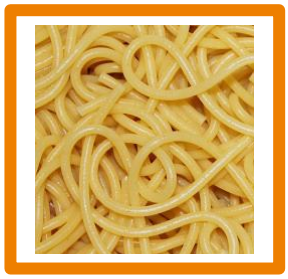
ToiletPaper #90

Pasta Orientata Agli Oggetti

Autor: M. W.

✕ Problem



Spaghetti-Code wird häufig mit "Unstrukturiertheit" in Verbindung gebracht. Objektorientierte (orientata agli oggetti) Programmierung und Paradigmen helfen dabei, Spaghetti-Code zu vermeiden. Allerdings bieten Vererbung und das Template Method Pattern auch neue Möglichkeiten, Spaghetti-Code zu entwickeln. Die Begriffe *Spaghetti-Code* und *unstrukturiert* sind allerdings zu schwammig, um diesen Aspekt zu verdeutlichen. Daher zunächst ein Versuch, Spaghetti-Code schärfer zu definieren:

Der Programmfluss von Spaghetti-Code erschwert, dass Implementierungsdetails zum einen isoliert und zum anderen verborgen werden können.

➔ Beispiel

Es sollen persönliche und generische Briefe mit einer Methode `compose()` erstellt werden. Die abstrakte Klasse delegiert dafür an Templates. Die konkreten Implementierungen nutzen `compile()`. Eine Implementierung ist in `PersonalLetter` skizziert.

```
public abstract class BaseLetter {
 public String compose(String msg) {
   return salutation() + body(msg)
     + signature();
 }
 protected abstract String salutation();
 protected abstract String body(String);
 protected abstract String signature();
 protected String compile(String) {...}
}
```

```
public class PersonalLetter
    extends BaseLetter {
  private String name;
  protected String salutation() {
    return compile("Dear _"+name+"_,");
  }
  protected String body(String) {...}
  protected String signature() {...}
}
```

Der Code ist strukturiert, `compile()` lässt sich aber schwer testen. Details sind nur eingeschränkt verborgen. Ursache ist der Programmfluss, der sich durch die Klassenhierarchie schlängelt. Ähnliche Beispiele heißen: `BaseMapper`, `BaseTest`, `Base...`

✓ Verbesserung

Tod dem `protected`! Das Template Method Pattern ist für `public` Methoden geeignet, wie z. B. `addAll()` und `add()` bei `AbstractList`. Eine Vererbung, bei der es nur um Wiederverwendung geht, lässt sich häufig durch Delegation ersetzen. Hier ist das Strategy Pattern besser geeignet. Die Klassen sind leicht zu isolieren. Details lassen sich mit `private` verbergen.
Das Interface `CompositionStrategy` definiert die Template-Methoden. `MarkdownCompiler` implementiert `compile()`.

```
public class LetterComposer {
  private CompositionStrategy strategy;
  public String compose(String msg) {
    return strategy.salutation()
      + strategy.body(msg)
      + strategy.signature();
  }
}
```

```
public class PersonalCompositionStrategy
    implements CompositionStrategy {
  private String name;
  private MarkdownCompiler compiler;
  public String salutation() {...}
  public String body(String) {...}
  public String signature() {...}
}
```

+ Weiterführende Aspekte

- Template Method Pattern
- Strategy Pattern
- SOLID Design Principles