favourable reviews, but not much more than a few loud yawns from the aforementioned general public). Several of his books have been translated into Japanese, Russian, Chinese, Spanish, Portuguese, Dutch, French, German, Polish, and other languages; and his articles have appeared in virtually all of the major computer journals. He is a keynote speaker at major computer conferences around the world, where he is regularly criticized for his utter lack of humour, and he served as the conference Chairman for Digital Consulting's CASE WORLD and SOFTWARE WORLD conferences from 1990 through 1995, as well as the Cutter Summit conference from 1997 through 2000.

Ed Yourdon received a B.S. in Applied Mathematics from MIT; he subsequently carried out graduate work at MIT and at the Polytechnic Institute of New York, an experience which convinced him that politics are far nastier in academia than in private industry. He has been appointed an Honorary Professor of Information Technology at Universidad CAECE in Buenos Aires, Argentina, and is a Faculty Fellow in the Information Systems Research Centre at the University of North Texas. He has lectured at MIT, Harvard, UCLA, Berkeley, Iona College, New Mexico Highlands University, Georgia Tec, and other universities around the world.

2 Einleitung

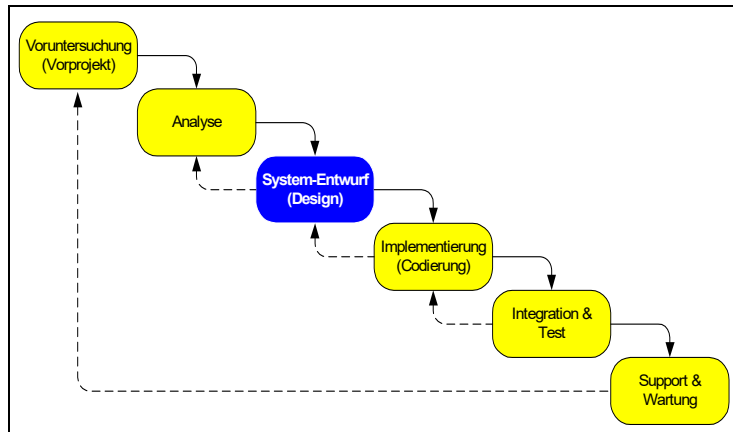


Abbildung 1: Wasserfallmodell

Unter dem Begriff „Structured Design“ versteht man im deutschen Sprachraum die Bezeichnung „Strukturierter Entwurf“.

In Anlehnung an den englischen Begriff hat sich die Abkürzung „SD“ durchgesetzt. Wie der deutsche Name schon sagt, handelt es sich dabei um eine Entwurfsmethode die in der System-Entwurfs-Phase (Design) zur Anwendung kommt.

2.1 Ziel des strukturierten Entwurfs

<p>Merke:</p> <p>!</p>	<p>Ziel der SD-Methode ist es, eine Softwarearchitektur zu erstellen, die aus hierarchisch angeordneten funktionalen Modulen besteht.</p>
--------------------------------------	--

Um den strukturierten Entwurf zu beschreiben, werden wie auch in der Designphase Diagramme verwendet. Diese heissen diesmal „Strukturdiagramme“ (Structure-Chart) und werden durch Beschreibungen in Form von „Modulspezifikationen“ ergänzt. Das heisst, dass jeder einzelne Block in der nebenstehenden Zeichnung durch eine Spezifikation beschrieben wird.

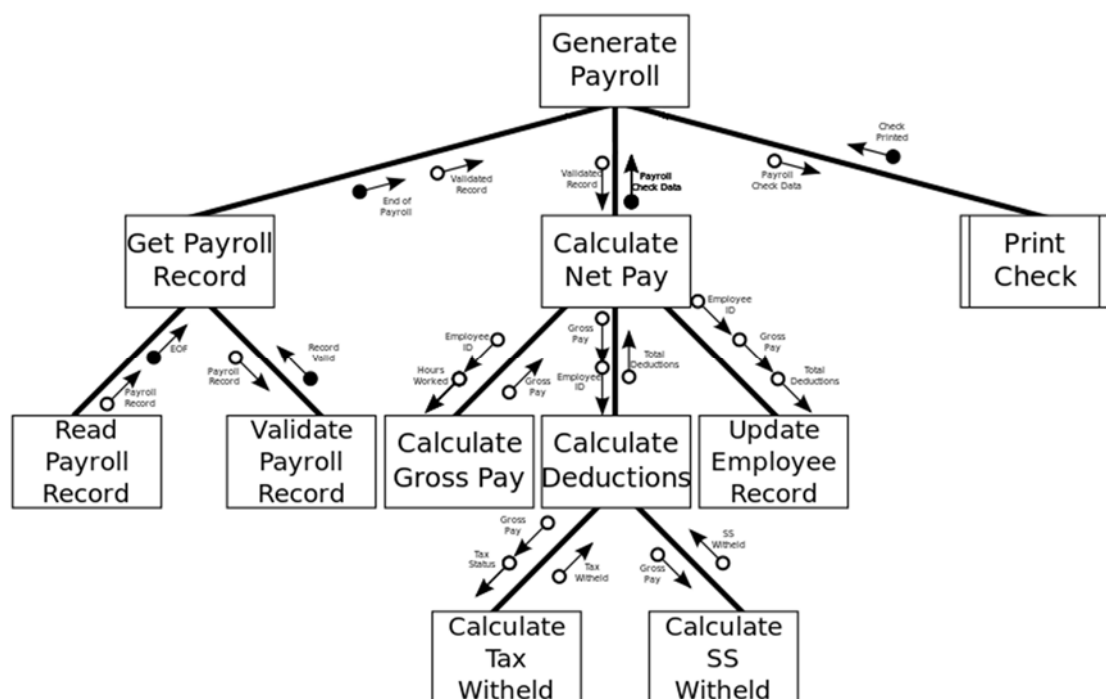



Abbildung 2: Beispiel eines Structure-Charts

3 Abstraktion


Was auch immer auf dem Rechner implementiert wird ist eine **Abstraktion realer Dinge**, also ein **Modell**. Im Vordergrund stehen dabei **Abstraktionen von Aktionen** und **Abstraktionen von Objekten**.

Abstraktion bedeutet Verallgemeinerung, das Absehen vom Besonderen. Beim Abstrahieren versucht man, im Hinblick auf die Aufgabe wichtige Merkmale von Aktionen oder von Dingen herauszugreifen und unwichtige zu ignorieren. Damit ist es überhaupt erst möglich, Begriffe zur Beschreibung gemeinsamer Beziehungen zwischen Dingen zu entwickeln. Abstraktion ist eigentlich ein Synonym für **Modellbildung**. Die wesentlichen Teile der Realität werden herausgegriffen und alles für die Aufgabenstellung Unwesentliche wird weggelassen.

Definition: 	<i>Bei der Abstraktion werden wichtige Merkmale von Aktionen einer Aufgabe hervorgehoben, während unwichtige ignoriert werden.</i>
---	--

3.1 Funktionale Abstraktion

Eine funktionale Abstraktion fasst mehrere Aktionen zusammen und macht sie unter einem Namen benutzbar. Anschaulich kann man sich hier ein Unterprogramm in einer Implementierungsumgebung vorstellen. Der Zweck einer funktionalen Abstraktion ist es, die Benutzung von Funktionen durch Angabe ihres Namens zu gestatten, ohne Kenntnisse der Implementierung verwenden zu müssen. Für den Entwurf ist es wichtig, Abstraktionen und ihren Gebrauch in der Definition genau zu beschreiben. Wird eine funktionale Abstraktion angewandt, dann erfolgt der Informationsaustausch mit der Umgebung über Ein-/Ausgabe- Parameter. Wie die Abstraktionen implementiert und realisiert sind, ist für ihren Nutzer nicht wesentlich.

