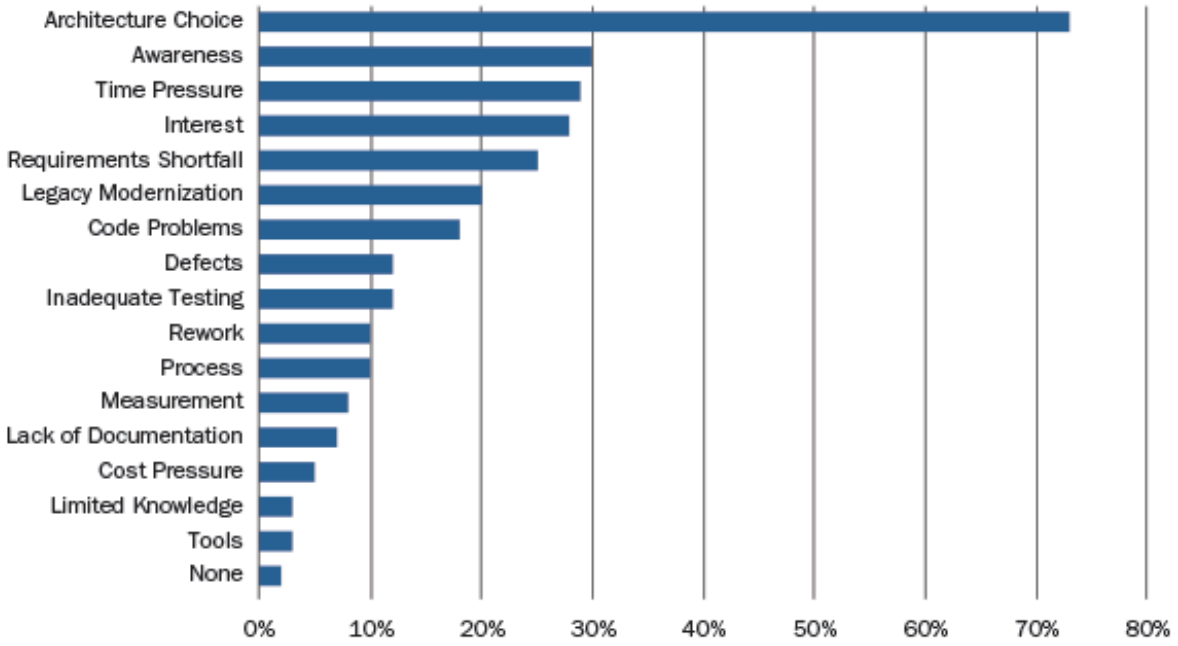
Anh Dung, Nguyen
Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe Software Engineering

Freie Universität Berlin

# Refactoring Prozess einer mobilen Applikation basierend auf Clean Architecture Prinzipien




# Auswahl der Architektur

Als Hauptgrund für technischen Schulden

# Übersicht

1. SOLID – Prinzipien und das „Clean Architecture“
2. Refactoring
3. GitJournal App Refaktorisierung
4. Fazit

# SOLID Prinzipien

- S: Das Single–Responsibility-Prinzip
- O: Das Open-Closed-Prinzip
- L: Das Liskov‘sche Substitutionsprinzip
- I: Das Interface-Segregation-Prinzip
- D: Das Dependency-Inversion-Prinzip

```
try {

  update user profile;

} catch {

  console log error;
  return error type

}
```

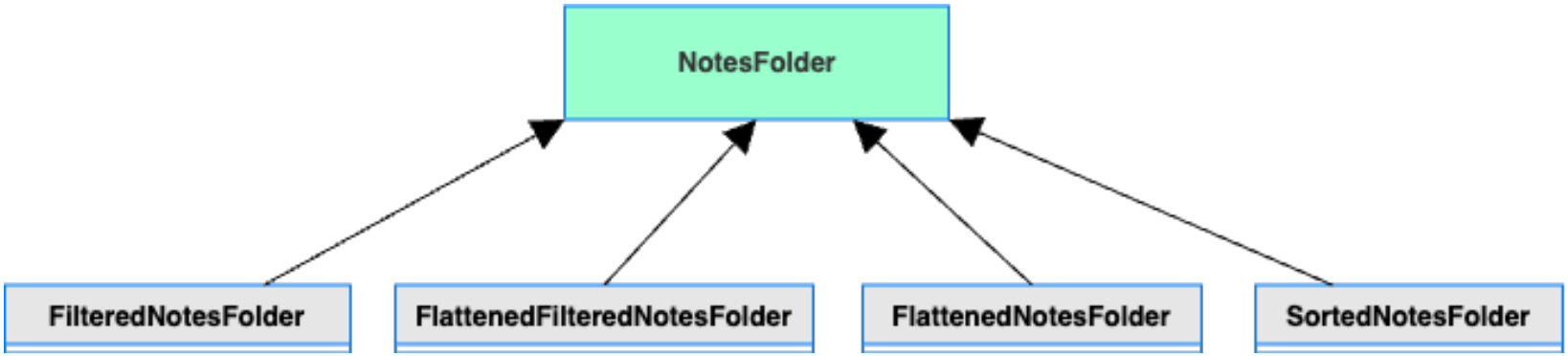
```
try {

  update user profile;

} catch {

  handle error;

}

class HandleError {
 ...
}
```

