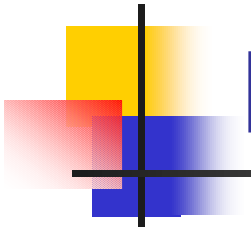
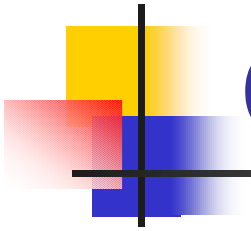


# The Blob
# Functional Decomposition

Von Thomas Heidinger

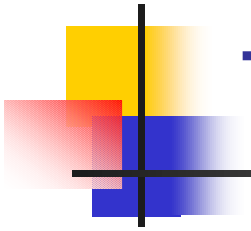
18.01.2002



# Gliederung

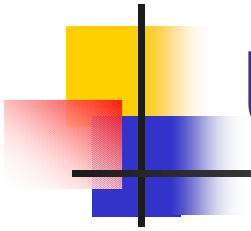
- The Blob
- Symptome
- Ursachen
- Bekannte Ausnahmen
- Lösung
- Beispiel

- Functional Decomposition



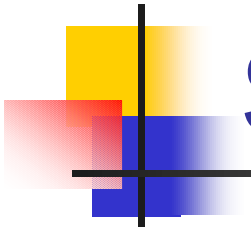
# The Blob

- Auch bekannt als: Winnebago, The God Class
- Grundübel: Eile, Trägheit
- Anekdotischer Beweis:“Diese Klasse ist wirklich das Herz unserer Architektur.“



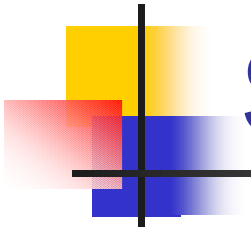
# Übliche Form

- Eine einzige Klasse (The Blob) dominiert den Ablauf
- Die anderen Klassen sind Datenklassen
- Daten und Operationen getrennt
- The Blob ist ein prozedurales Design, kein objektorientiertes!



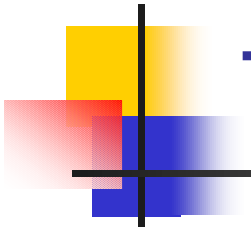
# Symptome und Konsequenzen

- Eine einzige Klasse mit sehr vielen Attributen und Operationen, die in keinem Zusammenhang stehen
- Eine einzige Controller Klasse, umgeben von einfachen Datenklassen
- Kein objektorientiertes Design
- The Blob verletzt die Vorteile der OO



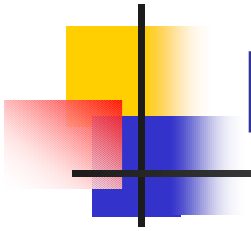
# Symptome und Konsequenzen

- Die Blob Klasse ist zu komplex, nicht wiederverwendbar, schwer zu testen
- Es ist aufwendig, die Blob Klasse in den Speicher zu laden, daher schlechte Performance



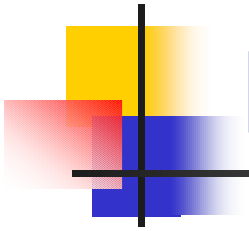
# Typische Ursachen

- Fehlen einer objektorientierten Struktur
- Fehlen irgendeiner Architektur
- Iterative Softwareentwicklung
- Spezifizierte Katastrophe



