Qualitätssicherung

Gliederung

- Konstruktive Qualitätssicherung
  - Richtlinien zur Quelltextgestaltung
    - Refactoring Pattern
    - Bug Pattern
    - Anti Pattern
- Analytische Qualitätssicherung

Qualitätssicherung

- Qualitätssicherung ist ein kontinuierlicher Prozess
- Früher Einsatz der Qualitätssicherung spart Geld
- Qualitätssicherung untergliedert sich
  - Konstruktive Qualitätssicherung
  - Analytische Verfahren

Konstruktive Qualitätssicherung

- Präventive Maßnahmen zur Vermeidung von Qualitätsmängeln
- Methoden
  - Vorgehensmodelle / Richtlinien
  - Quelltextgestaltung
  - Laufende Prozessüberwachung
  - Identifikation des Mangels und der Mangelursache
- Fehler nicht entstehen lassen ist günstiger als Fehler zu beheben

Quelltextgestaltung

- Verständlichkeit des Quelltextes
- Kommentierung des Quelltextes
  - Inline Kommentare
- Einheitliche Formatierungsrichtlinien
  - Zeilenumbrüche
- Beachtung von Entwurfs- und Implementierungsmustern
  - Refactoring
  - Bug Pattern (!)
  - Anti Pattern

Refactoring

- Semantikerhaltende Modifikation des Quelltextes
- Klarerer Quelltext / bessere Lesbarkeit
- Sehr viele Standardpattern
  - extract method
  - push down method
  - move method

Bug Pattern

- Beschreiben häufig auftretende Zusammenhänge zwischen Fehlverhalten und Ursache
- Vereinfachung des Debugging
- Einfach umzusetzende Richtlinien
- Standardpattern
  - Read after close
  - Rogue Tile
  - Split Cleaner
  - Dangling Composite
  - Null Flag

Rogue Tile

- Situation: Nach einer Fehlerkorrektur tritt der gleiche Fehler wieder auf (als hätte keine Korrektur statt gefunden)
- Ursache: Der fehlerhafte Code befindet sich durch copy-and-paste auch an anderen Stellen im Programm. Es wurden nicht alle Stellen korrigiert
- Lösung: Verwende Refactoring um die Kopien zu eliminieren. Vermeide copy-and-paste in der Zukunft

Split Cleaner

- Situation: Der Zugriff auf eine Ressource (Datenbank/Netzwerk/Dateizugriff) führt zu einer Ausnahme
- Ursache: Es könnte sein, dass die Ressourcenverwaltung über das Gesamtprogramm verteilt ist und so die zeitliche Abfolge bei Zugriffen auf die Ressource nicht garantiert werden können
- Lösung: Definiere eine Klasse, die für die Verwaltung der Ressource zuständig ist. Versuche durch geschickte Programmierung sicherzustellen, dass Reihenfolgen eingehalten werden. Definiere hilfreiche Ausnahmen

Dangling Composite

- Situation: Quelltext, der mit rekursiven Datenstrukturen arbeitet erzeugt eine NullPointerException
- Ursache: Möglicherweise wurde ein Terminierungsfall der rekursiven Datenstruktur nicht ausreichend berücksichtigt
- Lösung: Erzeuge für jede rekursive Datenstruktur eine neue Basisklasse empty. Bei der Definition rekursiver Datenstrukturen stütze stets die Basisfälle auf die Basisklasse ab