## SOLID - Das Single–Responsibility-Prinzip

- Ein Modul (Klasse/ Methode) darf nur aus **einen Grund** verändert werden

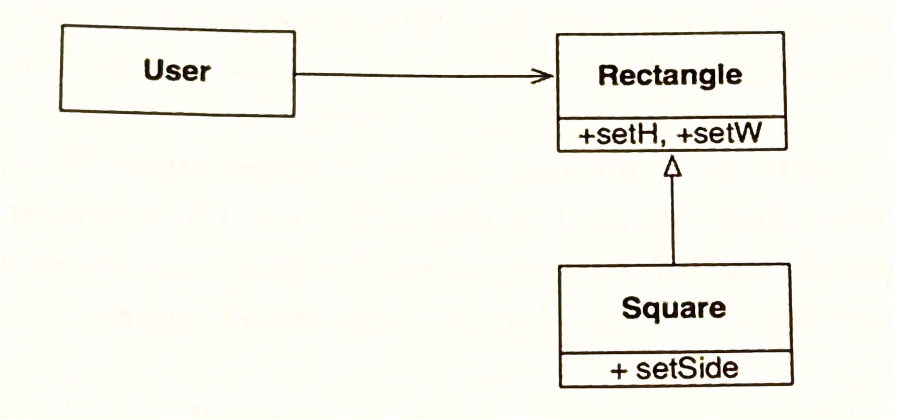
# SOLID - Das Open-Closed-Prinzip

- Das Verhalten eines Moduls soll **nicht verändert** werden und stattdessen **nur erweitert** werden.
- Trennung basierend auf die Veränderungen.



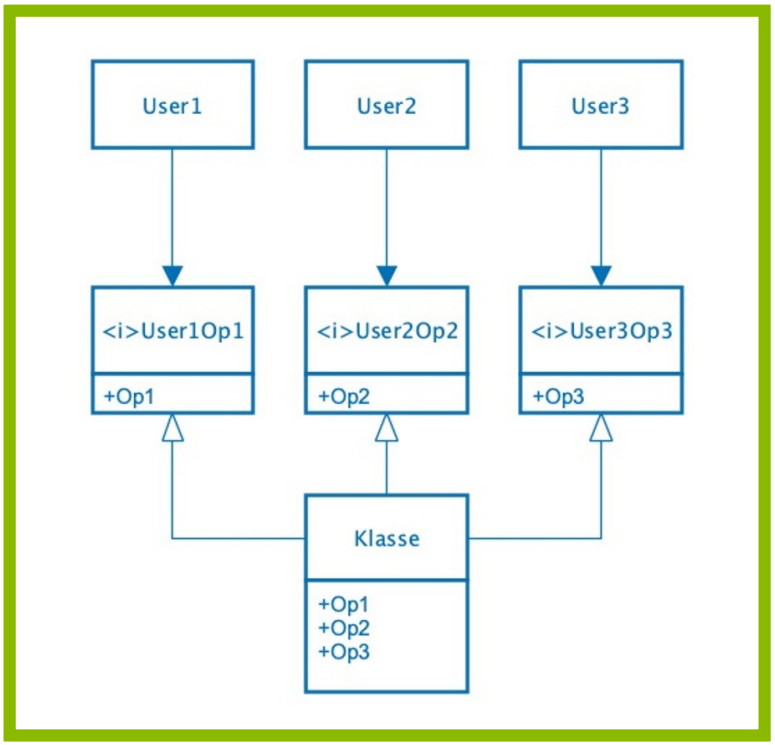
# SOLID - **Das Liskovsche Substitutionsprinzip**

