Softwaresysteme für Großprojekte

Basierend auf der Zusammenfassung von Julian Kotzur

Beherrschung von Software-Fehlern

Arten der Software-Inkorrektheit:
– Irrtum - mistake
  • Mentaler Fehler des/der Entwickler
  • Führt stets zu einem oder mehreren Fehlern
  • z.B. Denkfehler, fehlendes Wissen, falsche Formel
– Produktfehler - fault
  • Abweichung zw. geplanten und realisiertem Produkt
  • Kann zu fehlerhaften Zustand / Versagen führen
  • z.B. falsche/r Operanden / Operationen / Code
– Fehlerhafter Zustand - error
  • Abweichung zw. geplantem und realisiertem Zustand
  • Kann zu Versagen führen
  • z.B. inkorrektes Zwischenergebnis
– Versagen - failure
  • Abweichung zw. geplantem und realisiertem Verhalten
  • z.B. falsches/kein Ergebnis

Maßnahmen zur Fehlerbeherrschung:
– kontrolliertes, durchgängiges, transparentes Vorgehen
– Qualitätssicherung vor Verlassen jeder Prozessphase
– Redundante Maßnahmen zur Fehlertolerierung

Softwareprozess:
Abfolge von Aktivitäten und daraus resultierenden Ergebnissen, die zur Herstellung eines Softwareproduktes führen.

Prozessmodell:
Vereinfachte Beschreibung eines Softwareprozesses

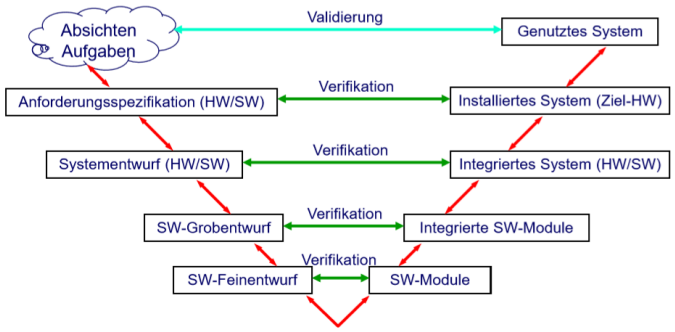
Vorgehensmodelle

Build-and-Fix Modell (agiles Vorgehen):
– chaotische Vorgehensweise ohne Lebenszyklus
– Für kleine Projekte/Teams, ungeeignet für große

Software-Lebenszyklus:
– Anforderungsphase: Bestimmung "Was wird gebraucht"
– Spezifikationsphase: Dokumentation der Anforderungen
– Entwurfsphase: Umsetzungsplanung
– Implementationsphase: Software wird geschrieben
– Integrationsphase: Software wird Zusammengesetzt
– Intallations-, Nutzungs-, Wartungs-, Ablösungsphase

Wasserfall-Modell:
– Wasserfallartige Abarbeitung der Lebenszyklusphasen
– Vorteil: Meilenstein und Dokumentation
– Nachteil: Fehler kann sich in Kette fortsetzen

V-Modell:
– Wasserfallsmodellerweiterung bei Qualitätssicherung
  • Verifikation (are we doing things right?)
  • Validierung (are we doing right things)



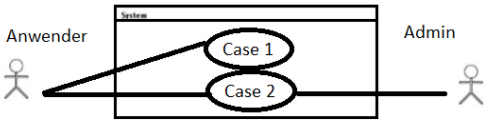
Allgemein: Anforderung - requirement

– Softwareanforderung: Muss von Software erfüllt werden
– Funktionale Anforderung: erwünschtes Verhalten
– Nicht-Funktionale Anforderung: Einschränkung

Eigenschaften:
– Vollständigkeit
  • Gewünschte Reaktionen für alle Eingaben klar definiert
– Konsistenz
  • widerspruchsfrei bzgl. sich und anderen Anforderungen
– Korrektheit
  • Soll der Absicht des Auftraggebers entsprechen
– Eindeutigkeit
  • Darf nur auf eine Art u. Weise interpretiert werden
– Realisierbarkeit
  • Beachtung der Einschränkungen(Budget,Hardware,etc.)
– Verfolgbarkeit
  • Eindeutig identifizierbar, um sie in Software zu finden
– Nachweisbarkeit
  • Eindeutige Kriterien zur Überprüfung ihrer Erfüllung

