

Software Engineering

Einführung in das Software Engineering

Ein Tutorial von Wirtschaftsinformatik-24.de

Inhaltsverzeichnis

Einleitung	2
Ziele vom Software Engineering	2
Prinzipien, Methoden und Verfahren	3
Software Life Cycle	3
Qualitätsanforderungen an Software	4
Vorgehensmodelle	5
Wasserfallmodell	5
Rapid Prototyping	7
Spiralmodell	8
V-Modell	10
Rational Unified Process	11
Übersicht Modellierungsmethoden	12
Strukturierte Analyse	13
Funktionsmodellierung	13
Datenflussmodellierung	14
Prozessspezifikation	15
Data Dictionary	15
Strukturiertes Design	16
Modularisierung	16
Kohäsion	17
Kopplung	19

Einleitung

Hintergrund für die Einführung des **Software Engineerings** ist, dass in den **Anfängen der Softwareprogrammierung** (60iger Jahre) komplexe Systeme völlig unstrukturiert entwickelt wurden.

Dies hatte zur Folge, dass die Wartungskosten der Systeme oft sehr viel höher waren als die eigentlichen Entwicklungskosten. Deswegen sollte versucht werden - wie im Ingenieurwesen auch - **Software-Systeme nach Prinzipien und Regeln zu erstellen.**

Ziele vom Software Engineering

Als das **Software Engineering** eingeführt wurde, sollten die bekannten Vorgehensweisen aus den **Ingenieurwissenschaften auf die Softwareentwicklung** übertragen werden.

- Verkürzung der Entwicklungszeit
- Anhebung der Qualitätsstandards
- Systematische Softwareentwicklung
- Senkung der Kosten für die Software-Entwicklung

Prinzipien, Methoden und Verfahren

Im **Software Engineering** unterscheidet man zwischen **Prinzipien, Methoden** und **Verfahren**. Diese drei elementaren Begriffe sollen im Folgenden definiert und voneinander abgegrenzt werden.

Prinzipien

Als Prinzipien bezeichnet man im Software Engineering **Handlungsgrundsätze**.

Beispiele: Prinzipien können beispielsweise Modularisierung, Hierarchisierung oder Strukturierung sein.

Methoden

Methoden hingegen sind **geplante Vorgehensweisen**, die dazu dienen, festgelegte Ziele zu erreichen.

Beispiele: Zerlegung von Problemen, Strukturiertes Programmieren.

Verfahren

Ein Verfahren im Software Engineering ist eine **Anweisung oder Kontrollstruktur**. Mit diesen sollen Methoden gezielt eingesetzt werden.

Beispiele: Nassi-Schneidermann-Diagramm.

Software Life Cycle

Der **Software Life Cycle** ist ein Zyklusmodell, welches auch als **Phasenmodell** bekannt ist. Es ist ein fester **Bestandteil der strukturierten Software-Entwicklung**.

- Analyse- und Definitionsphase
- Entwurfsphase
- Implementierungsphase
- Betriebs- und Wartungsphase

Qualitätsanforderungen an Software

Die **Qualität von Softwareprodukten** ist ein wichtiges Thema im Bereich der Softwareentwicklung. Wichtig dabei ist, dass der Qualitätsbegriff immer im engen Zusammenhang mit den **entstandenen Kosten** und der **benötigten Zeit für die Entwicklung** zu sehen ist.

Die **Qualitätsanforderungen** lassen sich dabei aus zwei Perspektiven beschreiben - aus der Sicht des Anwenders, der Users, und aus der Sicht des Entwicklers.

Qualitätsanforderungen Anwendersicht

- Erfüllung der erwarteten Funktionen
- Zuverlässigkeit / Effizienz
- Benutzbarkeit / Usability
- Sicherheit / Stabilität

Qualitätsanforderungen Entwicklersicht

- Möglichkeit der Erweiterungen
- Einfache Wartung
- Wiederverwendbarkeit
- Übertragbarkeit auf andere Projekte

Vorgehensmodelle

Vorgehensmodelle geben im Software Engineering den Rahmen vor, in welchem der Entwicklungsprozess organisiert wird.