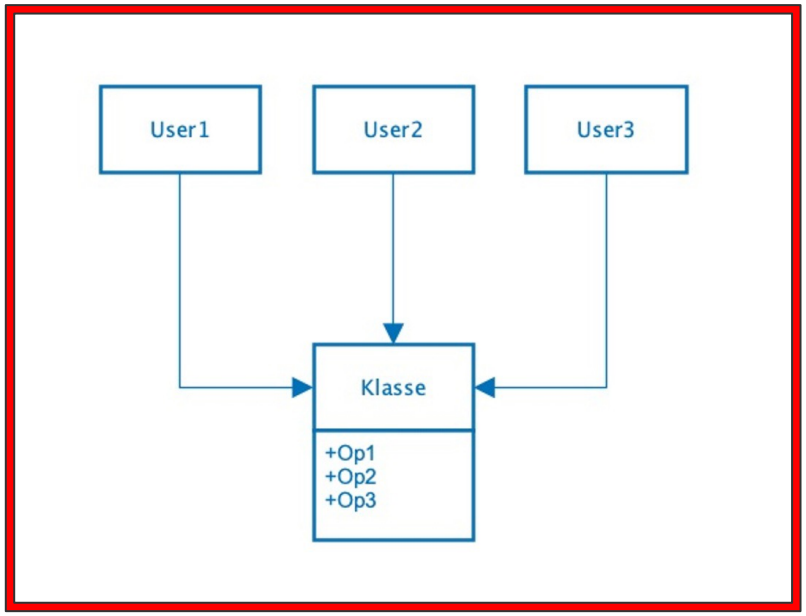
- Eine Klasse soll durch seine Unterklassen ersetzbar sein.
  - Einschränkung der Funktionalität.
- Negative Folgen bei Verletzung des Prinzips



# SOLID - Das Interface-Segregation-Prinzip

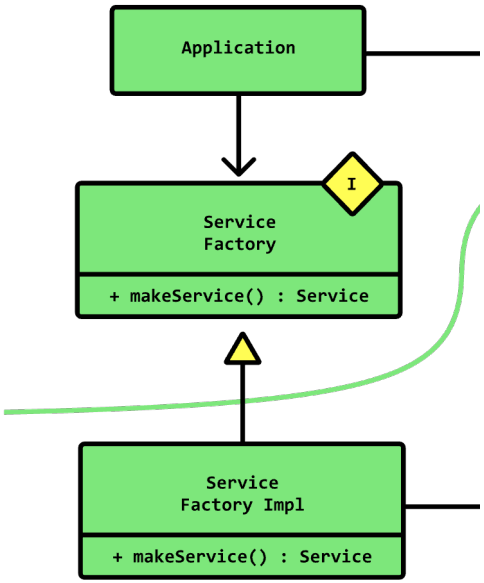
- Klassen sollen keine Methoden der Schnittstellen implementieren müssen, die diese nicht benötigt.

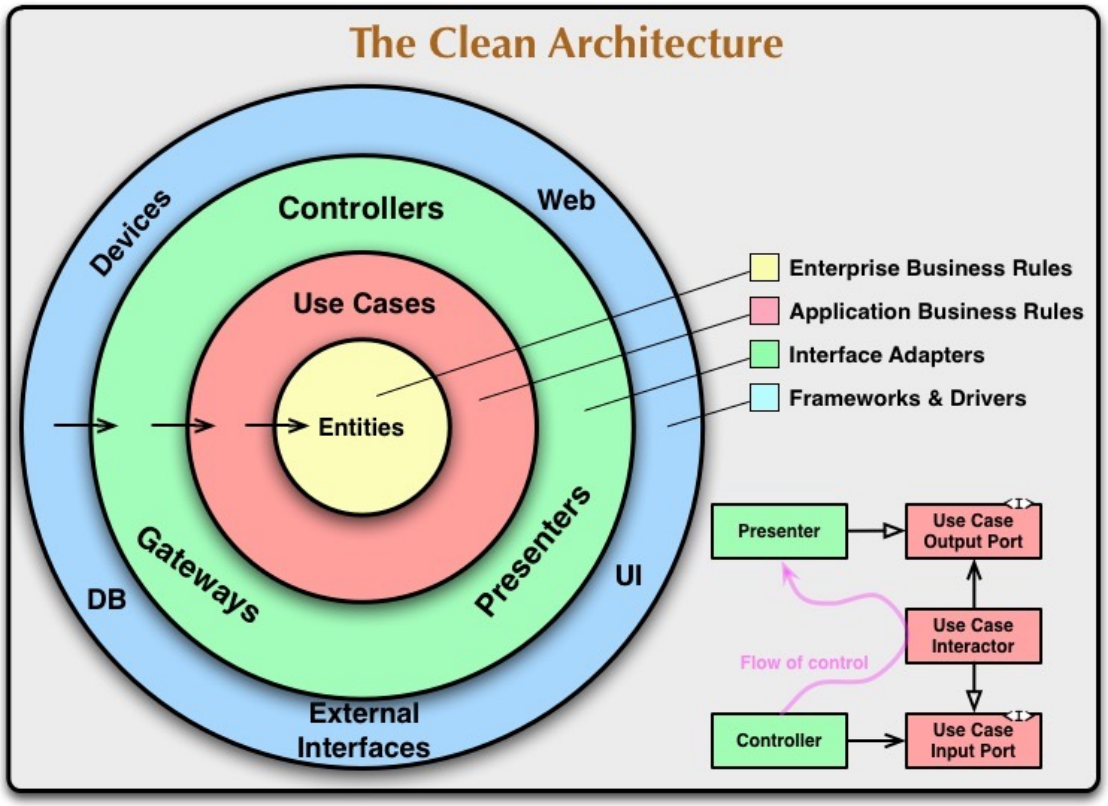


# SOLID - Das Dependency-Inversion-Prinzip

- Umkehrung der Abhängigkeit
- Entkopplung der Module mithilfe eines Interfaces
- Methoden:
  - Service Locator
  - Dependency-Injection