Anforderungsermittlung - Vorgehensweise

1. Anwendungsfallmodellierung (Use-Case):
– include: A – B, A beinhaltet B
– extends: A – B, A erweitert B



2. Festhalten der Anforderungen (Volere-Karte):
– Anforderung enthält:
  • Nummerierung / Motivation / Urheber
  • Abnahmekriterien / Kundenzufriedenheit / Konflikte
  • Abhängigkeiten / Probleme / Unterlagen / Historie

Anforderungsverwaltung

Änderungsprozess für Anforderungen:
– Vorprüfung – Auswirkungsanalyse – Durchführung

Spezifikationssprachen:
– Informell: z.b. Deutsch / Englisch
  • Vorteil: leicht zu lesen / schreiben
  • Nachteil: unübersichtlich / schwer zu prüfen
– Semi-formell: enthält Elemente wohldefinierter Semantik
  • Kompromiss zw. informeller und formeller Sprache
  • Beispiele: ER-Diagramm, Tabelle
– Formell: komplett wohldefinierte Semantik
  • Vorteile: eindeutig / widerspruchsfreiheit prüfbar
  • Nachteile: kompliziert und aufwendig
  • Beispiele: Zustandsautomat / Petri-Netze / OCL

Object Constraint Language (OCL):
– Vorteile:
  • Für Schnittstellenbeschreibung gut
  • Konsistenzeigenschaften statisch analysierbar
– Nachteile:
  • Rein datenbasiert und nicht zeitbezogen
– Verwendete Datentypen und Operationen:

Typ	Operationen
Integer	*,+,-,/, =, <, >, <=, >=, <>, …
Real	*,+,-,/, =, <, >, <=, >=, <>, …
Boolean	and, or, not, implies, if-then-else, …
String	toUpper, concat, …

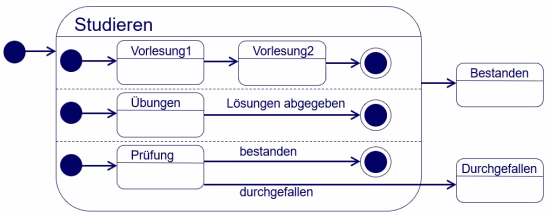
– Kontext Klasse:
  • context Klassenname

– Kontext Methode:
  • context Klasse::Methode(Param1:Type1,...):Returntype

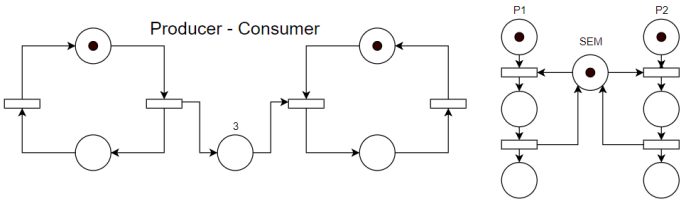
OCL - Schlüsselwörter:
– Momentaner Kontext: "self"
  • Beispiel: context Person / self.alter
– Invariante: "inv"
  • Muss wahr sein, damit Objekt wohldefiniert
  • Beispiel: inv: alter > 0
– Vorbedingung: "pre"
  • Muss zum Zeitpunkt des Opertionsaufrufs wahr sein
  • Beispiel: pre: alter > 18
– Nachbedingung: "post"
  • Muss nach Operationsausführung wahr sein
  • Beispiel: post: alter > 18
– Result: "result"
  • Bezeichnet Rückgabetyp einer Operation
  • Beispiel: post: result = a / b
– Initialisierung: "init"
  • initialisierung von Attributen
  • Beispiel: context Person::alter : Int / init: 0
– let ... in ...
  • Hilfsmittel für Hilfsvariablen
  • Beispiel: let area = r*r*PI in result = area * h

OCL - Collections:
– Fasst mehrere Objekte zusammen
– Entsteht aus einer 1 : N Verbindung
– select - Operation:
  • Auswählen bestimmter Objekte aus Collection
  • Syntax: collecion->select(v:Type|boolean - expression)
– forAll - Operation:
  • Einschränkung über alle Objekte
  • Syntax: collection->forAll(v:Type | bool - expression)
– exists - Operation
  • Einschränkung für einzelne Objekte
  • Syntax: collection->exists(v:Type | bool - expression)
– size() - Operation
  • Liefert die Größe einer Collection
  • Syntax: collection->size()
– Weitere: allInstances(), includes(A), isEmpty()

