Prinzipien

- abstraktion (modellbildung, vernachlässigung von details); 3 Merkmale:
  - Abbildungsmerkmal: Bildet reelles/fiktives ab => Realitätsbezug
  - Verkürzungsmerkmal: hebt Wesentliches hervor, lässt Unwesentliches weg
  - Pragmatisches Merkmal: Kann unter bestimmten Bedingungen original ersetzen
- Strukturierung: komplex => reduzierte Darstellung mit gleichem character und spezifischen Merkmalen
- Dekomposition (Stepwise Refinement, Zerteilung eines problems)
- Hierarchisierung (Erstellen einer Rangordnung => Hierarchieebenen; Spezialfall Strukturierung)
- Wiederverwendung (technische,organisatorische Vorraussetzungen; $\tilde{6}0\%$ Mehraufwand)
- Standardisierung (Festlegung/Einhaltung von Richtlinien, Normen)
- Verbalisierung (Ideen geeignet in Worten ausdrücken, z.B. Namen)
- Modularisierung (Zusammenbau des Systems aus Bausteinen s.U.)

Modularität

- hohe Kohäsion (Abhängigkeiten im Modul,eine Verantwortung)
- lose Kopplung (Abhängigkeiten zwischen modulen)
- Information Hiding (Geheimnisprinzip)
- Schnittstellenspezifikation
- Kontextunabhängigkeit (Analog zu lose Kopplung zur Umgebung
- Lokalitätsprinzip (Alle Bestandteile zu Problemlösung an einer Stelle)

=> Änderungsfreundlichkeit, Wartbarkeit, Standardisierung, Arbeits- Organisation und Planung, Überprüfbarkeit

Modulare Operatoren:

- Splitting: Aufteilen eines Moduls
- Substituting: Austausch von Modul durch Modul gleicher Funktion
- Augmenting: Hinzufügen von Modul für Funktionserweiterung (leicht)
- Excluding: Entfernen von Modul für Funktionsverringerung (schwer)
- Inverting: Herausziehen von redundanter Funktionalität aus Modulen in höheres Modul
- Porting: Verwenden eines Translators für ein Modul in nicht vorhergesehenem Kontext

Architectural Erosion: Verletzungen der Architektur. Änderungen => inkonsistenter Code (entfernen tragender Säulen).
Architectural Drift: Erweiterung ohne Beachtung der bestehenden Architektur (Anbauten).
Gegenmaßnahmen: Dokumentation, Open-Closed-Prinzip, Kapselung, Modularisierung

Strukturparadigmen

Orhogonal zu Programmierparadigmen

- Unstrukturiert (Spagetti-code, goto)
- Strukturiert (Blöcke (z.B. if))
- Modular (sammlung von Typen/Prozeduren, explizite schnittstellen)
- Objektorientierung (Kapselung von Daten/Methoden in Objekten)

Objektorientierung

Prinzipien eines wiederverwendbaren OO Designs:

- Program to an Interface, not an Implementation
- Favor delegation over class inheritance

Vererbung:
Subtyping: Vererbung von ähnlichem Verhalten (Signaturen) => Generalisierung/Spezialisierung und Realisierung
Subclassing: Vererbung von Implementierungen => nur Generalisierung/Spezialisierung

Liskovsches Substitutionsprinzip:
S1: Subtypen haben gleiche Operationen mit kompatibler Signatur (Compilergeprüft)
S2: Subtypen zeigen das gleiche Verhalten (Programmiergeprüft)
=> Design-By-Contract: Subtypen dürfen Precondition (Anfang der Methode/Parameter) schwächen (OR verknüpfung), und/oder Postconditions (Ende der Methode/Rückgaben) stärken (AND Verknüpfung) => Obermenge der gültigen Eingaben, Teilmenge der gültigen Ausgaben.

Ko- Kontravariant: Szenario: Unterklasse überschreibt Methode in Oberklasse mit anderen Typen in Ein- Ausgabewerten.