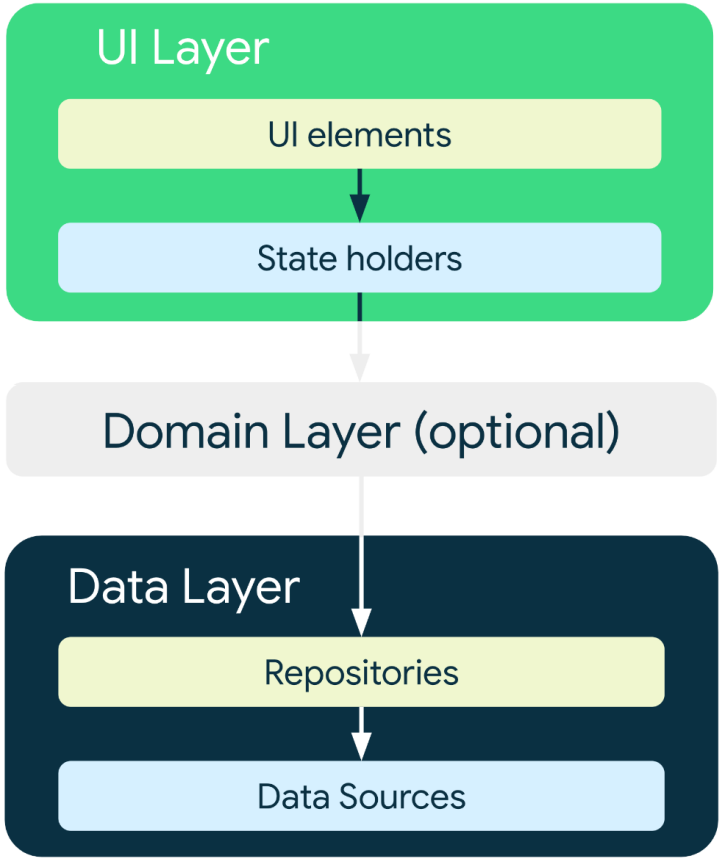





# Clean Architecture – Bob R. Martin

- Abhängigkeiten sind **nach innen** gerichtet.
- Innere Schichten sind stabiler und verändern weniger häufig als die äußeren Schichten

# Empfohlene Architektur für Android Entwicklung



# Refaktorisierung nach Martin Fowler

- Verbesserung eines Programms durch kleine Umgestaltung des Codes
- “Code Smells“ als Indikator

- Wesentlich Refaktorisierung Techniken für dieses Projekts:

1. Umbenennen
2. Methoden extrahieren
3. Methoden verschieben
4. Lokale Variablen einführen
5. Methode inline setzen
6. Parameter hinzufügen/ entfernen
7. Parameter Objekt einführen

