Refactoring

Lukas Panni
TINF18B5

DHBW Karlsruhe
Vorlesung Advanced Software Engineering Semester 5/6

Inhaltsverzeichnis

1 Code Smells **3**

1.1 Duplicated Code 3

1.2 Long Method 3

1.3 Large Class 3

1.4 Shotgun Surgery 3

1.5 Switch Statements 4

1.6 Code Comments 4

2 Refactorings **5**

2.1 ExtractMethod bei Commit 0c0b357 5

1 Code Smells

Im Folgenden sollen bekannte Code Smells im Code identifiziert werden. Im Abschnitt Refactorings werden Refactorings beschrieben, die einige der Identifizierten Code Smells beheben sollen.

1.1 Duplicated Code

Duplicated Code beschreibt, dass der gleiche beziehungsweise sehr ähnlicher Code an mehreren Stellen des Systems vorkommt. Dadurch ist der Wartungsaufwand vergleichsweise hoch, da bei jeder Änderung potentiell mehrere Stellen angepasst werden müssen. Das kann auch dazu führen, dass sich die Funktionalität der einzelnen Stellen im Laufe der Zeit minimal unterscheidet, wodurch das Verhalten des Systems inkonsistent wird. Um Duplicated Code zu reduzieren muss der doppelte Code ausgelagert werden und kann dann an verschiedenen Stellen wiederverwendet werden.

- TimeSpanDetails: Click-Listener-Code wird vier mal in gleicher Form (bis auf eine Variable) verwendet. Gelöst mit ExtractMethod bei Commit 0c0b357

1.2 Long Method

Der Code Smell *Long Method* zeichnet sich durch sehr lange Methoden aus, wobei die Länge, ab der eine Methode als zu lang betrachtet wird, von Projekt zu Projekt variabel sein kann. Lange Methoden erschweren das Verständnis des Codes, was wiederum die Wartbarkeit und auch die Erweiterbarkeit einschränkt. Als Lösung kann die Lange Methode in mehrere kürzere Methoden aufgeteilt werden.

1.3 Large Class

Ähnlich wie Long Method beschreibt *Large Class*, Klassen, die vergleichsweise viele Code-Zeilen beinhalten. Dies kommt häufig vor, wenn mehrere Verantwortlichkeiten in einer Klasse untergebracht werden. Auch hier kann die Verständlichkeit des Codes erschwert werden. Das Aufteilen der Klasse in mehrere Klassen, ist eine sinnvolle Lösung für diesen Code Smell.

1.4 Shotgun Surgery

Shotgun Surgery beschreibt, dass für vergleichsweise kleine Funktionale Änderungen Anpassungen an vielen Stellen notwendig sind und deutet auf schlechte Struktur und eine

Verflechtung von Verantwortlichkeiten hin. Durch eine Umstrukturierung des Codes, sodass jede Klasse nur eine Verantwortlichkeit hat, kann dies behoben werden.

1.5 Switch Statements

Die Verwendung von *Switch Statements* fördert Fehler durch die unintuitive Syntax und verleitet oft dazu das gleiche Switch-Statement an mehreren Stellen einzusetzen. Durch das übermäßige Verwenden von Switch Statements wird die Wartbarkeit und auch die Erweiterbarkeit des Codes eingeschränkt. Switch Statements können in Objektorientiertem Code häufig durch die Verwendung von Polymorphie reduziert werden.

1.6 Code Comments

Code Comments die beschreiben, was der Code an dieser Stelle tut, deuten häufig darauf hin, dass der Code an dieser Stelle unverständlich geschrieben ist.

2 Refactorings

Die hier beschriebenen Refactorings sollen das Design des Systems verbessern, die Wartbarkeit und Erweiterbarkeit verbessern und auch die Verständlichkeit erhöhen. Dadurch soll es unter anderem einfacher werden Fehler zu finden und zu beheben sowie neue Funktionen hinzuzufügen.

2.1 ExtractMethod bei Commit 0c0b357

In der Klasse *TimeSpanDetails* wurde sehr ähnlicher Code für einen Click-Listener an vier unterschiedlichen Stellen verwendet. Durch die Auslagerung in die Methode *getClickListener* kann der Duplicated Code vermieden werden. Durch die Einführung des Parameters *resource* kann die extrahierte Methode flexibel an allen Stellen wiederverwendet werden. Dadurch wird außerdem die Lesbarkeit des Codes erhöht. Auch eventuelle spätere Änderungen am Verhalten der Methode müssen so nur an einer Stelle durchgeführt werden.

Vorher:

```
[...]
view.findViewById(R.id.to_commit_repos)
    .setOnClickListener(v -> Navigation.findNavController(view)
    .navigate(R.id.action_1, getArguments()));
view.findViewById(R.id.to_issue_repos)
    .setOnClickListener(v -> Navigation.findNavController(view)
    .navigate(R.id.action_2, getArguments()));
[...]
```

Nacher:

```
[...]
view.findViewById(R.id.to_commit_repos)
    .setOnClickListener(getClickListener(view, R.id.action_1));
view.findViewById(R.id.to_issue_repos)
    .setOnClickListener(getClickListener(view, R.id.action_2));
[...]
```