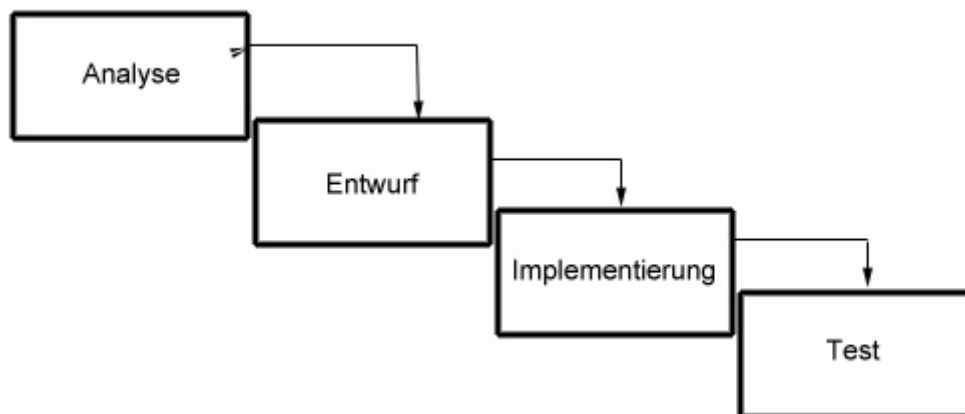
Dabei werden **alle Tätigkeiten**, die bei der Entwicklung einer Software nötig sind, **in einzelne Phasen zerlegt**. Es gibt **verschiedene**, gängige **Vorgehensmodelle**, die hier im Einzelnen vorgestellt und beschrieben werden.

Wasserfallmodell

Das Wasserfallmodell stellt in der Softwareentwicklung die einzelnen Phasen in Form eines Wasserfalls dar. Das Modell zeichnet sich durch einen sequenziellen Charakter aus - es besteht nicht die Möglichkeit, auf eine vorhergehende Phase zurückzugreifen.

Beim erweiterten Wasserfallmodell wurde diese Einschränkung berücksichtigt und korrigiert: Es ist eine Rückkehr in vorhergehende Phasen möglich. (iteratives Phasenmodell)

Folgende Phasen beinhaltet das Wasserfallmodell: Analyse, Entwurf, Implementierung, Test.



Vorteile vom Wasserfallmodell

- Der komplette Entwicklungsprozess wird in Phasen eingeteilt. Dies hat zum Vorteil, dass die Ergebnisse jeder einzelnen Phase getestet werden können.

Nachteile vom Wasserfallmodell

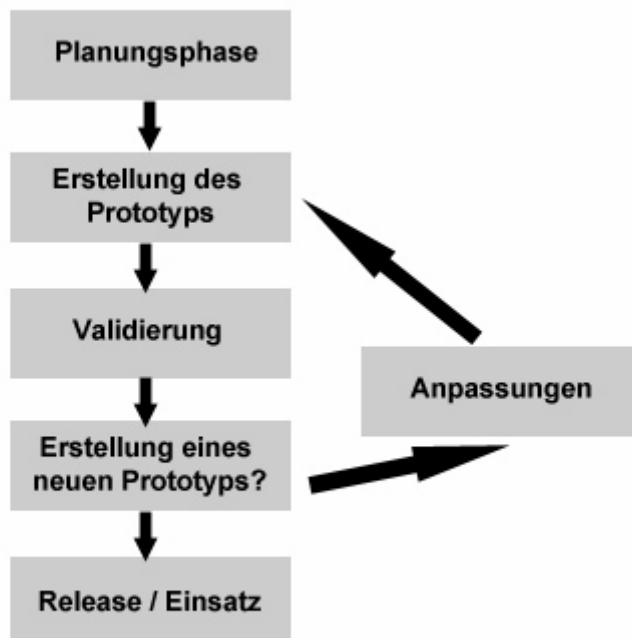
- **Ergebnisse sind erst nach dem Ende einer Phase einsehbar.** Falls die Ergebnisse nicht zufrieden stellend sind, muss eventuell die komplette Phase wiederholt werden.
- **Keine Unterstützung von parallelen Aufgaben**
- Im Wasserfallmodell werden **keine Verantwortlichen** definiert
- **Keine iterative Entwicklung:** Keine Zwischenprodukte, sondern nur ein Produkt am Ende der Entwicklung. Somit kann der Auftraggeber nicht zwischenzeitig eingreifen und erst über das komplett fertige Produkt urteilen.
- Fehlende **Richtlinien** für die **Dokumentation.**