# Refaktorisierung des GitJournal Apps

**Globale ChangeNotifier**

**GitJournal Repository**

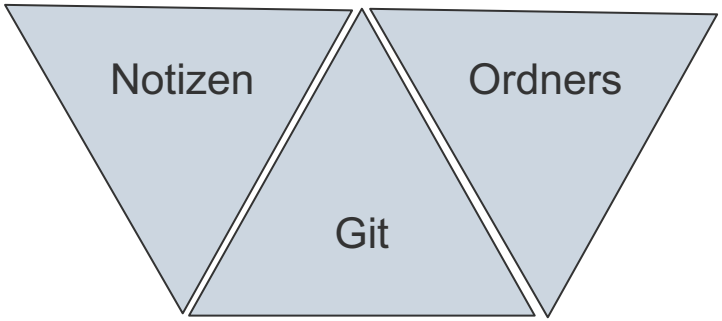
**„NotesFolder“ Klassen**

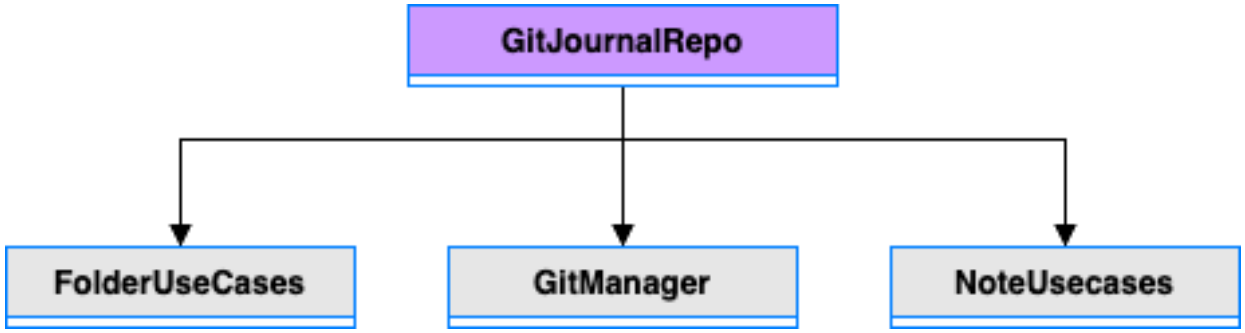
- Große Klasse
- Unterschiedliche Verantwortlichkeiten

# GitJournal Repository

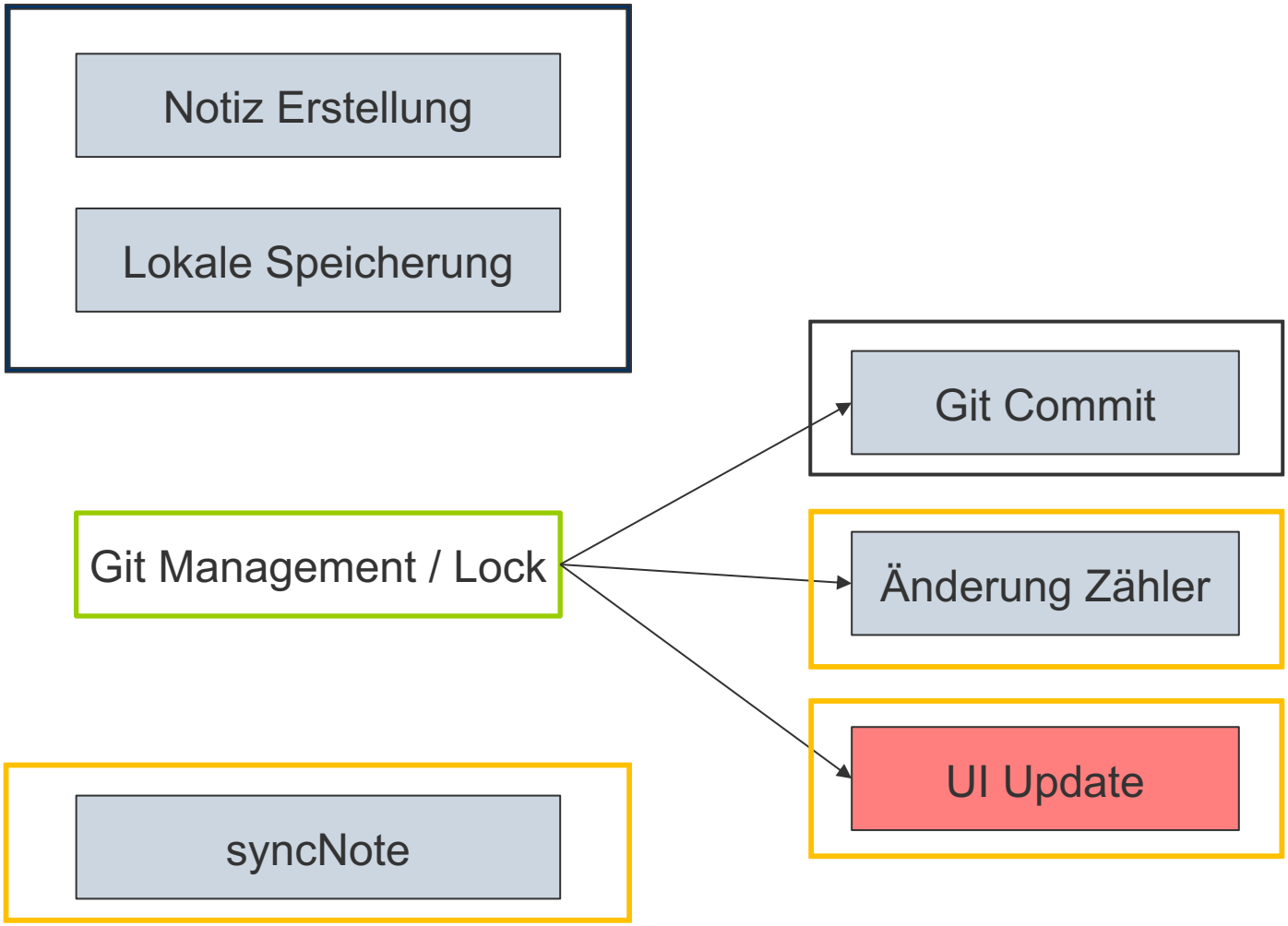
# GitJournal Repository Klasse

- Manipulieren der Notizen
- Manipulieren der Ordnern
- Interagieren mit dem Git-System