Definition: 	<i>Bei der funktionalen Abstraktion geht man davon aus, dass eine zu entwickelnde Funktion (Methode, Prozedur) schon existiert und diese durch ihren Namen aufgerufen werden kann.</i>
---	--

Damit ist die funktionale Abstraktion ein Hilfsmittel, um die Komplexität der funktionalen Teile der Systembeschreibung zu reduzieren im Sinne einer schrittweisen Verfeinerung bzw. Vergrößerung. Wir haben die Möglichkeit, eine komplexe Aktionsfolge durch eine Gruppe von funktionalen Abstraktionen zu ersetzen, die jeweils einzelne Teile der Aktionsfolge nach inhaltlich vernünftigen Gesichtspunkten zusammenfassen. Durch diese Zusammenfassungen erhalten wir also weniger Aktionsschritte, weil sich hinter jedem einzelnen Aktionsschritt bereits eine Folge elementarer Schritte verbirgt. In Programmiersprachen wird die funktionale Abstraktion durch Funktions-, Prozedur-, und Unterprogramntechniken unterstützt.

Bei der **Top-Down-Vorgehensweise** werden abstrakte Funktionen definiert, die der jeweiligen Abstraktionsebene angepasst sind. In der **Bottom-Up-Vorgehensweise** werden dagegen bereits vorhandene Funktionen zu neuen, abstrakteren Funktionen zusammengefasst.

Damit ist eine funktionale Abstraktion gut geeignet zur Beschreibung von Aktionen oder Folgen von Aktionen. Funktionale Abstraktionen sind aber nicht geeignet, um abstrakte Objekte zu beschreiben, dazu werden Datenabstraktionen benötigt.

3.2 Datenabstraktion

Eine Datenabstraktion ist eine Spezifikation von Wertebereichen und Objekten (Werten) unabhängig von der Darstellung in der realen Welt und von der Darstellung im Computer, d.h. es werden **unwichtigen Einzelheiten** abstrahiert.

Definition: !	<i>Eine Datenabstraktion ist dadurch gegeben, dass der Datentyp ausschliesslich durch die Operationen definiert ist, die mit ihm möglich sind.</i>
------------------------------------	---

Dazu ist es natürlich dennoch erforderlich, bei der Implementierung des Datentyps strukturelle Bindungen vorzunehmen. Diese Bindungen kommen aber erst bei der Implementierungsphase zur Anwendung und werden bis zu diesem Zeitpunkt nicht beachtet. Der Anwender der Datenabstraktion hat also mit der Implementierung der Daten bis dahin nichts zu tun. Er kommuniziert lediglich mit logischen Schnittstellen.

Der Anwender muss keine eigenen Programme schreiben, um die Datenstruktur zu manipulieren. Er darf auch nicht auf die interne Struktur der verborgenen Datenobjekte zugreifen. Abstrakte Datenobjekte können allein durch Anwendung der definierten Zugriffsoperationen benutzt werden, ohne Kenntnis der Implementation.

Die Datenabstraktion ist mit der Vereinbarung einer Variablen gleichzusetzen. Werden mehrere Exemplare benötigt muss ein eigener Datentyp definiert werden. Eine höhere Abstraktionsstufe wird also durch die Definition von abstrakten Datentypen [ADT] gegeben von welchem dann beliebig viele erstellt werden können.

Ziel der Datenabstraktion ist letztlich die Konstruktion von abstrakten Datentypen, bei denen die interne Datenstruktur der Umgebung völlig verborgen wird, jedoch über speziell vorgesehene Funktionen als Schnittstelle zur Umwelt verfügbar gemacht wird. Damit behält der abstrakte Datentyp die Kontrolle über die Datenkonsistenz und über die korrekte Nutzung der Daten. Gleichzeitig macht er die **inkompetente und unzulässige Änderung von Daten unmöglich**. Schliesslich führt er zu einer bestmöglichen Datenunabhängigkeit auf logischer und physikalischer Ebene.

In diesem Kapitel werden wir uns mit funktionalen Abstraktionen befassen. Auch bei Datenabstraktionen spielt die funktionale Abstraktion (der zum Datentyp gehörenden Operationen bzw. Dienste) eine entscheidende Rolle. Ausgangspunkt der Darstellung ist der Modulbegriff, dessen Eigenschaften untersucht werden.

4 Modularisierung

Definition: !	<i>Ein Modul ist zunächst nichts weiter als eine Sammlung von Programmanweisungen (also eine Folge von elementaren Aktionen).</i>
------------------------------------	---

4.1 Eigenschaften eines Moduls

- Ein Modul **besitzt einen Namen**
(Kurzbezeichnung, was das Modul tut aus der Sicht des aufrufenden Programms, nicht wie es dies tut).
- Das Modul ist **unter seinem Namen aufrufbar**
(d.h. bei Aufruf des Moduls kann die in ihm verborgene Aktionsfolge zur Ausführung gebracht werden).
- Es gibt zwei verschiedene Arten von Schnittstellen:
 - Es gibt eine **Eingabeschnittstelle** - Inputs
(Daten, die das Modul von seinem Aufrufer erhält).
 - Es gibt eine **Ausgabeschnittstelle** - Outputs
(Daten, die an den Aufrufer zurückgegeben werden).
- Das "innere" eines Moduls wird verborgen - Abstraktion einer Aktion
(was das Modul macht, um Ausgabedaten aus Eingabedaten zu erzeugen).
- Bei der Benutzung des Moduls darf keine Kenntnis der internen Funktion benutzt werden.
(Information-Hiding)
- Es gibt eventuell **interne Daten**, die das Modul benötigt, um seine Aufgabe auszuführen, die aber wie die interne Funktionsweise gegenüber der Umgebung verborgen werden.

In der Definition ist also nicht die Einschränkung enthalten, dass das Modul etwa unabhängig übersetzbar sein müsse. Derartige Vorstellungen, die wir mit dem besetzten Modulbegriff meistens verbinden, haben erst in der Implementierungsphase einen Sinn. Im Design wollen wir nur Aktionsfolgen mit ihren Vor- und Nachbedingungen identifizieren.

Merke: !	<i>Ein Modul kann aus mehreren einzelnen Funktionen bestehen.</i>
-------------------------------	---

Merke: !	<i>Eine Funktion (auch Element) gilt als kleinste zusammenhängende Gruppe von Anweisungen, die sich als Einheit (Sammlung einzelner Aktionen) ansprechen lässt.</i>
-------------------------------	---

4.2 Äussere und innere Sicht des Moduls

Ein Modul besitzt eine **äussere Sicht** (Schnittstelle zur Kommunikation mit der Aussenwelt) und eine innere Sicht. Die äussere Sicht eines Moduls enthält alle Informationen, die ein Benutzer der Abstraktion benötigt, nämlich Name, Inputs, Outputs, Aufgabe.

Die **innere Sicht** eines Moduls verbirgt die Implementierungsdetails vor dem Benutzer (Geheimnisprinzip). Dies sind die internen Daten, die interne Struktur, die Verarbeitungsregeln, der Algorithmus.

Im Strukturierten Design kümmern wir uns nicht sehr um das Innere des Moduls. Diese haben eine Aufgabe und eine festgelegte Schnittstelle. Diese beiden Eigenschaften reichen für die Konstruktion des Designs aus.

4.3 Ziele der Modularisierung

- Entwicklung von funktionalen Abstraktionen und Datenabstraktionen.
- Gliederung des Systems in überschaubare und gut wartbare Teile.
- Vermeidung von Coderedundanzen.
- Hohe Wiederverwendbarkeit der Module.