Rapid Prototyping

Mit dem **Rapid Prototyping Vorgehensmodell** kann schnell ein einsatzbereites Softwaresystem erstellt werden, welches im Wesentlichen bereits den endgültigen Funktionsumfang besitzt.

Dies hat den **Vorteil**, dass der **Auftraggeber frühzeitig das System testen kann**. Somit können auch im Entwicklungsprozess noch Verbesserungsvorschläge usw. eingebracht werden.

Ein wichtiger Bestandteil des Rapid Prototyping Modell ist die Erstellung eines **Prototypen** - dieser wird so oft verbessert und angepasst, bis alle Anforderungen komplett erfüllt sind.



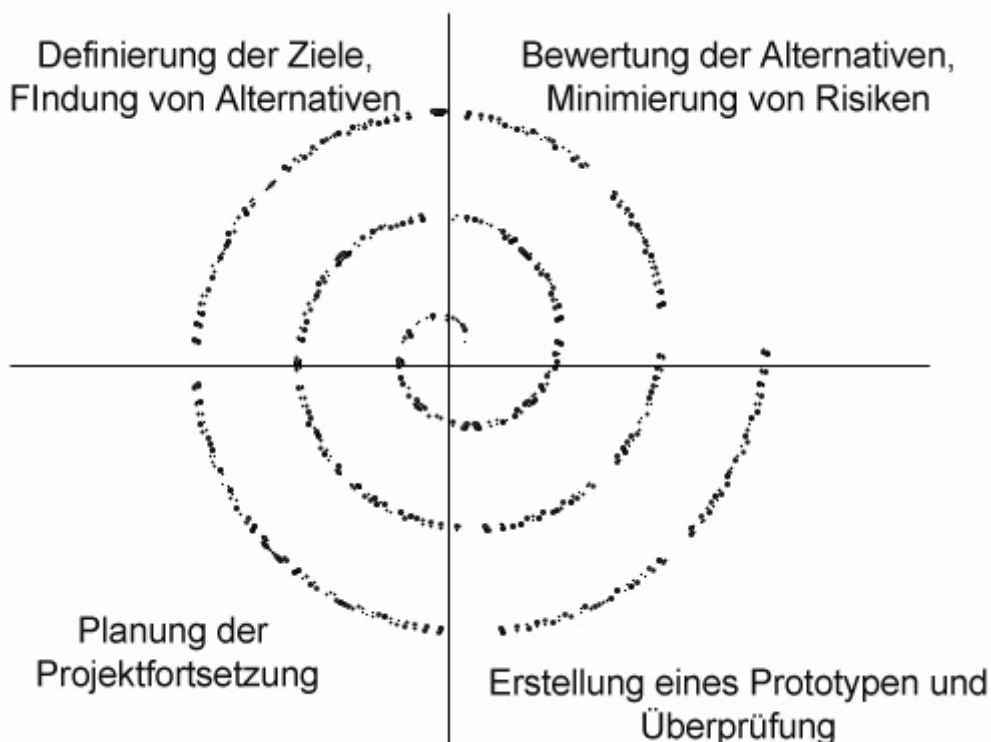
Spiralmodell

Das Spiralmodell ist ein Vorgehensmodell, welches von B. W. Boehm entwickelt worden ist. Bei dem Spiralmodell wird davon ausgegangen, dass es sich bei der Softwareentwicklung um einen iterativen Zyklus handelt.