Zustandsautomat:
– Pro: Eindeutig und Intuitiv
– Contra: Kann Unübersichtlich werden
– Beispiel: A erst wieder nach Erreichen von D UND Y
– Parallel in Zustandsautomat:
  • 1. Alle Superpositionen zw. Kammern bilden
  • 2. Pro Superposition alle Eingabe testen
  • 2. Eingabeveränderung einzeln pro Element möglich



Petri-Netze:
– Bestandteile:
  • Kreis: Platz für Tokens (Punkte)
  • Pfeile: Eingangskante / Ausganskante
  • Rechteck: Transition
– Schaltregeln:
  • Transition feuerbar, wenn Eingangsgewichte abgedeckt
  • Tokens wandern Eingangsstellen zu Ausgangsstellen
– Erreichbarkeitsgraph:
  • Entscheidungsbaum bestehend aus Belegungen
  • Äste bilden feuerbare Transitionen
  • Alle möglichen Belegungen sind enthalten



Software-Entwurf

Kopplung

– Grad der Interaktion zw. Komponenten ist entscheident
– Vorteilhaft sind wenig Kopplungen zw. Komponenten
– Von Schlecht zu gut:
– 1. Content coupling (Inhaltskopplung):
  • Sprung oder Änderung von Code
– 2. Common coupling(Globalkopplung):
  • Zugriff auf gemeinsamen Datenbereich
– 3. Control coupling(Kontrollkopplung):
  • Austausch von Steuerparameter für Ablaufsteuerung
– 4. Stamp coupling (Datenstrukturkopplung):
  • Datenstrukturen werden übergeben
  • Nur teile der Datenstruktur werden benötigt
– 5. Data coupling (Datenkopplung):
  • Nur benötigte Daten werden übergeben

Kohäsion

– Von schlecht zu gut:
– Funktionale Bindungen innerhalb einer Komponente
– Vorteilhaft: Möglichst Hohe Kohäsion
– Mehrere Kohäsionen möglich
– Zufällige: Völlig unabhängige Funktionen
– Logisch: Logischer Zusammenhang
– Zeitlich: Zeitlicher Zusammenhang
– Prozedural: Funktionale Reihenfolge
– Kommunikativ: Gemeinsame Datennutzung
– Sequentiell: Folgefunktion braucht Vorherige als Eingabe
– Funktional: Funktionen untrennbar zusammengefasst
– Informational: unabhängiger Code, selbe Datenstruktur

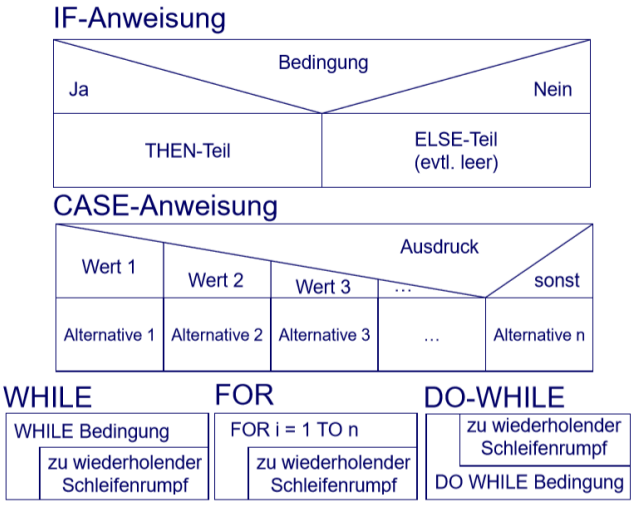
Vorgehen:

Mehrere Funktionen?
- Nein — funktional
- Ja, auf gemeinsamen Daten?
  - Ja, Reihenfolge relevant?
    - Ja — sequentiell
    - Nein — kommunikativ
  - Nein, in zeitlichem Zusammenhang aktiviert?
    - Ja, Reihenfolge relevant?
      - Ja — prozedural
      - Nein — temporal
    - Nein, bilden sie verwandte, alternative Funktionalitäten?
      - Ja — logisch
      - Nein — zufällig

Programmiersprachenneutrale Notation

– Ziel: Ohne Code Code beschreiben
– Vorteil: Programmiersprache kann später gewählt werden
– Pseudocode gehört in diese Kategorie