5 Modellnotationen des "Structure-Chart"

Diagramme des „Strukturierten Designs“ gelten als sehr verständlich da sie die statische Aufrufstruktur der einzelnen Module gut darstellen können. Structure-Charts haben zu anderen Darstellungsformen den gravierenden Vorteil, dass die **Qualität der Schnittstelle** in den Vordergrund der Überlegungen gestellt wird. Die Diagramme sollen die äussere Sicht jedes im Diagramm gezeigten Moduls suggestiv darstellen, so dass die Modulqualität in ihrer wichtigsten Dimension auf einen Blick beurteilt werden kann.

5.1 Namensgebung und Darstellung der Module

Namen müssen auf die inhaltliche Bedeutung (die Semantik) des Moduls oder des Datenelements hinweisen, für den Betrachter sofort verständlich sein und weitere formale Eigenschaften besitzen.

Im Vergleich zur Namensgebung von Prozessen in der SA-Methode (SA = Structure Analysis) gibt es folgende Unterschiede. Der Name eines Moduls beschreibt die Aufgabe. Es enthält auch die Leistungen aller vom Modul aufgerufenen anderen Module. Im Namen eines Moduls sind also auch die Funktionen aller gerufenen Module enthalten. Dies ist ein Unterschied zu den Prozessnamen in der SA, die nur dazu dienen, den Prozess selber zu beschreiben. Beim Übergang von einem SA-Modell zu einem SD-Modell müssen also mit Sicherheit auch die Namen verändert werden.

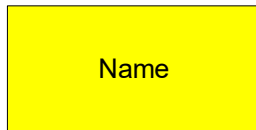


Abbildung 3: Darstellung eines Moduls

Ein **Modul** wird repräsentiert durch ein Rechteck mit Namen. Module werden innerhalb des Modells weiter spezifiziert durch Modulbeschreibungen (Modulspezifikationen), die die innere Sicht für die Implementierung festlegen.

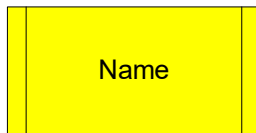


Abbildung 4: Darstellung eines Bibliotheksmoduls

Bibliotheksmodule sind Module, die entweder von der Entwicklungsumgebung verfügbar gemacht werden oder innerhalb des laufenden oder eines früheren Projektes in einer allgemein zugänglichen Bibliothek abgelegt worden sind. Diese Bibliotheksmodule erfordern keine individuelle Beschreibung. Man kann davon ausgehen, dass diese Module schon an anderer Stelle spezifiziert wurden.

5.2 Aufruf eines Moduls, eines Speichers oder eines externen Gerätes

Der Aufruf eines Moduls wird durch einen Pfeil dargestellt, der die Aufrufrichtung angibt. Für systemnahe Anwendungen gibt es auch die Ausdrucksmöglichkeit für **asynchrone Calls**, bei denen der rufende Prozess weiterläuft und nicht auf die Beendigung des gerufenen wartet.

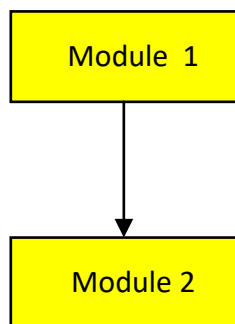


Abbildung 5: Synchroner Aufruf eines Moduls

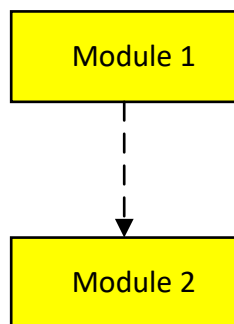


Abbildung 6: Asynchroner Aufruf eines Moduls

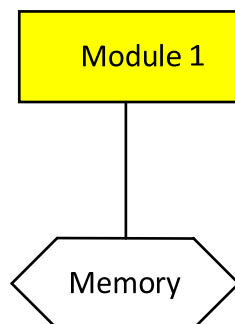


Abbildung 7: Verwendung eines Speichers

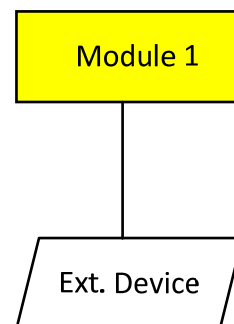


Abbildung 8: Zugriff auf ein externes Gerät

5.3 Parameterübergabe

5.3.1 Datenübergabe

Jeder Pfeil wird mit einem Namen versehen. Dieser sagt etwas über den Dateninhalt aus und besteht aus einem Hauptwort (z.B. Kontonummer, Nettolohn, PLZ)

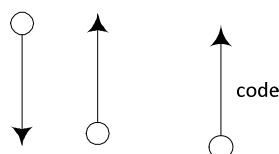
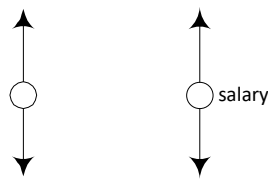


Abbildung 9: Unidirektionale Übergabeparameter

Unidirektionale Schnittstellendaten (Übergabeparameter) werden durch Datenelemente (couples) dargestellt, die im Datenkatalog in ihrer Zusammensetzung definiert werden. Die Pfeilrichtung beschreibt Sender und Empfänger der Datenelemente.

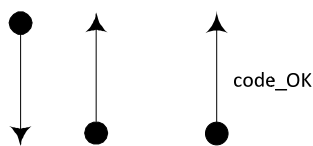


Bidirektionale Schnittstellen können Datenelemente auch einem Modul zur Änderung übergeben. Dies kann man durch einen doppelseitigen Pfeil bei den Datenelementen andeuten.

Abbildung 10: Bidirektionale Übergabeparameter

5.3.2 Steuerungsflags

Jeder Pfeil wird mit einem Flag-Namen versehen. Dieser sagt etwas über die Eigenart und Zustand des Flags aus.



Kontroll-Flags (Kontrollflüsse) dienen zur Steuerung von Modulen. Im weiteren Beschreiben sie die internen Zustände des Moduls von denen sie ausgehen. Sie beschreiben also eine Statusinformation des Moduls.

Abbildung 11: Kontroll-Flags

<p>Merke:</p> <p>!</p>	<p><i>Daten werden verarbeitet und beschreiben externe Objekte; Flags dienen zur Steuerung und beschreiben interne Zustände.</i></p>
--------------------------------------	---

5.4 Eigenschaften des Structure-Charts

Ein Structure-Chart **zeigt** die äussere Sicht eines Moduldesigns, also

- die Aufteilung eines Systems in Module,
- die Hierarchie und Organisation der Module,
- die Schnittstellen zwischen den Modulen,
- Namen und damit Kurzbezeichnungen der Funktion der Module.

Ein Structure-Chart **zeigt nicht** die innere Sicht der Module, also

- die interne Verarbeitung und Modulstruktur,
- interne Daten der Module,
- wie oft und ob überhaupt ein Modul von einem anderen aufgerufen wird,
- in welcher Reihenfolge abhängige Module aufgerufen werden.

6 Modul-Spezifikation

Das Innere jedes Moduls muss spezifiziert werden. Dies geschieht natürlich erst, nachdem die äussere Sicht definiert ist. Die Modulspezifikation muss von den Schnittstellen ausgehen. Ohne klare Schnittstellen-Spezifikationen ist ein Modultest nicht möglich.

Für die Modulspezifikation stehen mehrere Vorgehensweisen zur Verfügung. Modulköpfe und Pre-/Postconditions spezifizieren die äussere Sicht, Übernahme von PSPECs aus der SA und gezielt entwickelter Pseudocode sowie formale Spezifikation und Konstruktion der Programmeinheiten spezifizieren die innere Sicht.

6.1 Modulkopf

Für jedes Modul ist ein Modulkopf erforderlich. Dieser wird später im fertigen Programm am Beginn der Quellcodedatei eingefügt.