Das Spiralmodell ist in **4 Quadranten** aufgeteilt. Nach jeder Windung der Spirale ist ein Zyklus beendet. **Jeder Zyklus enthält folgende Aktivitäten:**

- Festlegung von Zielen und Rahmenbedingungen
- Evaluierung der Alternativen / Risikoerkennung
- Überprüfung des Zwischenproduktes
- Planung der Projektfortsetzung

Es empfiehlt sich, nach jeder Windung (also nach jeder Phase) einen Prototypen zu erstellen. **Ziel: Reduzierung des Entwicklungsrisikos.**



Vorteile

- **Iterative Entwicklung:** Veränderungen der Anforderungen, neue Kundenwünsche usw. können noch während der Entwicklung berücksichtigt werden
- **Risikoanalyse:** Risiken lassen sich identifizieren, bewerten und beseitigen.

Nachteile

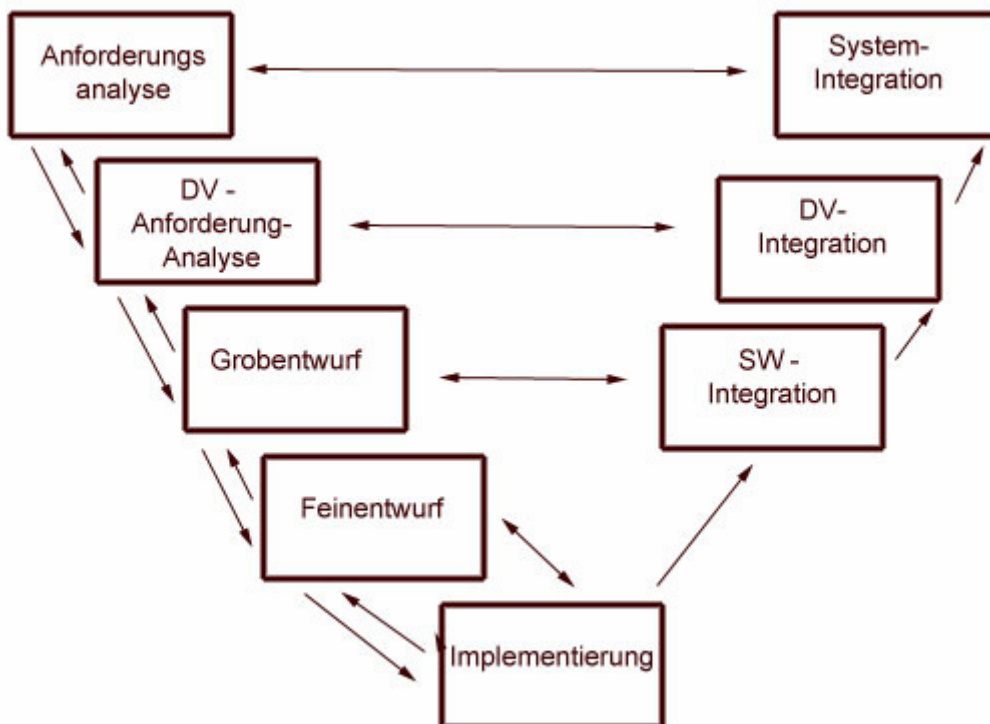
- Verantwortliche sind nicht definiert (z. B. Projektleiter)
- Das Spiralmodell berücksichtigt keine parallelen Aktivitäten

V-Modell

Das V-Modell stammt aus den 80iger Jahren und wurde vom Bundesministerium für Verteidigung entworfen. Das Besondere am V-Modell ist, dass es die Softwareentwicklung aus technisch-funktionaler Sicht betrachtet. Es ähnelt dem Wasserfallmodell - erlaubt im Gegensatz zu diesem jedoch die Rückkopplung auf die vorhergehende Phase. Eine weitere, wichtige Erweiterung sind die Qualitätssicherungsmaßnahmen.