Strukturprogramm:
– Graphische Darstellung von Kontroll-/Datenfluss



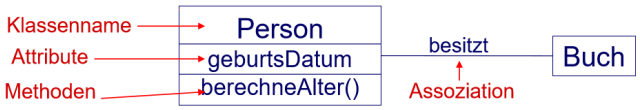
Objektorientierte Analyse

Ziel der Objektorientierung:
Bessere Wiederverwendbarkeit und damit geringere Entwicklungskosten bei der Softwareproduktion
Bestandteile:
– Klasse: Stellt Konstruktoren und Methoden
– Objekte: Instanzen von Klassen
– Vererbung: Prinzip Ober- und Unterklasse
– Polymorphie: Gleicher Name bei mehreren Methoden
– Delegation: Objekt leitet Daten an andere Objekte

Statische Modellierung

Ziel:
– Grober Aufbau eines UML Diagramms
– Identifikation relevanter Klassen
– Beschreibung der Klasseneigenschaften
– Beschreibung der Beziehung zw. Klassen

Klassendiagramm:

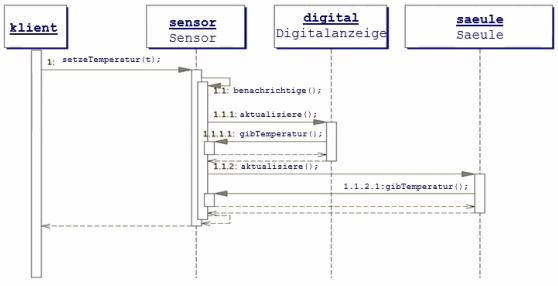


Vorgehensweise:
– 1. Klassenkandidaten identifizieren
– 2. Assoziationen identifizieren
– 3. Attribute spezifizieren

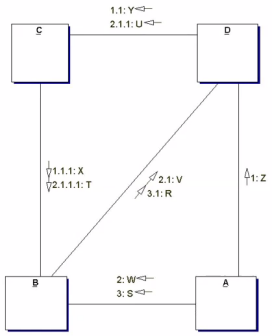
Sichtbarkeit:
– steht vor dem Namen
– public = + / protected = # / private = -

Dynamische Modellierung

Sequenzdiagramm:
Beschreibt zeitlichen Ablauf eines Nachrichtenaustauschs
– Dicke Pfeile:
  • synchrone Nachrichten
– Einfache Pfeile – :
  • Methodenaufrüfe
– Weitere Bestandteile:
  • Objekte: Rechtecke
  • Zeitachse



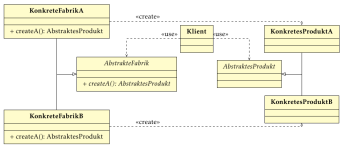
Kommunikationsdiagramm:
Beschreibt statische Verknüpfung zw. interagierenden Objekten
– Nachrichtenaustausch als Verknüpfungskanten
– Zeit mit Nummerierung definiert



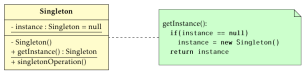
Entwurfsmuster

Erzeugungsmuster

Abstrakte Fabrik:
– System unabhängig von Art der Erzeugung
– System mit einer oder mehrerer Produkfamilien
– Gruppe gemeinsam genutzter Produkte
– Schnittstelle für Klassenbibliothek

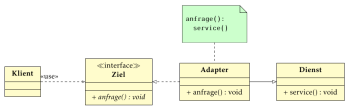


Singleton:
– Nur ein Objekt einer Klasse
– Spezialisierung eines Objekts in Unterklassen

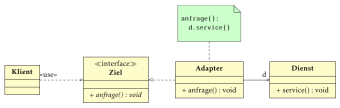


Strukturmuster

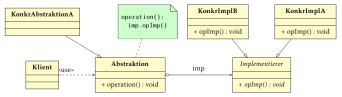
Adapter (klassenbasiert):
– Klassen erben voneinander
– Schnittstellenübersetzung



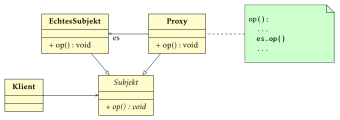
Adapter (objektbasiert):
– Geht immer
– Schnittstellenübersetzung