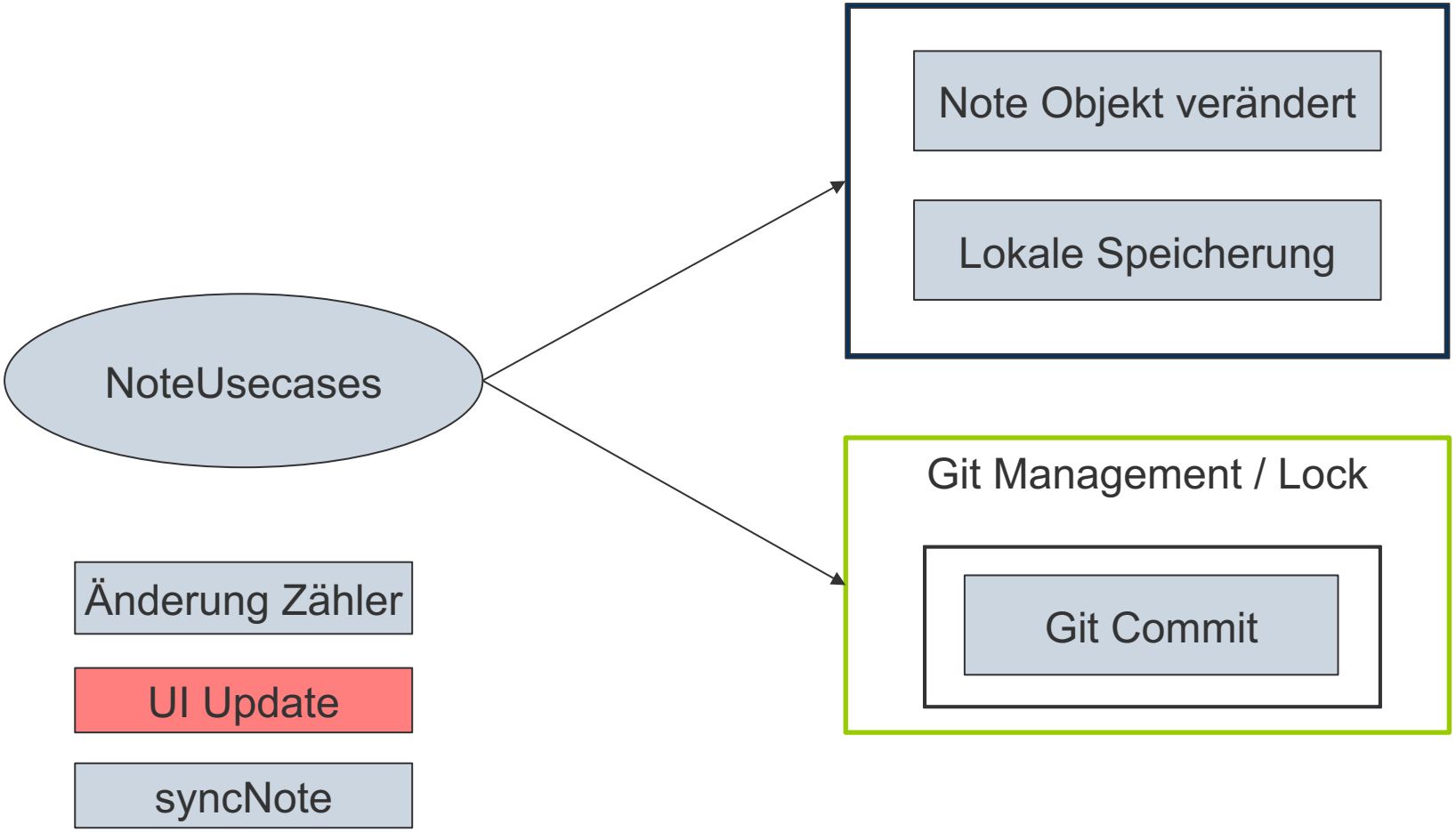


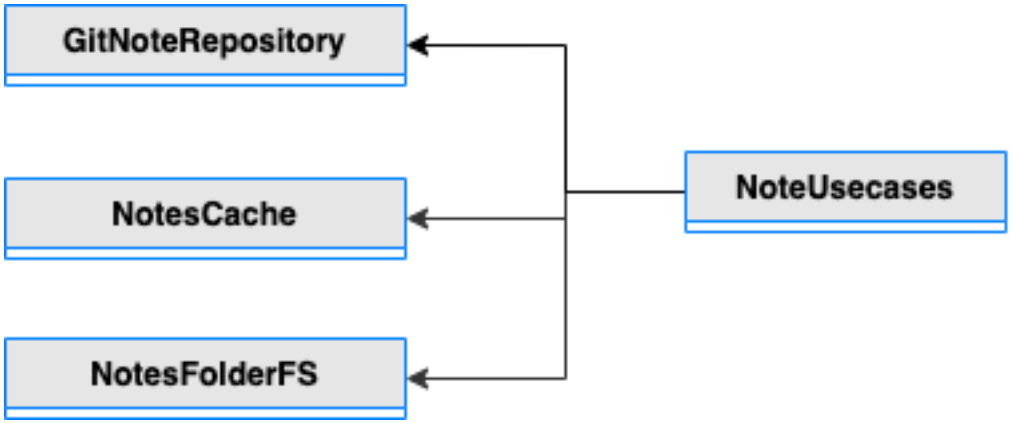
# Notizen - addNote Methode



# Notizen - addNote Methode



# Notizen - NoteUsecases

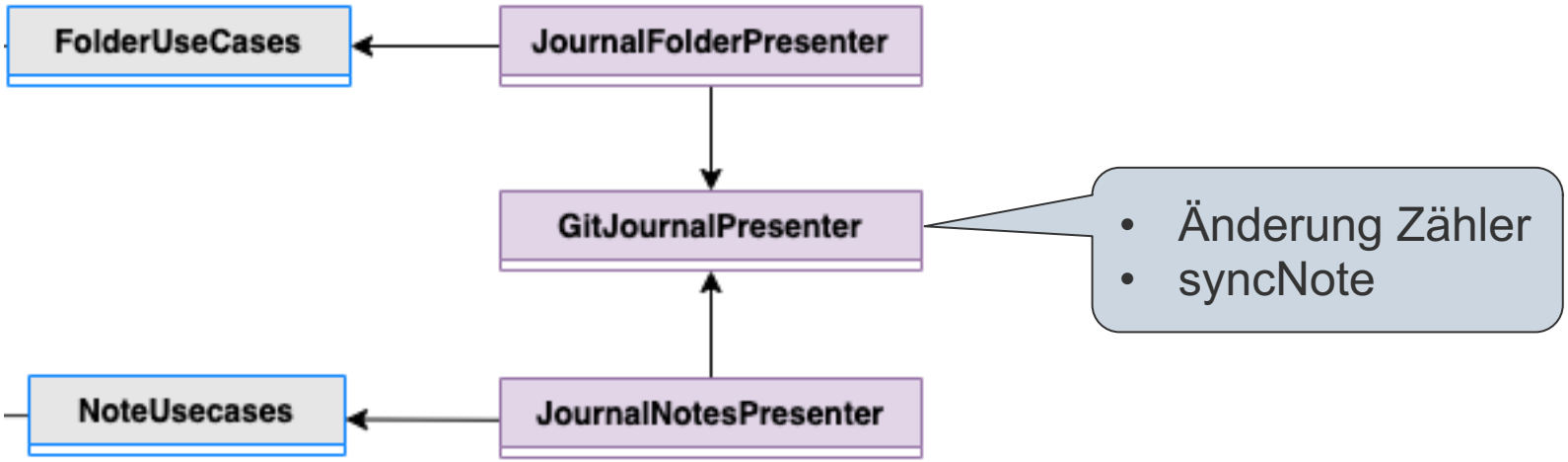


- Unsaubere Trennung
- Leistungsproblem

# GLOBALE CHANGENOTIFIER

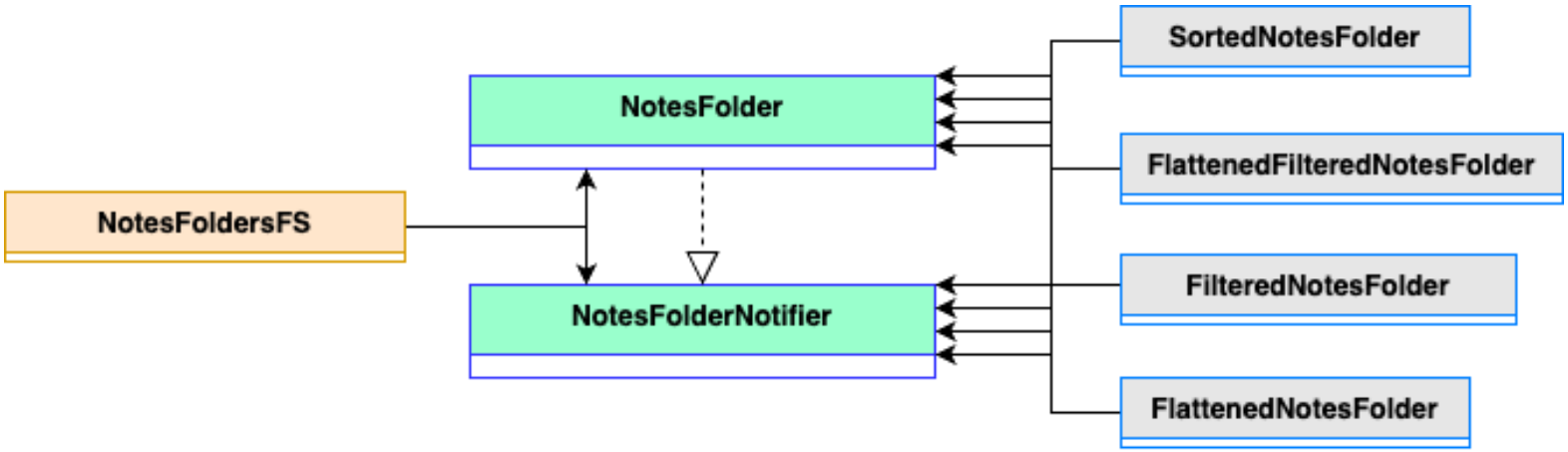
# GitJournal
# ChangeNotfier Moduls

- UI Aktualisieren