Module:		
Source:		
Call:		
Precondition:		
Postcondition:		
Description:		
Parameter:			
Name:	i/o/u/r:	Format:	Description:
.....

Abbildung 12: Inhalt eines Modulkopfs

6.1.1 Inhalt des Modulkopfs

Modul:	<i>Hier wird der inhaltlich sinnvolle (nicht nach technischen Gesichtspunkten vergebene) Name aus dem Structure-Chart eingetragen. Für realisierte Programme ist dieser auch als Kurzbezeichnung im Bibliotheks-Inhaltsverzeichnis sehr nützlich.</i>
Source:	<i>Bei Realisierung wird der Name des Quellmoduls unter "Source" nachgetragen.</i>
Call:	<i>Dieser Begriff beschreibt die Benutzung des Moduls und gibt insbesondere die möglichen Parameter und ihre Reihenfolge an, wird später bei der Codierung nachgetragen.</i>
Preconditions:	<i>Vorbedingungen: Beschreibt die Eigenschaften, die vor der Ausführung des Moduls erfüllt sein müssen.</i>
Postconditions:	<i>Nachbedingungen: Beschreibt die Situation, nachdem das Modul beendet ist.</i>
Parameter:	<i>Unter diesem Bereich werden alle Parameter aufgelistet die etwas mit dem Modul zu tun haben.</i>

Eine Spezifikation der benutzten Parameter kann eventuell bereits in der Zielsprache vorgenommen werden. Hier ist besonders darauf zu achten, dass die Semantik der Parameter vollständig und leicht verständlich beschrieben wird.

Bei der Realisierung wird der Modulkopf ergänzt um den technischen Modulnamen, Autor, Datum und Versionsnummer.

Derartige Modulköpfe werden also im Rahmen des Designs angelegt und zu Beginn der Realisierung in die Realisierungsumgebung exportiert. Die Modulköpfe werden damit zum Ausgangspunkt des Programmcodes.

6.1.2 Spezifikation mit Preconditions / Postconditions

Jede Zustandsänderung eines Ablaufes in einem Programm wird durch Anweisungen hervorgerufen. Daher können wir zu jeder Anweisung und natürlich auch zu jeder Gruppe von Anweisungen den "Zustand davor" und den "Zustand danach" beschreiben. Die relevanten Aspekte dieser Zustände können also auch bei jeder Anweisung spezifiziert werden.

Wir gehen also davon aus, dass ein Modul als zusammenhängende Folge von Anweisungen durch Angabe der Vor- und Nachbedingungen präzise spezifizierbar ist. Vor- und Nachbedingungen beschreiben die Funktion des Moduls, ohne viel über den Algorithmus selber festzulegen. Die äussere Sicht des Moduls wird definiert durch den Zustand zu Beginn des Moduls und den Zustand am Ende. Wie der Algorithmus dann implementiert wird, das bleibt dem nächsten Konstruktionsschritt, der Realisierung, überlassen und ist Gegenstand der funktionalen Abstraktion.

6.1.2.1 Vorbedingungen (Preconditions)

Diese beschreiben die Eigenschaften, die **vor Ausführung des Moduls** erfüllt sein müssen.

Folgende Fragen müssen zum Beispiel beantwortet werden:

- Welche Inputs müssen verfügbar sein?
- Welche Beziehungen bestehen zwischen den Inputs?
- Welche Beziehungen bestehen zwischen Inputs und Speicherinhalten?
- Welche Beziehungen bestehen zwischen Inhalten verschiedener Speicher?

6.1.2.2 Nachbedingungen (Postconditions)

Diese beschreiben die Situation, **nachdem das Modul beendet ist**.

Diese Beschreibung enthält meistens:

- Die vom Modul erzeugten Outputs,
- die Beziehungen zwischen Ausgabe- und Eingabeparametern,
- die Beziehung zwischen den Ausgabeparametern und Speichereinträgen,
- Änderungen von Speichereinträgen, welches das Modul durchgeführt hat.

6.2 Beispiel einer Modulspezifikation

Module:	<i>Nullstelle berechnen</i>		
Source:	<i>Funktionenberechnung.c</i>		
Call:	CalcZeroPoint (function, lowerLimit, upperLimit, tolerance)		
Precondition:	1. lowerLimit < upperLimit 2. tolerance > Darstellungsgenauigkeit des Rechners 3. function stetig differenzierbar, liegt als REAL FUNCTION vor 4. function besitzt im Intervall eine Nullstelle		
Postcondition:	Der unter CalcZeroPoint zurückgegebene Wert hat folgende Bedeutung: lowerLimit <= CalcZeroPoint <= upperLimit: Abs (function(CalcZeroPoint)) < tolerance; berechnete Nullstelle (ok) CalcZeroPoint = lowerLimit – 1000.0: Newton-Verfahren konvertiert nicht CalcZeroPoint = lowerLimit – 2000.0: Vorbedingung nicht erfüllt		
Description:	Berechnet die Nullstelle der Funktion: function im Intervall [lowerLimit, upperLimit] innerhalb der Genauigkeit: tolerance		
Parameter:			
Name:	i/o/u/r:	Format:	Description:
function	i	double	Funktion deren Nullstelle berechnet werden soll
lowerLimit	i	double	Untergrenze des Intervalls
upperLimit	i	double	Obergrenze des Intervalls
tolerance	i	double	vorgegebene Rechnergenauigkeit
CalcZeroPoint	o	double	berechnete Nullstelle bzw. Fehlercode

7 Kopplung versus Kohäsion

In diesem Abschnitt des Strukturierten Entwurfes wollen wir auf die Kriterien eines guten Modulentwurfes eingehen.

Diese sind gleichzeitig die Eigenschaften, die später nach Realisierung zu einem qualitativ hochwertigen System führen, das die unterschiedlichen Qualitätsanforderungen der Anwender realisiert.

Merke: !	<i>Kopplung und Zusammenhalt (Kohäsion) sind die wichtigsten Kriterien für die Qualität des Moduldesigns.</i>
-------------------------------	--

Die Kopplung beschreibt die Kommunikation zwischen Modulen; der Zusammenhalt die internen Eigenschaften jedes einzelnen Moduls. Darüber hinaus gibt es weitere Kriterien, die bei der Konstruktion des Moduldesigns zu beachten sind und im Folgenden behandelt werden sollen.

7.1 Kopplung

7.1.1 Gründe von Modulkopplungen

Merke: !	<i>Kopplung ist der Grad der Abhängigkeit zwischen Modulen.</i>
-------------------------------	--

In jedem Design muss das Ziel verfolgt werden, die Kopplung zu minimieren, also Module so unabhängig voneinander wie möglich zu gestalten. Hierfür gibt es folgende Gründe:

- Je loser die Kopplung zwischen Modulen ist, desto geringer ist die Gefahr von Fernwirkungen (Fehler in einem Modul erscheinen als Symptom in einem anderen Modul).
- Bei Änderung eines Moduls soll möglichst eine Änderung anderer Module nicht erforderlich werden.
- Bei der Nutzung eines Moduls sollen die internen Details aller anderen Module unberücksichtigt bleiben.
- Das System soll so einfach und so verständlich wie möglich werden.
(*"KISS-Prinzip": "keep it simple, stupid"*)

7.1.2 Kopplungsarten

Generell werden drei Kopplungsarten unterschieden. Diese heissen:

- Normale Kopplung
- Globale Kopplung
- Inhaltsbezogene Kopplung

Für die einzelnen Formen der Kopplung gibt es eine Skala, die von guten und der Gesamtqualität nicht abträglichen Formen bis zu vollkommen unbrauchbaren Arten der Kopplung reicht, die man jedoch in der Praxis manchmal durchaus antreffen kann.