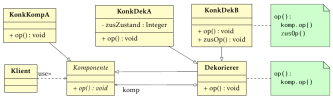
Brücke:
– Zur Laufzeit Auswahl der Implementierung
– Einfache Erweiterbarkeit der Implementierung



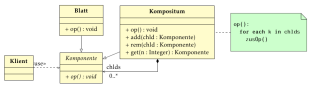
Proxy (objektbasiert):
– Objekt vor Veränderung schützen
– Zugriffsrechte beschränken
– Stellvertreterobjekt



Dekorierer (objektbasiert):
– Hinzufügen / Auswählen von Funktionalitäten zur Laufzeit
– Klassen werden nicht verändert
– Neue Klassen geben Funktionalität

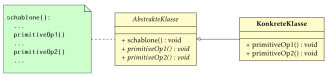


Kompositum (objektbasiert):
– Objekten in Baumstruktur
– z.B. Dateisystem

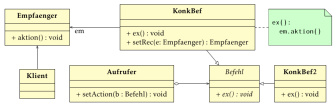


Verhaltensmuster

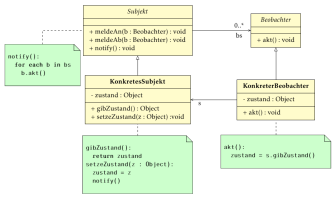
Schablone (klassenbasiert):
– Abstrakte Klasse ohne Implementierung
– Methoden in konkreter Klasse implementiert



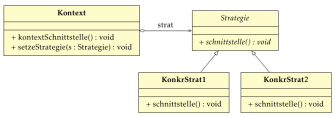
Befehl (objektbasiert):
– Befehle sollten verwaltet werden können
– Befehle werden als Parameter übergeben
– Beispiel: Menüeinträge in Programmen (links oben)



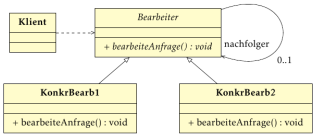
Beobachter (objektbasiert):
– Änderung soll mehrere Objekte beeinflussen
– Mehrere konkrete Beobachter möglich



Strategie (objektbasiert):
– Zur Laufzeit Auswahl des Lösungsverfahrens
– Heuristisches Auswahlverfahren
– Unterschiedliche Aufgabenstellungen



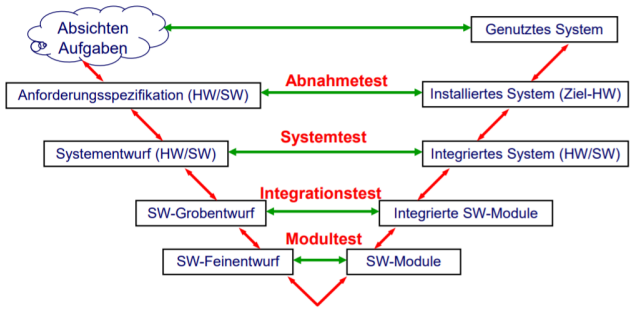
Zuständigkeitskette (objekt):
– Unterschiede Anfragen zum bearbeiten
– Bearbeiter iteriert über KB um zuständige Klasse zu finden



Zusammenfassung

	Erzeugungsmuster	Strukturmuster	Verhaltensmuster
klassenbasiert		Adapter	Schablonenmethode
objektbasiert	Singleton Abstrakte Fabrik	Adapter Brücke Dekorierer Kompositum Proxy	Befehl Beobachter Strategie Zuständigkeitskette

Testen



Teststrategien

– Modultest:
  • Prüfung der einzelnen Module auf richtigkeit
– Integrationstest:
  • Prüfung des Zusammenspiels der Module
– Systemtest:
  • Prüfung der Software auf der Zielhardware
– Abnahmetest:
  • Prüfung des beim Kunden installierten Systems
– Regressionstest:
  • Wiederverwendung von vorhandenen Tests
  • Umschreiben alten Testfall-Codes

– Nicht-inkrementelles Testen:
  • Module einzeln Testen, dann zusammengesetzt testen
  • −: Fehler schwer zu lokalisieren, benötigt viel Zeit
– Inkrementelles Testen:
  • Kombination aus Implementierung und Integration
  • +: Einfache Fehlerlokalisierung und schneller