Tätigkeitsbereiche des V-Modells

- Software-Erstellung
- Qualitätssicherung
- Konfigurationsmanagement
- Projektmanagement



Vorteile

- Vollständigkeit (alle vier Tätigkeitsbereiche - siehe oben - werden berücksichtigt)
- Hilfestellungen - das V-Modell bietet Anleitungen und Empfehlungen zur Lösung bestimmter Probleme.
- Hohe Akzeptanz durch Standardisierung

Rational Unified Process

Das Rational Unified Process Vorgehensmodell (kurz: RUP) wurde speziell für die objektorientierte Softwareentwicklung entworfen. Es basiert auf der Unified Modeling Language (UML).

Phasen des RUP

- Konzeptphase
- Spezifikationsphase
- Konstruktionsphase
- Einführungsphase

Kernprozesse

- Geschäftsprozessmodellierung
- Anforderungsanalyse
- Softwarespezifikation und Entwurf
- Implementierung
- Test
- Inbetriebnahme

Übersicht Modellierungsmethoden

Um Softwaresystem zu modellieren, gibt es verschiedenen Methoden. Man unterscheidet in **strukturierten, datenorientierte und objektorientierte Methoden**.

Strukturierte Methoden

Strukturierte Methoden unterteilt man in folgende Bereiche:

- **Strukturierte Analyse:** beinhaltet Funktionsmodelle, Datenflussmodelle, Prozessspezifikation und das Data Dictionary. Grundsätzlich stellt sich in der strukturierten Analyse folgende Frage: **Was soll das geplante Software System können?**
- **Strukturiertes Design:** beinhaltet Operationsmodell und Modulmodelle. Im strukturierten Design stellt sich dann folgende Frage: **Wie sollen die Funktionalitäten des Softwaresystems umgesetzt werden?**

Datenorientierte Methoden

Das ER-Modell und das Data Dictionary bilden die Grundlagen der datenorientierten Methoden.

Objektorientierte Methoden

Folgende, auf UML basierende Diagramme, bilden die Grundlagen für **objektorientierte Methoden**:

- Anwendungsfalldiagramm
- Komponentendiagramm
- Einsatzdiagramm
- Kollaborationsdiagramm
- Zustandsdiagramm
- Aktivitätsdiagramm
- Klassendiagramm
- Sequenzdiagramm

Strukturierte Analyse

Die Strukturierte Analyse ist ein Bestandteil der strukturierten Methoden. Folgende Punkte bilden ihre Grundlage: **Funktionsmodellierung, Prozessspezifikation, Datenflussmodellierung, Data Dictionary.**

Funktionsmodellierung

Bei der Funktionsmodellierung (auch geläufig unter dem Begriff funktionale Dekomposition) werden die Ein- und Ausgabevorgänge eines Software-System untersucht und dargestellt.

Die Funktionsmodellierung liefert einen Überblick über die Struktur des Systems. Die Darstellung dieser Struktur erfolgt über ein Baumdiagramm. In dem Baumdiagramm werden die einzelnen Funktionen Schritt für Schritt aufgeschlüsselt und verfeinert.

Ein Nachteil bei dieser Form der Modellierung ist, dass sie kaum auf konkrete Details eingeht und Daten beispielsweise kaum berücksichtigt werden.