7.1.2.1 Klassifizierung der Kopplungsarten

Kopplungsstufe:	Anzustrebendes Ziel	Kopplungsart:
Kopplung klein	<p>sehr gut</p> <p>mittelmässig</p> <p>sehr schlecht</p> <p><i>Abbildung 13: Kopplungsarten</i></p>	Normale Kopplung: <ul style="list-style-type: none"> • Datenkopplung • Datenstrukturkopplung • Kontrollkopplung
Kopplung mittel		Globale Kopplung (sollte nicht verwendet werden)
Kopplung gross		Inhaltsbezogene Kopplung (sollte nicht verwendet werden)

In der praktischen Anwendung ist nur die normale Kopplung akzeptabel, welche die folgenden drei Arten der Kopplung umfasst:

- Datenkopplung
- Datenstrukturkopplung
- Kontrollkopplung

Zwei Module A und B sind normal gekoppelt, wenn Modul A das Modul B aufruft, B die Kontrolle an A zurückgibt und wenn alle Informationen zwischen A und B durch explizite Parameter im Aufruf selber übergeben werden.

Merke: 	Globale- und Inhaltsbezogene-Kopplungen sollten, wenn immer möglich, vermieden werden!
-----------------------	---

Für die Kopplung von Modulen gilt das folgende Geheimnisprinzip:

Merke: 	Jedes Modul sollte immer nur die Informationen erhalten, die es für seine Aufgabenerledigung auch tatsächlich benötigt.
-----------------------	--

7.1.2.2 Datenkopplung

Merke: !	Normal gekoppelte Module sind datengekoppelt, wenn alle Übergabeparameter nur Variablen von einfachen Datentypen sind (einzelnes Feld oder homogene Tabelle).
-------------------------------	--

Die Datenkopplung ist eine notwendige Kommunikation zwischen Modulen.

Beispiel einer Datenkopplung:

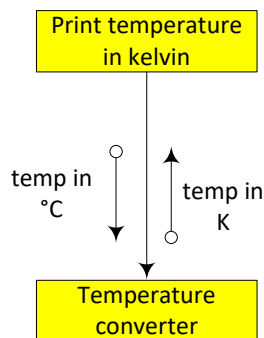


Abbildung 14: Beispiel einer Datenkopplung

In diesem Beispiel wird eine gegebene Temperatur in °C an ein Umrechnungsmodul übergeben. Dieses Modul hat die Aufgabe den Celsius-Temperaturwert in Kelvin umzurechnen und dann den neu berechneten Wert dem aufrufenden Modul für die Ausgabe zu übergeben.

Das aufrufende Modul „Temperatur in Kelvin ausgeben“ ruft das zweite Modul „Temperaturumrechnung“ auf.

Als Daten werden zwischen den Modulen die Variablen „Temp in °C“ und „Temp in K“ ausgetauscht. Bei beiden Datenelementen handelt es sich um einfache Datentypen.

Folgende Punkte müssen bei dieser Kopplungsart beachtet werden:

- Es dürfen nur Daten übergeben werden die auch im Modul verwendet werden. So genannte „Tramp“-Daten dürfen nicht übergeben werden, da diese ja nicht gebraucht werden. „Tramp“-Daten sind Daten, die ohne offensichtliches Ziel durch das ganze System gereicht werden, von niemandem gebraucht werden und für niemanden von Bedeutung sind.
- Module mit einfacher Datenkopplung sollten stets dort aufgerufen werden, wo ihre Leistung tatsächlich benötigt wird, sonst können leicht „Tramp“-Daten entstehen.
- Es sollten nicht mehr als 7 Parameter an ein Modul übergeben werden. Durch diese Massnahme bleibt die Schnittstelle übersichtlich.
- Die Übergabeparameter sollten von möglichst einfacher Datenstruktur sein.
- Das aufgerufene Modul soll leicht durch ein anderes mit gleicher Schnittstelle ausgetauscht werden können.
- Es soll ohne Abänderung auch von anderen Modulen verwendet werden können, da es nichts über das aufrufende Modul (Aufrufer) wissen muss.

7.1.2.3 Datenstrukturkopplung

Merke: !	Zwischen zwei normal gekoppelten Modulen liegt Datenstrukturkopplung vor, wenn eines dem anderen eine zusammengesetzte Datenstruktur übergibt.
-------------------------------	---

Beide Module benutzen also die gleiche, nicht globale Datenstruktur, die aber als Argument explizit übergeben wird.

Beispiel einer Datenstrukturkopplung:

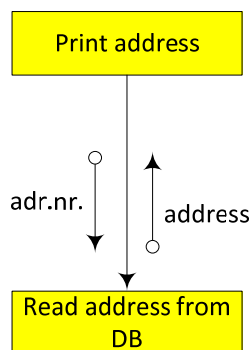


Abbildung 15: Beispiel einer Datenstrukturkopplung

In diesem Beispiel wird eine Adresse aus der Datenbank herausgelesen. Das Modul „Adresse ausgeben“ ruft das zweite Modul „Adresse aus DB lesen“ auf.

Das aufrufende Modul schickt zum zweiten Modul die Variable: „Adr.Nr.“. Mit diesem Wert kann das aufgerufene Modul die Adresse aus der Datenbank selektieren und dann die Rückgabedaten: „Adresse“ an das erste Modul übergeben. Bei den Rückgabedaten handelt es sich um eine Datenstrukturkopplung.

Die Datenstrukturkopplung hat eine kleine Kopplung, wenn das übergebene strukturierte Datenelement nur Komponenten enthält, die das empfangende Modul auch tatsächlich benötigt und wenn diese Komponenten auch inhaltlich zusammengehören.

In allen anderen Fällen spricht man von zu breiter Kopplung oder von Bündelung von Datenelementen, die nur aus dem einen Grunde vorgenommen ist, die Anzahl der Schnittstellenparameter zu reduzieren. Die Komplexität der Schnittstelle bleibt natürlich gleich. Zu breite Kopplung und Bündelung können zu unverständlichen Datenstrukturen führen. Man sollte nur diejenigen Datenfelder zu einer Struktur zusammenfassen, die auch sinngemäss zusammengehören.

Die Datenstrukturkopplung hat die folgenden Eigenschaften:

- Es dürfen nur Datenstrukturen verwendet werden, wenn auch alle Daten darin zum Einsatz kommen.
- Jede Änderung der übergebenen Datenstruktur (Format, Reihenfolge...) berührt alle Module, die sie benutzen. Betroffen sind auch diejenigen Module, die geänderte Felder gar nicht nutzen. Dadurch entstehen Abhängigkeiten zwischen sonst unabhängigen Modulen.
- Die Wiederverwendbarkeit von datenstrukturgekoppelten Modulen wird eingeschränkt. Je enger die Komponenten der Datenstruktur zusammengehören, desto ungefährlicher ist die Datenstrukturkopplung, desto wiederbenutzbarer wird auch das Modul.
- Es dürfen keine künstlich zusammengesetzten Datenstrukturen gebildet werden.

7.1.2.4 Kontrollkopplung

Merke: !	Zwischen zwei normal gekoppelten Modulen liegt Kontrollkopplung vor, wenn eines dem anderen ein Datenelement übergibt, das die interne Verarbeitungslogik des anderen beeinflussen soll.
-------------------------------	---

Zwischen den Modulen werden also Kontrollelemente (Schalter) übermittelt, die direkt den internen Kontrollfluss im empfangenden Modul beeinflussen.