- Side Effekt durch unsaubere Trennung

- Keine klare Struktur
- Trennung nicht eindeutig definiert
- Abhängigkeiten nicht eindeutig

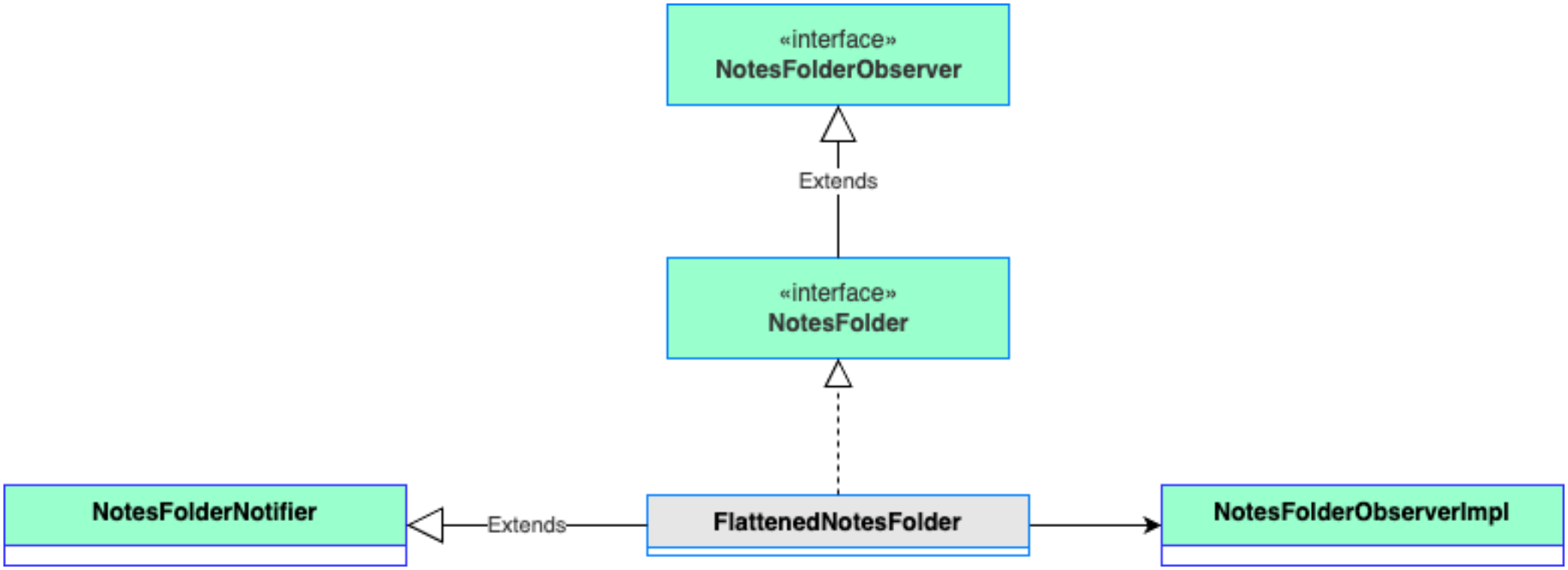
# “NOTESFOLDER“ KLASSEN




## Umstrukturierung der „NotesFolder“ Klassen

- Verletzung des Interface-Segregation-Prinzips
- FlattenedNotesFolder → NotesFolderNotifier → NotesFolder
  FlattenedNotesFolder → NotesFolder
- FlattenedNotesFolder implementiert das NotesFolder Interface mit NotesFolderNotifier

# NotesFolder - NotesFolderNotifier



# Fazit

- SOLID Prinzipien und “Code Smells“ geben gute Hinweise
- SOLID Verletzungen sind aber nicht eindeutig
- Kleine Refaktorisierung kann zu besseren Architektur führen.
- Neu schreiben könnte an manche Stelle bessere Lösung sein.