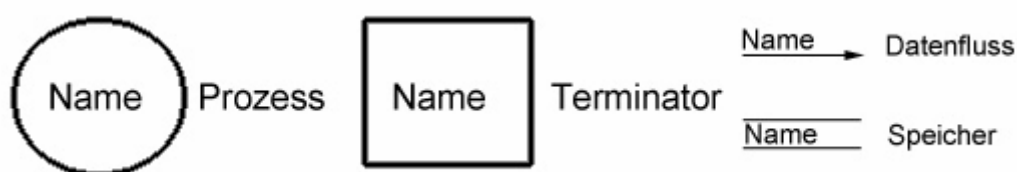


- Untersucht die **externen Schnittstellen** eines Systems
- liefert Überblick über die **Struktur des Systems**
- Funktionen werden in einem Baumdiagramm dargestellt und zerlegt - nach dem **Top-Down-Prinzip**
- **Nachteil:** Datenflüsse werden nur wenig berücksichtigt

Datenflussmodellierung

Die Datenflussmodellierung zählt zu den **wichtigsten Modellierungsmethoden** im Bereich der Software-Entwicklung. Bei der Datenflussmodellierung werden Datenflüsse berücksichtigt, was beim Funktionsmodell nicht der Fall ist.

Ein wichtiger Bestandteil der Datenflussmodelle ist das **Datenflussdiagramm**, welches zur grafischen Darstellung der Datenflüsse dient. Ein Datenflussdiagramm baut auf folgenden, vier Symbolen auf:

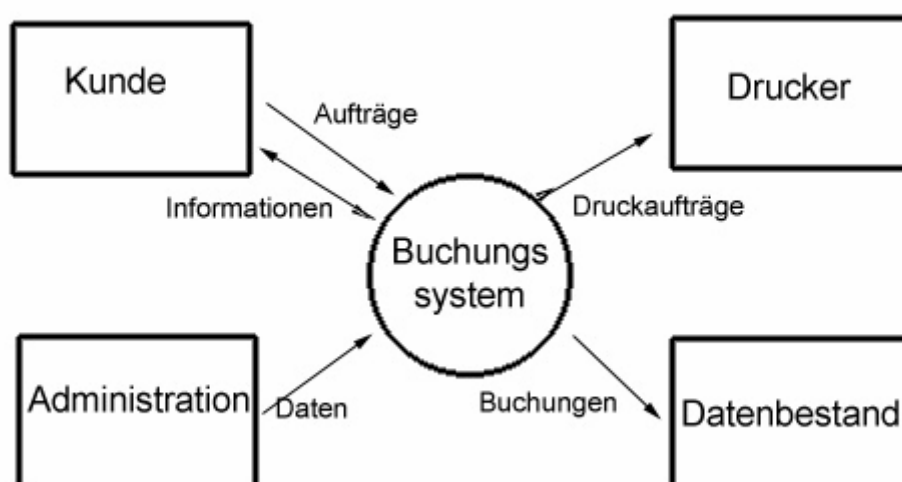


Eine wichtige Eigenschaft der Datenflussmodellierung ist, dass nicht berücksichtigt wird, unter welchen Umständen bestimmte Prozesse aktiviert werden. Auch findet der zeitliche Aspekt keine Berücksichtigung.

Das Kontextdiagramm

Bei dem **Kontextdiagramm** handelt es sich in der Datenflussmodellierung um ein Diagramm, das die gesamte Funktionalität eines Systems zusammenfasst. Es steht hierarchisch an oberster Stelle. Das Kontextdiagramm hat also das Ziel, einen Überblick darüber zu geben, was das System im groben kann und mit welchen Schnittstellen es kommuniziert. (= Darstellung der Außenbeziehung).

Datenflussmodellierung



Dieses Diagramm wird im nächsten Schritt nach und nach verfeinert. Jeder Prozess wird in einem eigenen Datenflussdiagramm dargestellt - diese Verfeinerung wird solange durchgeführt, bis man eine untere Ebene erreicht hat, auf welcher die weitere Verfeinerung der Prozesse keinen Sinn macht.

Prozessspezifikation

Datenflussdiagramme liefern keine Informationen darüber, was ein einzelner Prozess leisten soll. Deswegen wird in der Prozessspezifikation jeder Prozess präzisiert - genauer gesagt wird beschrieben, wie Eingaben in Ausgaben transformiert werden. Es stellt sich also grundsätzlich folgende Frage in der Prozessspezifikation: Was muss getan werden, um eine Eingabe in eine Ausgabe zu transformieren?