# Bekannte Ausnahmen

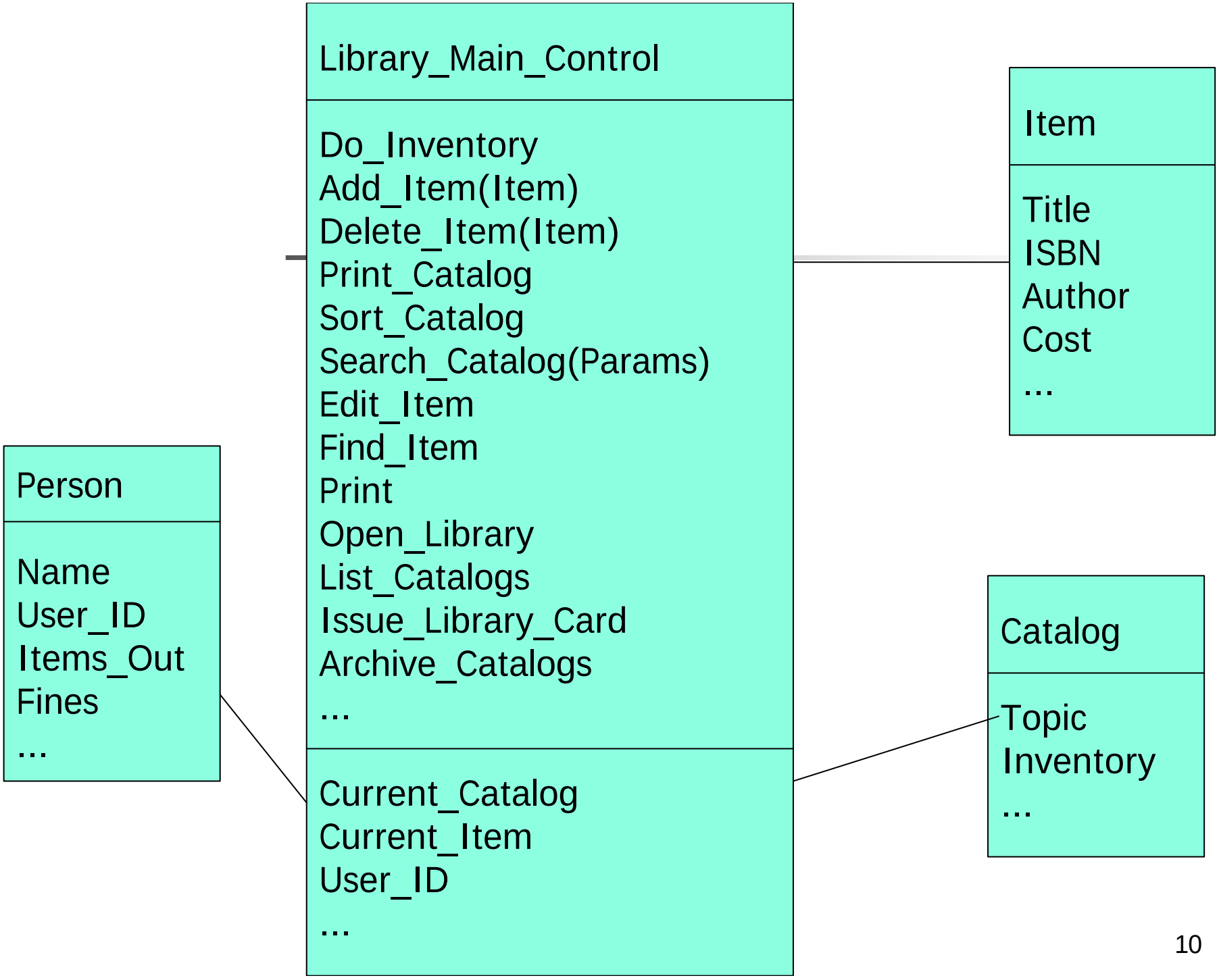
- Alte Systeme einhüllen,
  um sie besser nutzen zu können;
  keine Software Aufteilung erfordert,