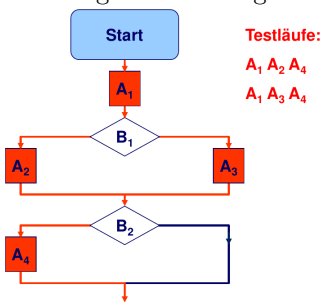
Funktional: Testfälle anhand von Spezifikation
Strukturell: Testfälle anhand von Code-Struktur

Black-Box (Funktional):
– Äquivalenzklassentest:
  • Eingabebereiche, die zu gleichen Ergebnissen führen
– Grenzwerttest:
  • Eingabe an den Grenzen der Äquivalenzklassen
– Cause-Effect-Graphing:
  • Testdaten aufgrund der Ursache-Wirkung-Überlegung
– Selbsterklärend: Error Guessing / Random Testing

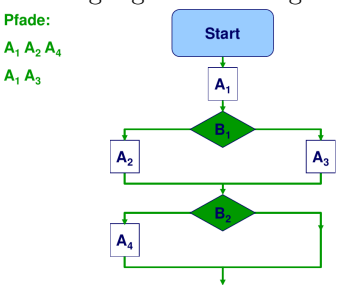
Grey-Box (Strukturell):
– Anweisungsüberdeckung
  • Alle Anweisungen mind. einmal durchlaufen
– Verzweigungsüberdeckung
  • Alle Verzweigungen mind. einmal durchlaufen
– Pfadüberdeckung
  • Alle möglichen Pfade mind. einmal durchlaufen
  • Unmöglich bei Schleifen

White-Box (Strukturell):
– Mehrfachbedingungstest:
  • Große Flags werden atomar getestet
– Grenzwerttest:
  • Grenzen der Verzweigungsbedingungen werden getestet
– Datenflussbasiertes Testen:
  • Betrachtung, wo & wann Variablen geänd./init. werden

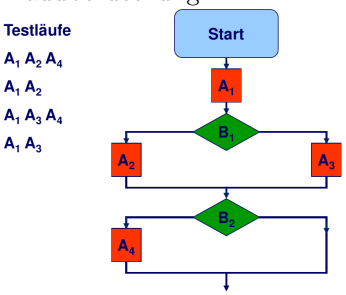
Anweisungsüberdeckung:



Verzweigungsüberdeckung:



Pfadüberdeckung:



Wartung

Wartung:
– Bezieht sich auf in Betrieb befindlicher Software
– 2/3 der Kosten einer Software entstehen durch Wartung
– Wartungsaufgaben:
  • Behebung von Fehlern
  • Anpassung an neue Anforderungen
  • Vorbeugende Maßnahmen ergreifen
  • Anpassung an neue Umgebungen
– Wartungsmanagement:
  • Dokumentation von Korrektur und Änderung
  • Nachvollziebarkeit der Wartungsarbeiten

Refactoring:
– Verbesserung interner Struktur ohne Verhaltensänderung
  • Codeverbesserung (Fehlerfinden)
  • Designverbesserung (Weiterentwicklung)
– Wann man Refactorisieren sollte:
  • Fehlerbehebung
  • Hinzufügen von Funktionalitäten
– Wann man nicht Refactorisieren sollten:
  • Neuschreiben von schlechtem Code
– Refactoring-Technik: Methode extrahieren:
  • Codeabschnitt auf Methode auslagern
  • Übrig bleibt Methodenaufruf
– Refactoring-Technik: Methode integrieren:
  • Gegenteil von Methode extrahieren
– RF-Technik: Methode durch Methodenobjekt ersetzen:
  • Methode wird in ein Objekt umgewandelt
– RF-Technik: Klasse extrahieren:
  • aus einer Klasse werden mehrere Klassen
– RF-Technik: Klasse integrieren:
  • Gegenteil von Klasse extrahieren
– RF-Technik: Delegation verbergen:
  • Klassenschnittstellen zu anderen Klassen verringern
– RF-Technik: Verzweigung zu Polymorphie:
  • Statt If-else, Polymorphie verwenden
– RF-Technik: Methode nach oben schieben
  • Mehrere Unterklassen haben die gleichen Methoden
  • Oberklasse erhält diese Methode
– RF-Technik: Unterklasse extrahieren
– RF-Technik: Oberklasse extrahieren
– RF-Technik: Vererbungstruktur entzerren