Der Begriff der Prozessspezifikation ist auch als **MiniSpec** oder **PSPEC** geläufig.

- Flowcharts
- Nassi-Shneidermann-Diagramme
- Entscheidungstabellen
- Strukturierte Sprache (= Gleichgewicht zwischen natürlicher Sprache und der strukturierten Programmierung)

Data Dictionary

Ein Data Dictionary - zu deutsch Datenkatalog / Wörterbuch - ist ein Verzeichnis, in welchem die Daten, welche in dem zu entwickelnden Software-System vorkommen, beschrieben werden. Ziel des Data Dictionary ist es, eine redundanzfreie Definition der Datenstruktur herzustellen.

Datenstruktur und Datenelemente werden im Data Dictionary in einer modifizierten Backus-Naur-Form beschrieben.

- Kunde = Name + Vorname
- Name = {Buchstabe}
- Vorname = {Buchstabe}
- Telefon = {Ziffer}
- Datum = Tag + Monat + Jahr
- Ziffer = [0|1|2|3|...|9]
- Buchstabe = [a|b|...|z|A|B|...|Z]

Strukturiertes Design

Während es in der Strukturierten Analyse darum ging, zu beschreiben, was das zukünftige Software-System leisten soll, geht es im **Strukturierten Design** darum, wie diese Anforderungen umgesetzt werden.

- **Entwicklung der Systemarchitektur** unter Berücksichtigung sowohl technischer (Hardwarevoraussetzungen z. B) als auch wirtschaftlicher Aspekte (Kosten).
- Den **Prozessen**, die festgelegt wurden, werden **Hardwarekomponenten zugeordnet**.
- **Arbeitsteilige Entwicklung** der Software wird geplant
- Wichtig: Im Strukturierten Design wird **programmiersprachenunabhängig** gearbeitet

Vorgehensweise

- Zerlegung des Systems in einzelne Komponenten (Pakete, Klassen, Module)
- Definierung und Festlegung der Schnittstellen
- Entwurf algorithmischer Strukturen

Modularisierung

Bei der Modularisierung wird ein System in einzelne Komponenten und Module zerlegt.

- Dabei wird die Komplexität der Beziehungen der System-Komponenten als Kopplung bezeichnet
- Die Komplexität der Komponenten an sich wird als Kohäsion bezeichnet
- Daraus folgt, dass sich eine **gute Modularisierung durch eine geringe Kopplung und eine hohe Kohäsion auszeichnet**.

Ein Modul ist wie folgt untergliedert:

- Modulname
- Schnittstellenspezifikation
- Modulrumpf

Modulname

Der Name des Moduls wird festgelegt.

Schnittstellenspezifikation

Hier wird beschrieben, **was das Modul leistet**, welche Funktion es erfüllt. Allerdings wird in der Schnittstellenspezifikation nicht festgelegt, **wie** diese Funktion umgesetzt wird.

Modulrumpf

Im Modulrumpf wird beschrieben, **wie die Funktion umgesetzt**. Hierbei handelt es sich also um den wichtigsten Teil des Moduls. Der Modulrumpf ist durch Zugriffe von Außen geschützt.

Kohäsion

Kohäsion klingt zwar wie eine Krankheit, ist aber ein Begriff aus dem Software-Engineering. Kohäsion bedeutet: **innere Bindung der Funktionen, Anweisung einer Komponente**.

Man könnte auch sagen, dass Kohäsion die Stärke der Bindung von Funktionen einer Komponente beschreibt. ("Grad der funktionalen Zusammengehörigkeit")

Wodurch zeichnet sich optimales Systemdesign aus?

- **Maximierung der Kohäsion** - Funktionen einer Komponente sollten eine starke Bindung haben
- **keine Aufteilung von Funktionen** auf mehreren Komponenten
- **keine Zusammenfassung** nicht zusammengehöriger Komponenten

Kohäsion - normale Bindung

Man unterscheidet die Kohäsion nach verschiedenen Arten. Die erste Art der Kohäsion ist die normale Bindung. Bei der normalen Bindung enthält jedes Modul nur Funktionen, die inhaltlich zusammenpassen und mit gemeinsamen Daten arbeiten.