Beispiel einer Kontrollkopplung:

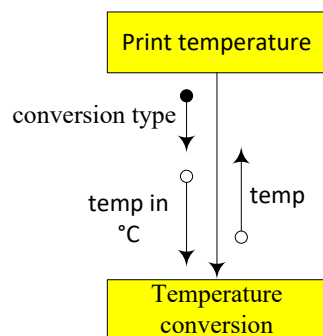


Abbildung 16: Beispiel einer Kontrollkopplung

In diesem Beispiel wird eine gegebene Temperatur in °C an ein Umrechnungsmodul geschickt. Dieses hat die Aufgabe den °C-Wert in eine andere Temperaturskala umzurechnen und dann den neu berechneten Wert dem aufrufenden Modul zur Ausgabe zu übergeben. Mit dem Kontroll-Flag wird dem aufgerufenen Modul mitgeteilt, welche Temperaturumrechnungsart es verwenden soll. So kann z.B. die Umrechnung auf Kelvin oder Fahrenheit eingestellt werden.

Als Daten werden zwischen den Modulen die Variablen „Temp in °C“, „Welche Umrechnung“ und „Temp“ ausgetauscht. Bei allen Datenelementen handelt es sich um einfache Datentypen.

Da das aufrufende Modul die Logik des aufgerufenen Moduls kennen muss, ist diese Kopplungsart nicht so gut wie die reine Datenkopplung. Trotzdem ist die Kontrollkopplung durchaus akzeptabel und wird dadurch auch häufig eingesetzt.

Zu unterscheiden ist die Richtung und Art der Kontrollinformation. Fließt die Information vom aufgerufenen Modul zum Aufrufer zurück, so ist dies ein normaler und gebräuchlicher Vorgang. Auf diese Weise können Statusmeldungen über das aufgerufene Modul an den Aufrufer zurückgegeben werden, welches dann in der geeigneten Weise reagieren kann.

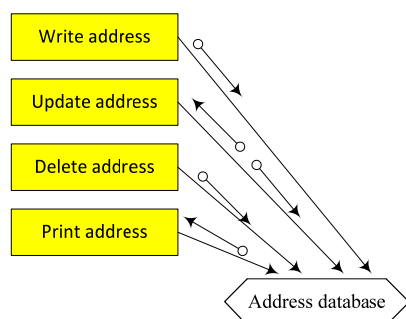
Kontrollinformationen die aber in umgekehrter Richtung fließen (Aufrufer → aufgerufenes Modul) sollten aus der Sicht der Wiederverwendbarkeit unterlassen werden, da diese meist nur für spezielle Zwecke eingesetzt werden können.

7.1.2.5 Globale Kopplung

Merke: !	<i>Zwei Module heissen global gekoppelt, wenn sie den gleichen externen, global definierten Speicherbereich benutzen.</i>
-------------------------------	--

Globale Kopplung ist aus vielen Gründen gefährlich, dennoch ist sie bei vielen, gerade jüngeren Entwicklern sehr beliebt. Sie vereinfacht scheinbar die Übergabe von Parametern an Unterprogramme. Wird aber der Aufwand mitberücksichtigt welcher bei der Fehlersuche entsteht, wenn mehrere Module auf dieselbe Datenbasis zugreifen, dann wird man erst merken, welche Schwierigkeiten diese Kopplungsart mit sich bringt.

Beispiel einer globalen Kopplung:



Im nebenstehenden Diagramm erkennen Sie den Aufruf einer Adress-Datenbank aus diversen Modulen. Auf diese Art werden Daten global zur Verfügung gestellt ohne dass diese explizit über Parameterschnittstellen den einzelnen Modulen übergeben werden.

Abbildung 17: Beispiel einer globalen Kopplung

Worin bestehen die Gefahren bei dieser Kopplungsart?

- Ein Fehler oder eine Fahrlässigkeit in irgendeinem Modul, das globale Daten benutzt, kann sich in einem beliebigen anderen Modul auswirken, welches diese globalen Daten ebenfalls benutzt. Jedes Modul, welches globale Daten benutzt, kann Daten verändern, ohne eine Spur zu hinterlassen. Globale Daten residieren nicht in der "schützenden funktionalen Hülle eines Moduls". Die Fehleranalyse in einem System, das globale Datenkopplung benutzt, ist in jedem Falle um ein Vielfaches aufwendiger als bei Nutzung einfacherer Kopplungsarten. Im Prinzip muss jede einzelne Anweisung geprüft werden, denn jede einzelne Anweisung kann für den Fehler verantwortlich sein. Das Abstraktionsprinzip ist auf das grösste verletzt.
- Die Benutzung vieler globaler Datenfelder führt zu unverständlichen Modulen. Bei der Analyse eines fehlerhaften Programms muss nämlich verstanden werden, welches andere Modul die Felder wie benutzt. Es reicht nicht mehr aus, sich nur die Schnittstellen der anderen Module anzusehen.
- Besonders unerfreulich ist die Situation, wenn verschiedene Module dieselben globalen Felder für unterschiedliche Zwecke benutzen. Der Feldinhalt hängt dann davon ab, welches Modul das Feld als letztes benutzt hat. Die Fehlersuche ist erschwert.
- Durch die Verwendung globaler Daten sind Feldnamen festgelegt. Im Modul selber, aber auch im Übergabebereich müssen schwerwiegende Kompromisse hinsichtlich der Qualität der Namensgebung hingenommen werden. Dadurch werden Programme tendenziell unverständlich.

- Systeme mit Nutzung globaler Daten sind schwer wartbar. Es ist manchmal kaum zu übersehen, welche Konsequenzen durch Änderung eines globalen Datenfeldes entstehen.
- Die Mehrfachbenutzbarkeit von Modulen ist durch Nutzung globaler Daten stark eingeschränkt.

7.1.2.6 Inhaltskopplung

Merke: !	Zwischen zwei Modulen besteht eine Inhaltskopplung, wenn das eine Modul das Innere des anderen in irgendeiner Form adressiert und sich damit auf die Art der Codierung bezieht.
-------------------------------	--

Ein inhaltsgekoppeltes Modul kann direkt auf den Inhalt eines anderen zugreifen und dieses unter Umständen sogar ändern (z.B. Modifizierung eines Programmstatements, Sprung in das benutzte Modul). **Diese Kopplung ist grob gefährlich!**

Beispiel einer Inhaltskopplung:

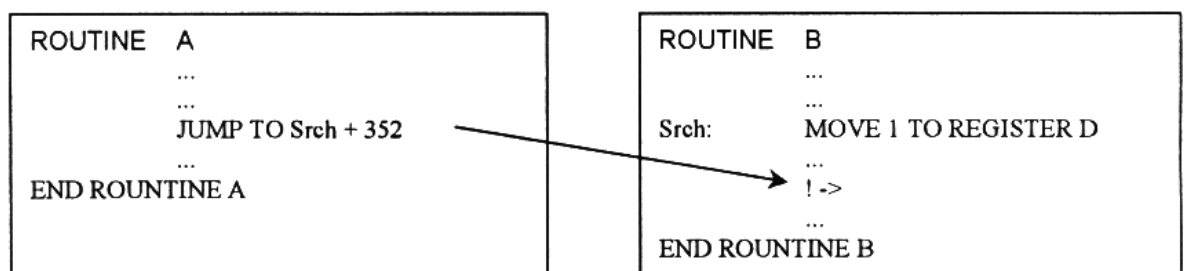


Abbildung 18: Beispiel einer Inhaltskopplung

Aus der Routine A wird inmitten einer anderen Routine B verzweigt. Der Sprung aus Routine A endet nicht etwa bei der Marke: Srch von Routine B, sondern bei Srch + 352 Bytes (Speziell mit einem Ausrufezeichen markiert). Das bedeutet, dass Routine A unüberschaubar an die internen Details der Routine B gebunden ist. In der Routine B kann zwischen Srch und „!“ nichts geändert werden, ohne Gefahr zu laufen, dass im Ablauf der Routine A dadurch ein Fehler entsteht.