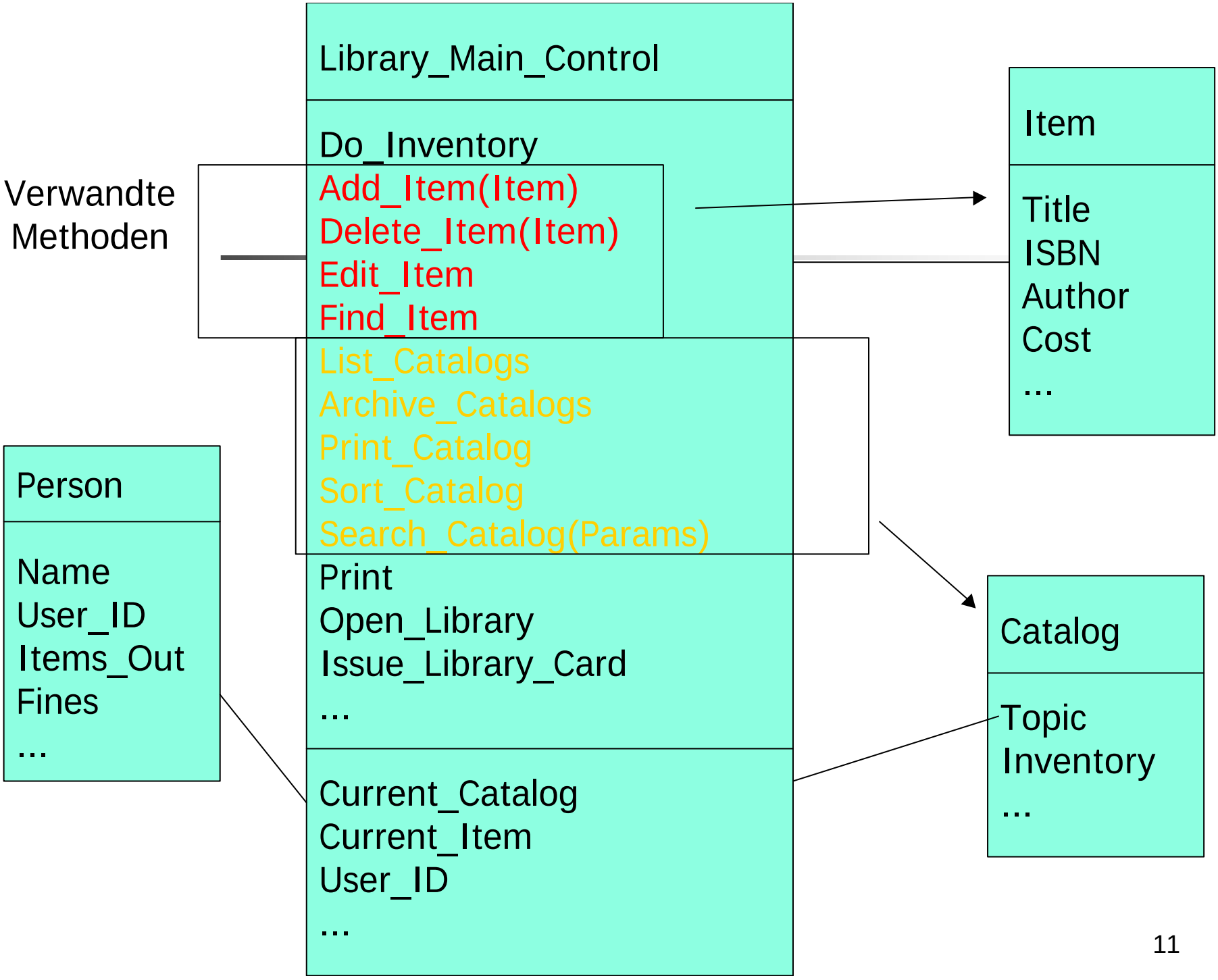


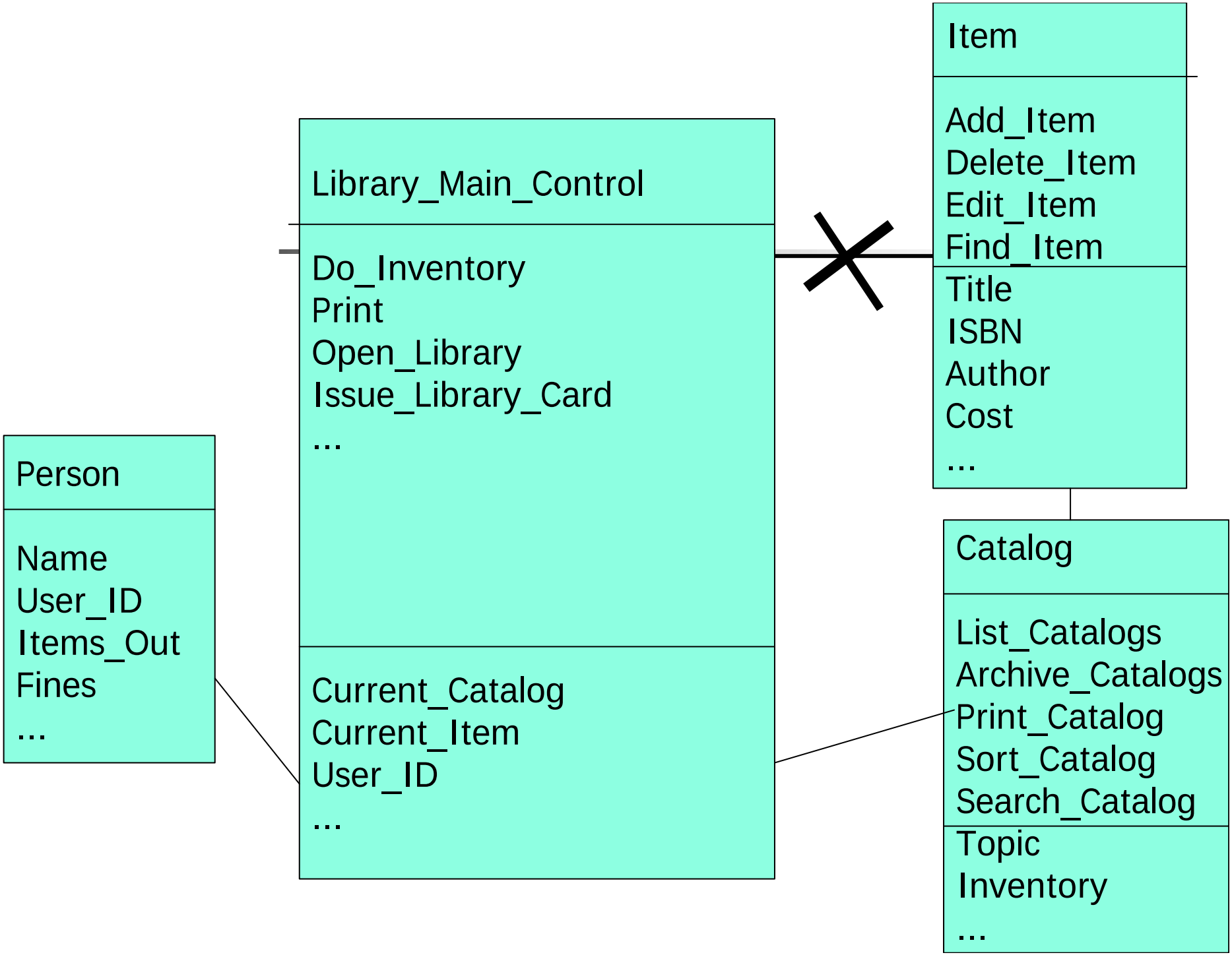
# Lösung

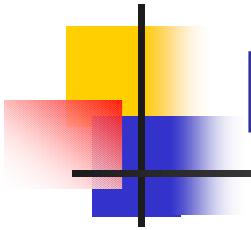
- Verhalten vom Blob zu den Datenobjekten verschieben:

1. Verwandte Attribute und Operationen identifizieren und gruppieren
2. Für diese Gruppen “natürliche Heime“ finden und sie dorthin verschieben
3. Alle “entfernt verbundene“, redundante, indirekten Assoziationen löschen
4. Eventuell neue Assoziationen einfügen


Library_Main_Control
Do_Inventory
Add_Item(Item)
Delete_Item(Item)
Print_Catalog
Sort_Catalog
Search_Catalog(Params)
Edit_Item
Find_Item
Print
Open_Library
List_Catalogs
Issue_Library_Card
Archive_Catalogs
...
Current_Catalog
Current_Item
User_ID
...
Item
Title
ISBN
Author
Cost
...
Person
Name
User_ID
Items_Out
Fines
...
Catalog
Topic
Inventory
...