Arten der Normalen Bindung

Funktionale Bindung

- Das Modul enthält nur Elemente, die eine einzige Funktion ausführen
- Modul mit funktionaler Bindung ist oft erkennbar an seinem aussagekräftigen Namen
- Modul verfolgt einen klaren Zweck

Sequentielle Bindung

- Das Modul enthält **eine Folge von Aktivitäten**, die sequentiell abgearbeitet werden.

- Modul verfügt über eine gute Kopplung und ist leicht wartbar
- **Nachteil:** schlechte Wiederverwendbarkeit

Kommunizierende Bindung

- Das Modul führt **verschiedene Funktionen** aus, die alle die selben Ausgabe- und Eingabeparameter nutzen
- Die Funktionen stehen jedoch **nicht im funktionalen Zusammenhang**.

Weitere Bindungsarten

Prozedurale Bindung

- Modul enthält **verschiedene Funktionen**, bei denen **Kontrolldaten** von Funktion zu Funktion weitergegeben werden.
- Es sind kaum Beziehungen zwischen Eingangs- und Ausgangsparametern vorhanden

Temporäre Bindung

- Modul enthält Aktivitäten, die **zeitlich unabhängig voneinander** sind.
- Klassische Beispiel für eine temporäre Bindung: Initialisierung einer Variablen

Logische Bindung

- Das Modul enthält Funktionen, die fest zusammenhängen und einen logischen Zusammenhalt bilden.
- Gefahr bei der logischen Bindung: Programmteile werden untrennbar miteinander verbunden, wodurch die Wartung des Softwaresystems sehr erschwert wird.

Zufällige Bindung

- Die Aktivitäten / **Funktionen** stehen in **keiner Beziehung** zueinander ("zufällig").
- Keine Verbundenheit durch Datenflüsse oder Kontrollflüsse
- Hierbei handelt es sich um eine Bindungsart, die noch schlechter ist als die logische Bindung.

Kopplung

Die Kopplung bezeichnet in der Softwareentwicklung beziehungsweise in der Software-Technik den Grad der Abhängigkeit der miteinander verbundenen Komponenten (z. B. Module).

Arten der Kopplung

Die Kopplung wird nach verschiedenen Arten unterschieden. Die wichtigsten Kopplungsarten sowie ihre Vorteile / Nachteile sind hier aufgelistet.

Datenkopplung

Nachteile::

- Bei einer zu hohen Anzahl an **Übergabeparametern** leidet die Übersichtlichkeit. Daher sollte ein Modul nicht mehr als 7 Übergabeparameter haben.
- **Tramp-Daten** - Daten, die ziellos im System herumirren. Sie stellen für viele Module nur eine Belastung dar und erzeugen zusätzlichen Programmcode. Tramp-Daten stehen für eine schlechte Organisation.

Datenstrukturkopplung

- Ein Modul übergibt dem anderen Modul eine **zusammengesetzte Datenstruktur**
- **Beispiel für Datenstruktur:** Kundensatz (setzt sich zusammen aus Name, Straße, Telefon usw.)
- Bei der Datenstrukturkopplung sollte darauf geachtet werden, dass die Datenstruktur nur Daten beinhaltet, die das andere Modul auch wirklich benötigt.

Kontrollkopplung

- Eine Kontrollkopplung liegt vor, wenn ein Modul an das andere Modul ein **Datenelement übergibt**, das die **interne Logik des Moduls beeinflusst**.
- Diese Datenelemente werden **Kontrollelementen – oder der auch Flags** - genannt.

Nachteile:

- Die Unabhängigkeit der Module wird beeinträchtigt - ein Modul ist keine Black-Box mehr, denn ein anderes Modul kennt genau das Innere des Modul und beeinflusst dieses.