Null Flag

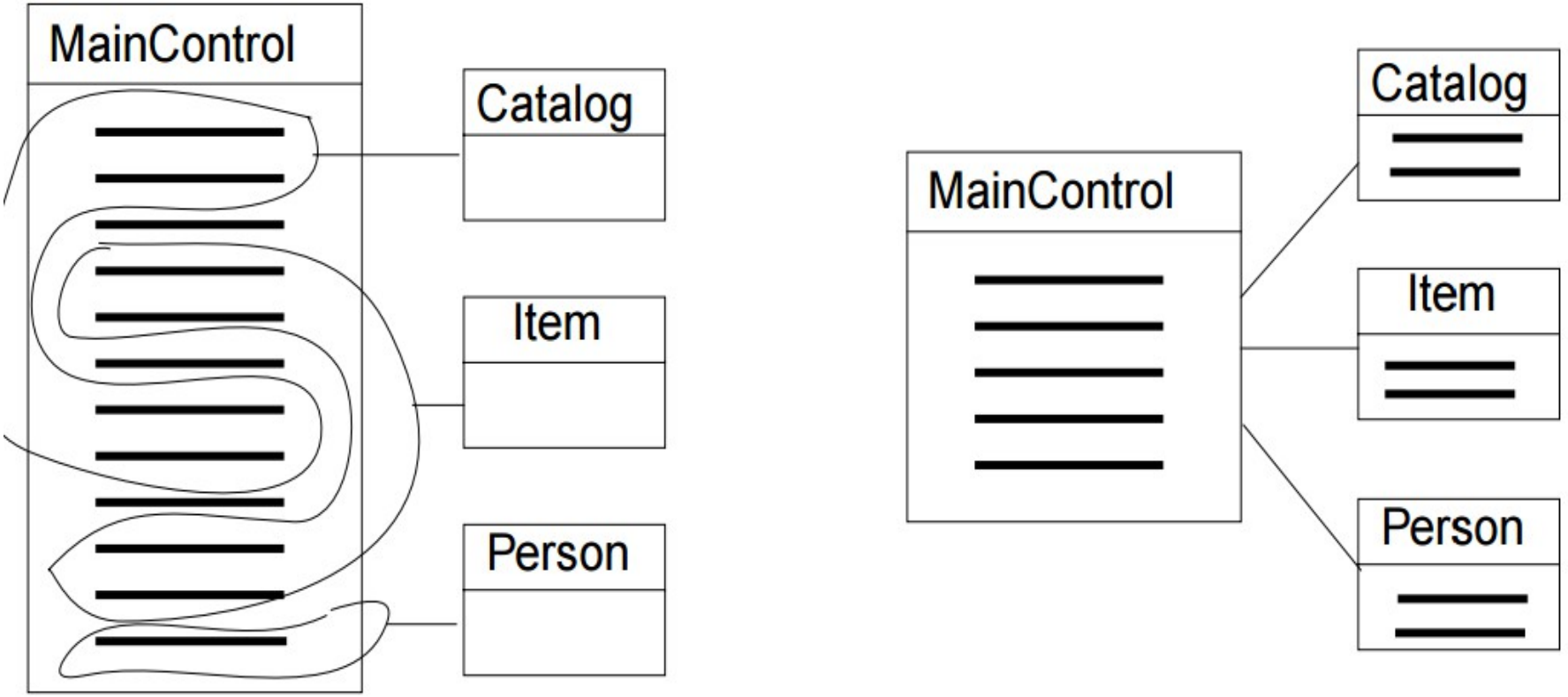
- Situation: Bei Bearbeitung des Rückgabewerts eines Methodenaufrufs wird bei Zugriff auf den Rückgabewert eine NullPointerException ausgelöst
- Ursache: Die aufgerufene Methode signalisiert möglicherweise eine Ausnahme durch Rückgabe des null-Wertes. Dieser Fall wird nicht ordentlich abgefangen
- Lösung: Signalisiere Ausnahmefälle stets durch Auslösen einer geeigneten Ausnahme. Bei Verzicht auf Laufzeitausnahmen wird auf Compiler Ebene sichergestellt, dass der Aufrufer die Ausnahme berücksichtigt

Anti Pattern

- Beschreiben typische Fehler, die bei Design oder Implementierung gemacht werden
- Anti Pattern helfen, schlechtes Design / schlechten Code zu erkennen
- Copy- Paste
- Spaghetti/Lasagna Code
- Reinvent the Wheel
- Magische Werte
- ...