7.2 Zusammenhalt (Kohäsion)

Merke: !	<i>Zusammenhalt ist der Grund der funktionellen Zusammengehörigkeit der Elemente (Anweisungen oder Gruppen von Anweisungen, «Funktionen») eines Moduls.</i>
-------------------------------	--

Für ein gutes Moduldesign werden die folgenden Ziele verfolgt:

- Den **Zusammenhalt** des Moduls soll **maximiert** werden.
- Jedes Modul soll **eine** Aufgabe erfüllen, diese aber komplett. Dadurch können die Module besser wiederverwendet werden.

7.2.1 Beziehung von Kopplung und Zusammenhalt

Kopplung und Zusammenhalt stehen in enger Beziehung zueinander:

- Die Elemente jedes Moduls sollen kaum zu den Elementen anderer Module in Beziehung stehen, - dies würde zu starker Kopplung führen.
- Wenn man sicherstellt, dass alle Module guten Zusammenhalt haben, dann ist meistens auch die Kopplung dieser Module lose. Umgekehrt kann man eine lose Kopplung nur erreichen, wenn man starke, also gut zusammenhaltende Module benutzt.

7.2.2 Arten der inneren Bindung (Zusammenhalt)

Wartbarkeit	Innere Bindungsart
<p>Abbildung 19: Innere Bindungsarten</p>	<ul style="list-style-type: none"> • Funktionale Bindung • Sequentielle Bindung • Kommunikative Bindung • Problembezogene Bindung (Prozedurale Bindung) • Zeitliche Bindung (Temporäre Bindung) • Programmstrukturelle Bindung (Logische Bindung) • Zufällige Bindung

Da in der praktischen Anwendung **nur der normale (auch datenstrukturelle) Zusammenhalt akzeptabel** ist, wird auch nur diese Bindung in den Unterlagen genauer behandelt. Generell lässt sich der datenstrukturelle Zusammenhang in drei Arten unterteilt. Diese heissen:

- funktionaler Zusammenhalt
- sequentieller Zusammenhalt
- kommunizierender Zusammenhalt

7.2.3 Definition des normalen (datenstrukturellen) Zusammenhalts

Analog zu der normalen Kopplung definieren wir:

Merke: !	„Ein Modul besitzt normalen Zusammenhalt, wenn es eine oder mehrere Funktionen umfasst, die inhaltlich eng zusammengehören, und die mindestens auf einer gemeinsamen Datenstruktur operieren, die lokal definiert und damit für andere verborgen ist oder die explizit als Parameter übergeben wird.“
-------------------------------	--

Andere Formen des Zusammenhaltes (**problembezogener, zeitlicher, programmstruktureller, zufälliger Zusammenhalt**) führen zu schlechter Wartbarkeit und geringer Wiederbenutzbarkeit und sind daher zu vermeiden. Dies sind die **gefährlichen Zusammenhalte**.

7.2.3.1 Funktionaler Zusammenhalt

Merke: !	Ein Modul heisst funktional zusammenhaltend, wenn es nur Elemente enthält, die zusammen eine einzige Funktion ausführen.
-------------------------------	---

Eigenschaften des funktionalen Zusammenhalts:

- Hat einen aussagefähigen Namen, der bezeichnend ist, aber nur eine Funktion unverwechselbar beschreibt (Beispiele: berechnen Netto-Gehalt, Datum prüfen, Sinus eines Winkels berechnen).
- Im internen Aufbau mag das Modul beliebig kompliziert sein und auch mehrere Unterfunktionen aufrufen. Die Gesamtleistung aber lässt sich als eine problembezogene und eindeutige Funktion bezeichnen.
- Die äussere Sicht eines funktional zusammenhaltenden Moduls ist meistens leichter zu verstehen als seine innere Sicht.
- Ein funktional zusammenhaltendes Modul besteht aus Elementen, die alle zusammen notwendig, aber auch ausreichend sind, um eine spezielle problembezogene Aufgabe zu erfüllen.
- Jeder Aufrufer (Module welches die Leistung eines anderen Moduls beansprucht) wird genau gleichbehandelt.

Beispiel eines funktionalen Zusammenhangs

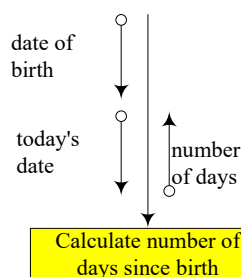


Abbildung 20: Beispiel eines funktionalen Zusammenhangs

Im nebenstehenden Beispiel sind alle spezifischen Subroutinen die zur Berechnung der Anzahl Tage benötigt werden in einem Modul integriert. Ausserhalb des Moduls sind diese aber nicht erkennbar und können dadurch auch nicht angesprochen werden. Sie dienen also nur demjenigen Modul, indem sie integriert sind.

7.2.3.2 Sequentieller Zusammenhalt

<p>Merke:</p> <p>!</p>	<p><i>Ein sequentiell zusammenhaltendes Modul besteht aus einer Folge von Aktivitäten, die nacheinander bearbeitet werden und bei denen die Ausgaben des einen die Eingabe des nächsten sind.</i></p>
--------------------------------------	--

Eigenschaften des sequentiellen Zusammenhalts:

- Besitzen normalerweise gute Kopplungseigenschaften.
- Sind leichter wartbar.
- Eventuell ist ihre Wiederverwendbarkeit eingeschränkt, weil die Aktivitäten nur alle gemeinsam in vorab festgelegter Reihenfolge ausgeführt werden können.

Ein guter sequentieller Zusammenhalt ist dadurch gekennzeichnet, dass die Elemente des Moduls einzeln nicht benötigt werden, so dass ihre Aufteilung in mehrere Funktionen ohnehin nur theoretischen Wert hätte.

Beispiel eines sequentiellen Zusammenhangs

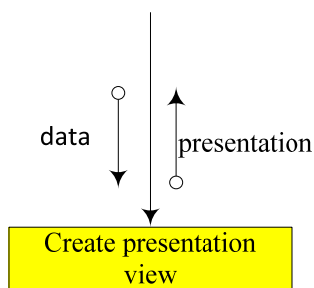


Abbildung 21: Beispiel eines sequentiellen Zusammenhangs

In diesem Beispiel soll eine Präsentation erstellt werden. Um dies zu tun, müssen die folgenden Schritte nacheinander ausgeführt werden:

1. Daten aufbereiten
2. Daten an der richtigen Stelle im Formular einsetzen
3. Formular in eine druckfertige Version umwandeln

Wie Sie erkennen können, müssen diese Schritte durch einzelne Subroutinen abgearbeitet werden und zwar so, dass die Reihenfolge eingehalten wird.

7.2.3.3 Kommunizierender Zusammenhalt

Merke: !	<i>Ein kommunizierend zusammenhaltendes Modul führt verschiedene Aktivitäten aus, die alle dieselben Eingabe- oder Ausgabe-Daten benutzen. Ansonsten haben die Aktivitäten jedoch wenig miteinander zu tun. Insbesondere ist die Reihenfolge nicht durch inhaltliche Erfordernisse festgelegt.</i>
-------------------------------	---

Solch ein Modul kann z.B. einen komplizierten Datensatz einlesen und mehrere Felder daraus extrahieren.

Eigenschaften des kommunizierenden Zusammenhalts:

- Führt verschiedene Aktivitäten aus, die alle dieselben Eingabe- oder Ausgabe-Daten benutzen.
- Sind gut wartbar und können meistens auch gut wiederverwendet werden.

In fast allen Fällen lässt sich die Qualität steigern, wenn man das Modul in mehrere, nur funktional zusammenhängende Module gliedert. Diese werden jedoch manchmal sehr klein.