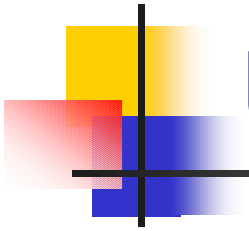

Library_Main_Control
Do_Inventory
Add_Item(Item)
Delete_Item(Item)
Edit_Item
Find_Item
List_Catalogs
Archive_Catalogs
Print_Catalog
Sort_Catalog
Search_Catalog(Params)
Print
Open_Library
Issue_Library_Card
...
Current_Catalog
Current_Item
User_ID
...
Verwandte Methoden
Item
Title
ISBN
Author
Cost
...
Person
Name
User_ID
Items_Out
Fines
...
Catalog
Topic
Inventory
...


Item
Add_Item
Delete_Item
Edit_Item
Find_Item
Title
ISBN
Author
Cost
...
Library_Main_Control
Do_Inventory
Print
Open_Library
Issue_Library_Card
...
Current_Catalog
Current_Item
User_ID
...
Person
Name
User_ID
Items_Out
Fines
...
Catalog
List_Catalogs
Archive_Catalogs
Print_Catalog
Sort_Catalog
Search_Catalog
Topic
Inventory
...



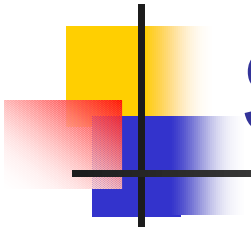
# Functional Decomposition

- Auch bekannt als: No Object-Oriented AntiPattern „No OO“
- Grundübel: Gier, Trägheit
- Anekdotischer Beweis:“Dies ist unsere ‘‘main‘‘ Routine, hier in der Klasse Listener.“



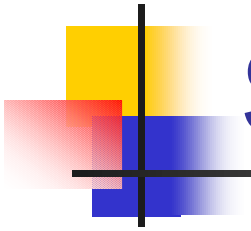
# Übliche Form

- Func. Dec. ist das Ergebnis, wenn nicht objektorientierte Entwickler in oo-Sprache programmieren.
  Sie sind “main“-Routine gewöhnt, die viele Subroutinen aufruft.
  In oo-Sprache machen sie jede Subroutine zu einer Klasse.
- Code ähnelt einer prozeduralen Sprache



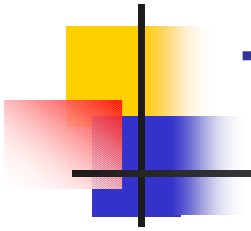
# Symptome und Konsequenzen

- Klassen haben Funktionsnamen
- Alle Klassenvariablen sind private
- Klassen mit nur einer Aktion, wie eine Funktion
- Degenerierte Architektur, nichts mit oo-Architektur gemeinsam



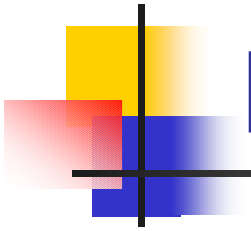
# Symptome und Konsequenzen

- Keine Verwendung von oo-Prinzipien wie Vererbung oder Polymorphismus
- Unmöglich zu erklären, wie das System funktioniert. Klassendiagramme machen keinen Sinn
- Wiederverwendung der Software nicht möglich
- Frustration der Tester



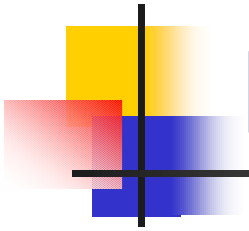
# Typische Ursachen

- Fehlen von objektorientiertem Verständnis
- Fehlende Erzwingung einer Architektur
- Spezifizierte Katastrophe