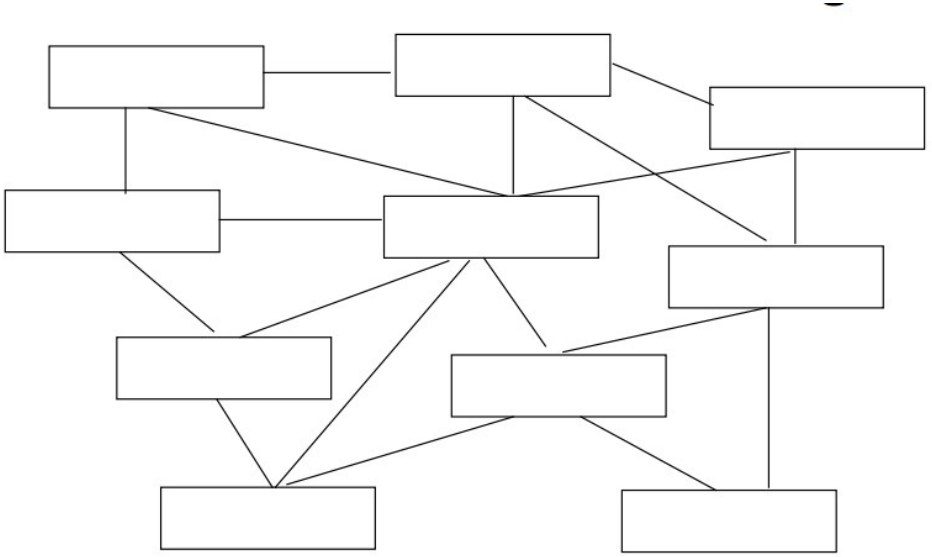
The Blob

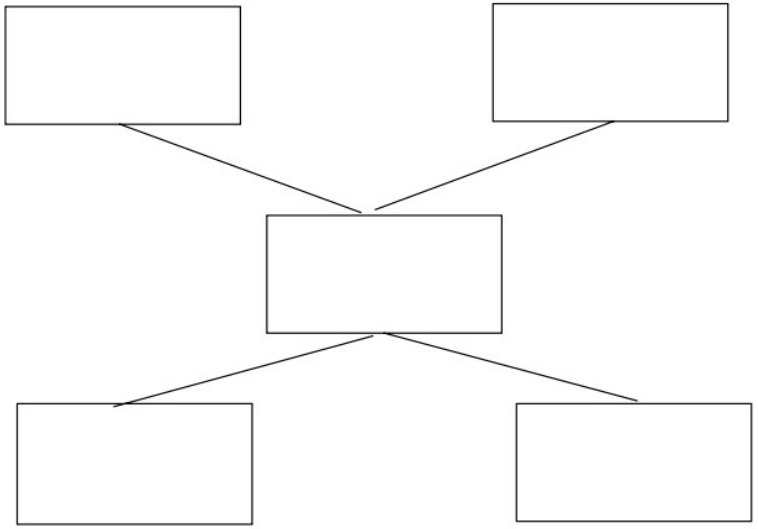
- Eine einzige Klasse dominiert den Ablauf. Darum herum sind Datenklassen angeordnet. Ablauf-orientierter Entwurf: Daten von Operationen getrennt.
- OO-Kenntnisse fehlen
- Schlecht wartbar, änderbar; nicht wiederverwendbar



Poltergeister

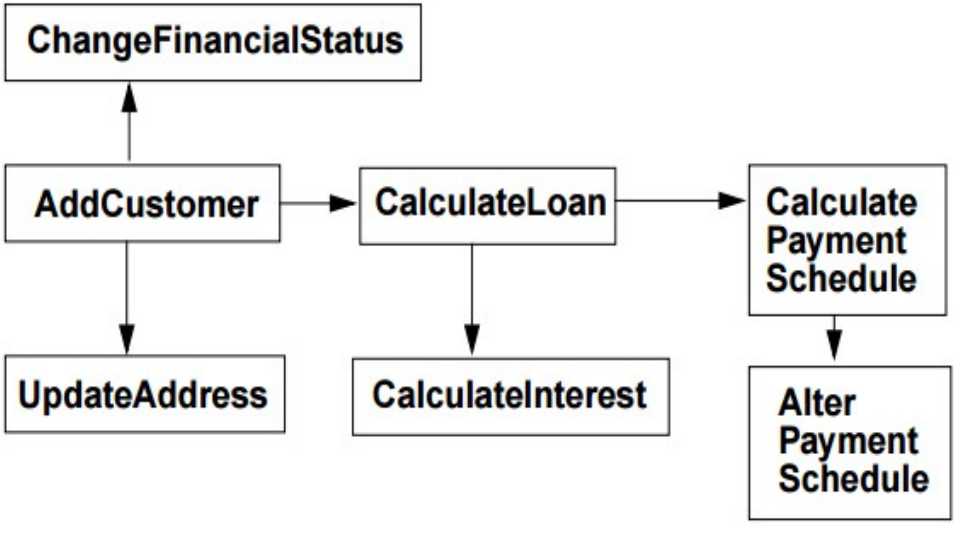
- Übermäßig viele Klassen und Beziehungen dazwischen, unnötige Klassen mit winzigen Aufgaben, kurzlebige Objekte ohne Zustand; schwache Abstraktionen, unnötig komplexe Struktur
- übertriebener Einsatz von Klassen; Anfängerfehler
- Unnötige Klassen und Komplexität stören das Verständnis, verschlechtern die Wartbarkeit, Änderbarkeit