	Parametertyp in Unterklasse	Parametertyp	Rückgabetyp
kovariant	ist Subtyp des Parameters in Oberklasse	unsicher	sicher
kontravariant	ist Obertyp des Parameters in Oberklasse	sicher	unsicher

Delegation: siehe Design Pattern Strategy

Law of Demeter/Feature Envy: Empfehlung: Klassen sollten nur mit direkt gekoppelten Klassen arbeiten, nicht mit indirekt verknüpften

A → B → C => A ruft methoden von B auf, nicht von C

high Fan-In, lowFan-Out: Fan-in: Anzahl der Module, die das Modul verwenden (tolerabel) Fan-out: Anzahl der verwendeten Module (möglichst klein)

Open-Close Prinzip: Offen für Erweiterung, geschlossen für Veränderungen

Architekturmuster

Vorraussetzung: Modularität

Model-View-Controller: Darstellung von variablen Daten in verschiedenen Sichten

Leichte Änderbarkeit der Views => Unabhängig vom Kern implementiert;

Direke Anpassung der Sichten bei Änderungen

Lösung: Modularisierung:

Model: Kernfunktionalität und Datenmodell; registriert und informiert View u. Controller

View: Darstellung der vom Model abgefragten Daten

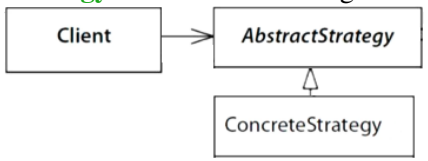
Controller: Auswertung Benutzereingaben => Aufruf von Model bzw. Viewfunktionen

Bei Implementierung als Observer Pattern: Observable->Model, Observer->View,Controller

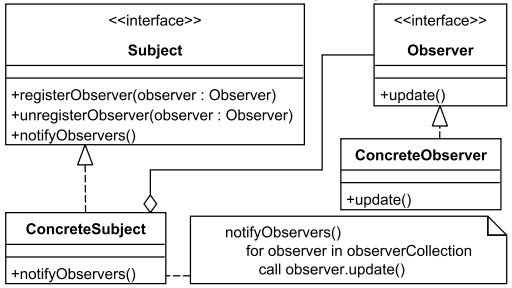
Design Pattern

Vorraussetzung: Objektorientierung

Strategy: Austauschbare Algorithmen



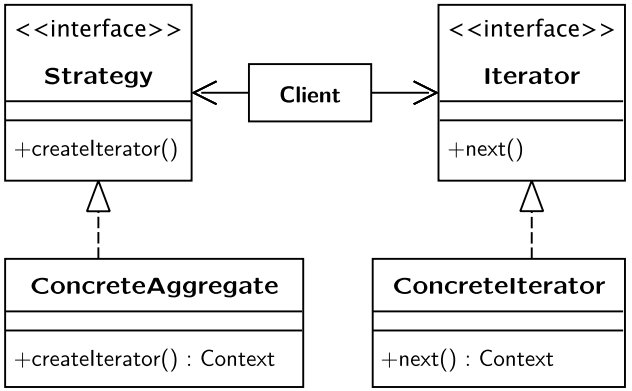
Observer: Informieren von Objekten über Zustandsänderungen eines Objektes

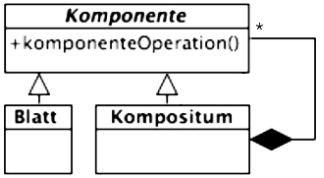


Varianten:

Pull: keine information im notify Push: Zustandsänderung wird mitgeschickt

Iterator: Sequentieller Zugriff auf Elemente eines Aggregats ohne interna preiszugeben.





Composite:

Programmieridiome

Vorraussetzung: Konkrete Sprache

Mixin: Erweitert bestehende Klasse ohne Vererbung zu verändern

ohne Mehrfachvererbung, mit Mehrfachvererbung