Im Gegensatz zur sequentiellen Kohäsion spielt die Reihenfolge der Aktivitäten in der kommunikativen Kohäsion keine Rolle. Auf das untenstehende Beispiel bezogen, spielt es also keine Rolle ob zuerst der Sinus-, dann der Kosinus- und als letztes der Tangens-Wert berechnet wird. Die Reihenfolge kann also in beliebigem Sinne verändert werden.

Beispiel eines kommunizierenden Zusammenhalts

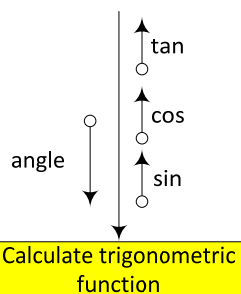


Abbildung 22: Beispiel eines kommunizierenden Zusammenhalts

Nebenstehendes Beispiel zeigt das Modul „Berechne Winkelfunktionen“. Sie erkennen, dass mit nur einem Eingabewert verschiedene Ausgabedaten erstellt werden. Es werden also für die verschiedenen Berechnungen die gleichen Basisdaten herangezogen. Benötigt der Anwender aber nur den Sinuswert, so ist dies nicht möglich da die anderen Werte ebenfalls mitgeliefert werden. Er hat somit keine andere Möglichkeit als die nicht benötigten Daten zu unterdrücken. Dieser Sachverhalt ist ein Hinweis auf eine schlechte Kopplung und sollte, wenn immer möglich, unterdrückt werden.

7.2.3.4 Problembezogener (prozeduraler) Zusammenhalt

In einem Modul werden verschiedene, möglicherweise unabhängige Funktionen zusammengefasst, bei denen die Kontrolle von einer an die nächste übergeben wird. Bei dieser Art des Zusammenhalts gibt es meistens kaum Beziehungen zwischen den Eingangs- und Ausgangsparametern des Moduls. Oft werden halbfertige Ergebnisse weitergereicht.

7.2.3.5 Zeitlicher Zusammenhalt

Verschiedene unabhängige Aktivitäten werden ausgeführt, deren einziger Zusammenhang darin besteht, dass sie zur gleichen Zeit oder zu einem bestimmten Zeitpunkt unmittelbar nacheinander in festgelegter Reihenfolge ausgeführt werden (z.B. Initialisierung, Finalisierung). Module mit zeitlichem Zusammenhalt bestehen meist aus unabhängigen Funktionen oder Funktionsfragmenten. Diese stehen oft zu Aktivitäten anderer Module in engerer Beziehung als untereinander. Als Resultat ergibt sich meist eine starke Kopplung zum Aufrufer.

7.2.3.6 Programmstruktureller Zusammenhalt

Beim Aufruf des Moduls wird eine Funktion oder eine Menge von miteinander verflochtenen Funktionen gesteuert über ein Flag ausgeführt.

Ein früher sehr beliebtes Beispiel für programmstrukturellen Zusammenhalt ist die Zusammenfassung aller Ein-/Ausgaben einer Datei in einem Unterprogramm. Die einzelnen Funktionen haben nichts miteinander zu tun. Um alle Aufgaben abdecken zu können, ist eine unverhältnismässig breite Schnittstelle erforderlich, von der bei jedem konkreten Aufruf immer nur ein kleiner Teil der Parameter tatsächlich genutzt wird. Beim programmstrukturellen Zusammenhalt besteht die Gefahr, dass Programmteile miteinander untrennbar verknüpft werden mit der Folge einer schlechten Änderbarkeit und mangelhaften Zuverlässigkeit. Ausserdem entsteht eine starke Kopplung zum Aufrufer.

7.2.3.7 Zufälliger Zusammenhalt

Das Modul besteht aus mehreren, nicht zusammengehörigen Codefragmenten. Diese können zum Beispiel entstehen, wenn man vorher nur zeitlich zusammenhaltende Module mit Schaltern (Flags) versieht, um Teile daraus in anderem Zusammenhang zu verwenden, oder wenn mehrere Funktionen wahllos zusammengelegt werden, oder wenn ein grösseres Programm wahllos in Module zerlegt wird.

8 Zusammenfassung

Die Forderungen nach hohem Zusammenhalt und loser Kopplung lassen sich leicht gemeinsam erfüllen: Module mit funktionalem, sequentiell oder kommunikativem Zusammenhalt besitzen einfache Schnittstellen und können mit ihren Aufrufern datengekoppelt, datenstrukturegekoppelt oder kontrollgekoppelt kommunizieren.

Entscheidende Eigenschaften sind:

- Jedes Modul sollte immer nur die Informationen erhalten, die es für seine Aufgabenerledigung auch tatsächlich benötigt.
- An Module sollten Informationen stets als explizite Parameter mit einfachen Datenstrukturen übergeben werden.
- Die inneren Details von Modulen sollten vor jedem Nutzer verborgen werden (information hiding).
- Jedes Modul soll eine Aufgabe erfüllen, diese aber vollständig, ohne Zwischenprodukte zu erzeugen oder von anderen Modulen erzeugte Zwischenprodukte verarbeiten zu müssen. Dann ist auch eine lose Kopplung erreichbar.

Abbildungen:

Abbildung 1: Wasserfallmodell	3
Abbildung 2: Beispiel eines Structure-Charts	3
Abbildung 4: Darstellung eines Moduls	8
Abbildung 5: Darstellung eines Bibliotheksmodul	8
Abbildung 6: Synchroner Aufruf eines Moduls	8
Abbildung 7: Asynchroner Aufruf eines Moduls	8
Abbildung 8: Verwendung eines Speichers	8
Abbildung 9: Zugriff auf ein externes Gerät	8
Abbildung 10: Unidirektionale Übergabeparameter	8
Abbildung 11: Bidirektionale Übergabeparameter	9
Abbildung 12: Kontroll-Flags	9
Abbildung 13: Inhalt eines Modulkopfs	10
Abbildung 14: Kopplungsarten	14
Abbildung 15: Beispiel einer Datenkopplung	15
Abbildung 16: Beispiel einer Datenstrukturkopplung	16
Abbildung 17: Beispiel einer Kontrollkopplung	17
Abbildung 18: Beispiel einer globalen Kopplung	18
Abbildung 19: Beispiel einer Inhaltskopplung	19
Abbildung 20: Innere Bindungsarten	20
Abbildung 21: Beispiel eines funktionalen Zusammenhangs	21
Abbildung 22: Beispiel eines sequentiellen Zusammenhangs	22
Abbildung 23: Beispiel eines kommunizierenden Zusammenhangs	23

Historie

Vers.	Bemerkungen	Verantwortlich	Datum
0.1	- erstes Zusammenführen der Informationen	W. Odermatt	09.03.2003
0.2	- erster Entwurf fertig gestellt	W. Odermatt	16.03.2003
1.0	- letzte Layoutanpassungen gemacht	W. Odermatt	17.03.2003
1.1	- Inhaltsüberarbeitung	W. Odermatt	22.02.2017
2.0	- Anpassungen und Erweiterungen an neuen Modulinhalt: M411	W. Odermatt	06.03.2017

Referenzunterlagen

LfNr.	Titel / Autor / File / Verlag / ISBN	Dokument-Typ	Ausgabe
1	Ed Yourdon's Home Page: http://www.yourdon.com/	Theorieunterlagen	2003
2	Wikipedia, Structure chart, Page: https://en.wikipedia.org/wiki/Structure_chart	Theorieunterlagen	2017
3	„Systementwicklung mit Strukturierten Methoden“ / Jörg Raasch / Hanser-Verlag / 3-446-17263-7	Fachbuch	1992
4	„Lehrbuch der Software-Technik“ / Helmut Balzers / Spektrum-Verlag / 3-8274-0042-2	Fachbuch	2009