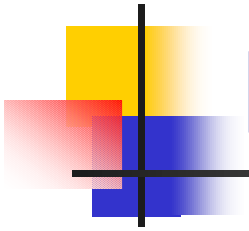
# Bekannte Ausnahmen

- Die Funktionale Zerlegung ist ok, wenn keine oo-Lösung erforderlich ist



# Lösung

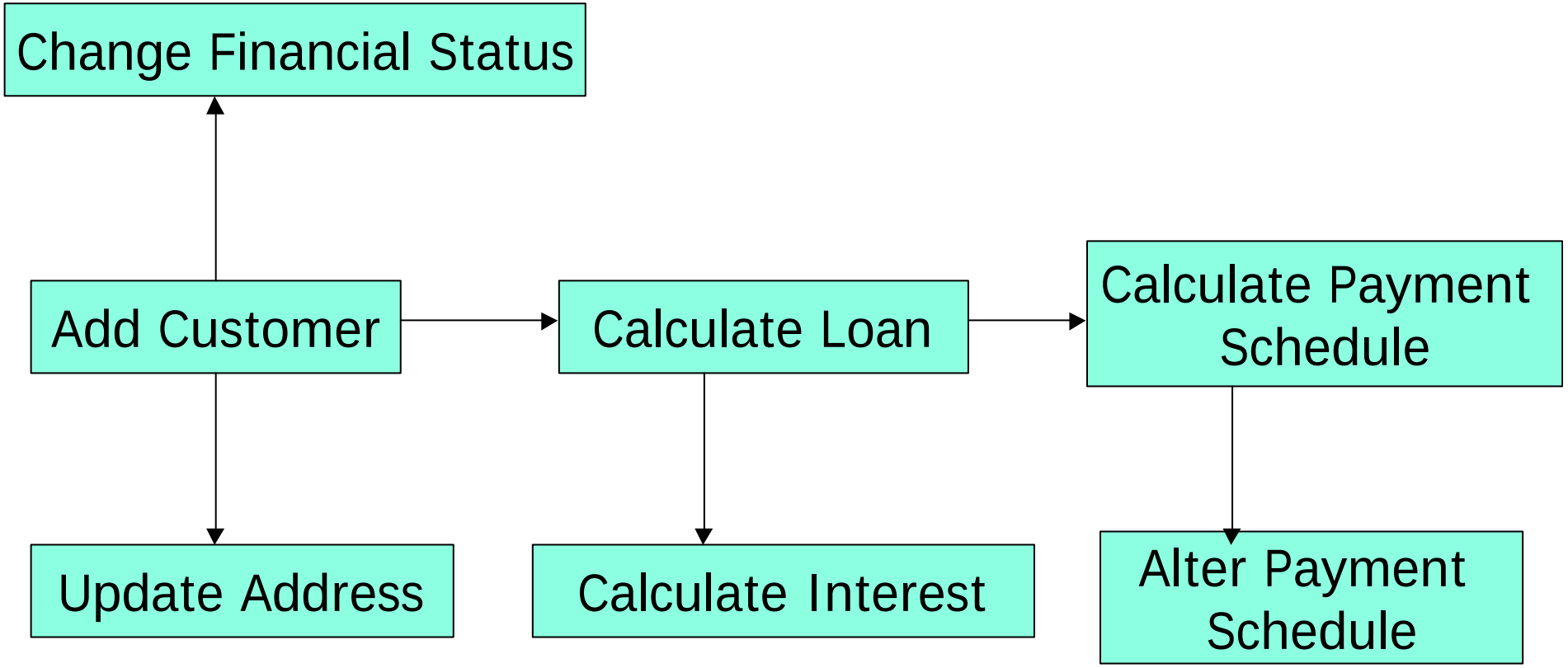
- Ermittle die Anforderungen an die Software und definiere daraus eine Zerlegung der Software
- Formuliere ein Design Modell, das die wesentlichen Teile der existierenden Software enthält
- Für Klassen, die nicht im Design Modell sind, verfahre man wie folgt:



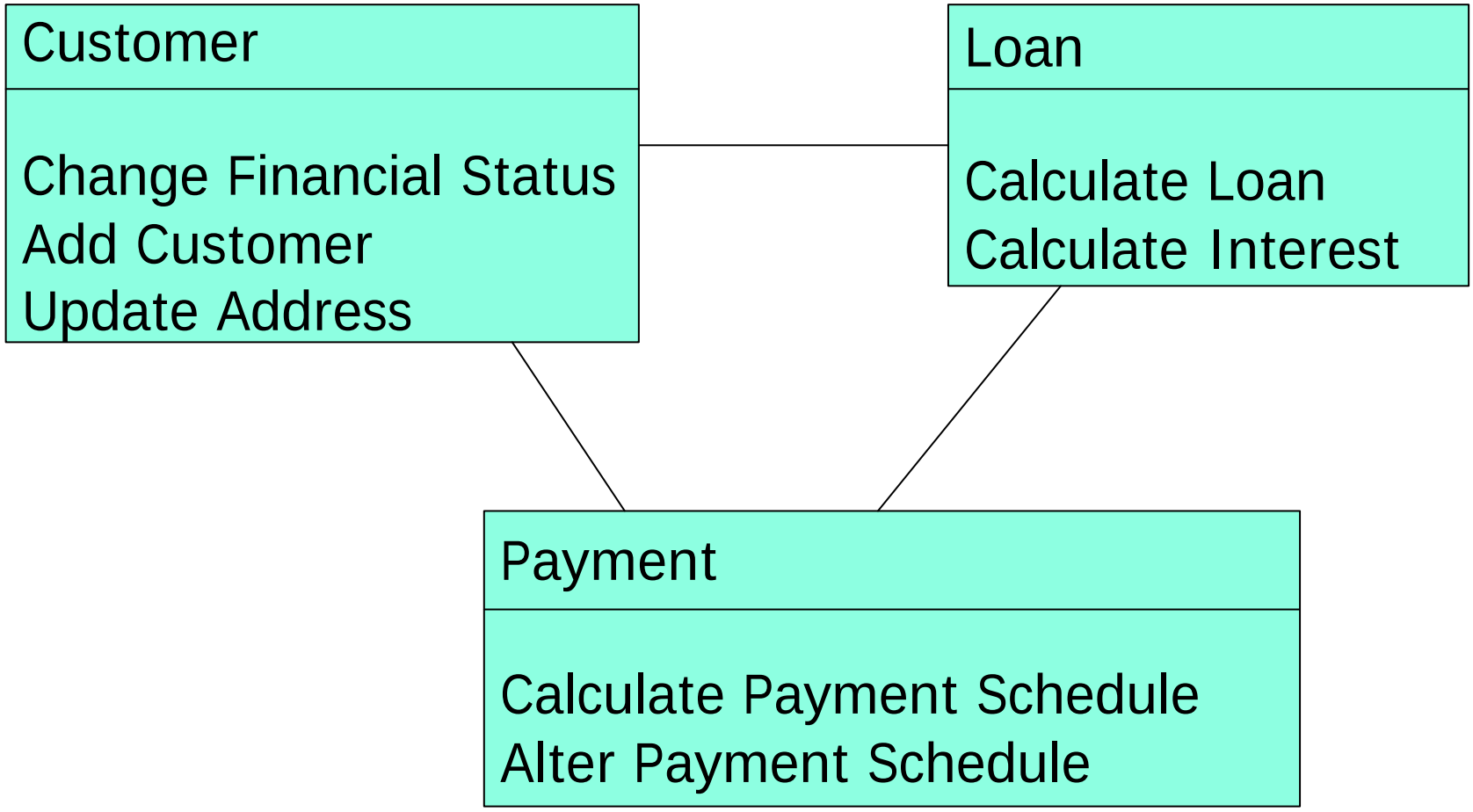
# Lösung

1. Wenn Klasse nur eine Methode hat, dann sollte Methode Teil einer anderen Klasse werden
2. Versuche verschiedene Klassen zu einer Klasse zu kombinieren
3. Enthält die Klasse keine Zustandsinformationen, sollte man sie als Funktion neu schreiben

# Beispiel


Change Financial Status
Add Customer
Calculate Loan
Calculate Payment Schedule
Update Address
Calculate Interest
Alter Payment Schedule

# Beispiel


Customer
Change Financial Status
Add Customer
Update Address
Loan
Calculate Loan
Calculate Interest
Payment
Calculate Payment Schedule
Alter Payment Schedule