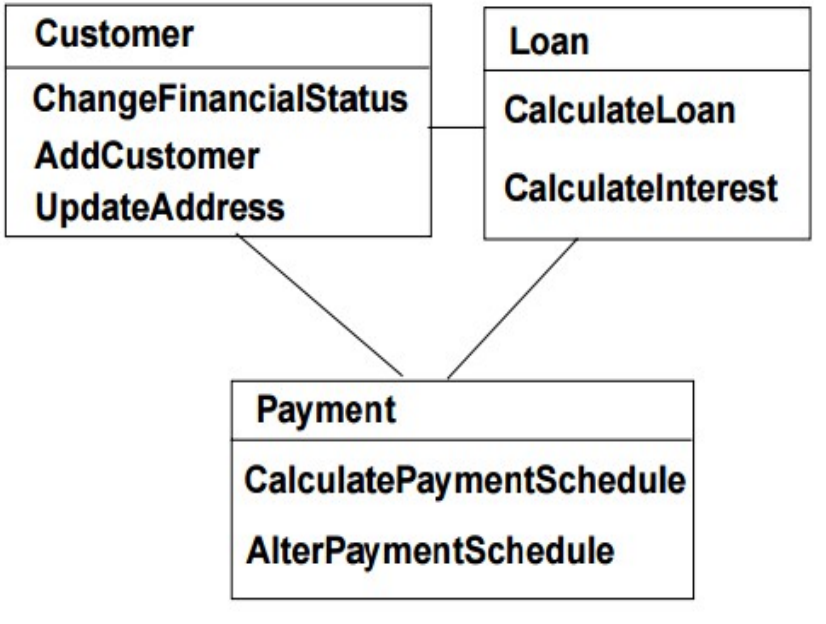


Funktionale Zerlegung

- Funktionen schrittweise verfeinert; eine Klasse für jede Funktion; OO-Techniken nicht angewandt
- OO-Kenntnisse fehlen, Programmierstil der 1970er Jahre
- schwer zu verstehen und zu warten; keine Wiederverwendung.



funktional



objektorientiert

Analytische Qualitätssicherung

- Reviews
- Statische Quelltextanalyse
- Programmverifikation
- Testen

Analytische Qualitätssicherung

- Reviews
  - Technisches Review
  - Informelles Review
  - Formale Inspektionstechnik
  - Walkthrough

Analytische Qualitätssicherung

Technisches Reviews

- Fachliche Prüfung von Artefakten
- Ziele
  - Diskussion
  - Fehler auffinden
  - Probleme lösen

Analytische Qualitätssicherung

Informelles Review

- Inhaltlich ähnlich dem technischen Review
- Meist Verzicht auf detaillierte Dokumentation/Protokollierung
- Häufig wechselseitiges Gegenlesen
- Möglichkeit zur Stellungnahme häufig gegeben
- Zeitersparnis gegenüber technischem Review

Analytische Qualitätssicherung

Inspektion

- Formale Inspektionstechnik
- Durch IEEE genormt
- Genau definierte Rollen der Teilnehmer
- Anwesenheit eines Moderators
- Vorbereitung des Reviews erforderlich

Analytische Qualitätssicherung

- Statische Quelltextanalyse
  - Finden typischer Fehler (Bug Pattern)
  - Finden von Deadlocks, null pointers, overflows
  - Problematisch: false positives vs. übersehene Fehler
- Programmverifikation
  - Garantie der Programmkorrektheit
  - Entdeckung aller Fehler relativ zur Spezifikation
- Testen
  - Prüfen auf die Anwesenheit von Fehlern