REFACTORING

Studientag 01853

Moderne Programmiertechniken und -methoden

AUSGANGSPUNKT

- Softwarefäulnis
- eine kleine Änderung kann einen großen Aufwand nach sich ziehen
- nur unvermeidliche Änderungen werden umgesetzt

REFAKTORISIERUNG

- Änderung einer Softwarestruktur mit dem Ziel, sie verständlicher und einfacher änderbar zu machen, ohne das beobachtbare Verhalten zu ändern
- Refactorings sind i.d.R. kleineren Umfangs
- Refactoring-Tools sind Metaprogramme, können also wie Programme spezifiziert werden (Vor- und Nachbedingungen, automatisiertes Testen)

REFAKTORISIERUNG II

- Refactoring to patterns: aufgrund unklarer Ausgangslage nicht unbedingt automatisiert möglich
- Refactoring zwischen patterns: einfacher umsetzbar
- Refactorings müssen getestet werden, da Vorbedingungen ggf. nicht sofort offensichtlich sind
- Refactoring-Tools brauchen i.d.R. Zugriff auf den AST

BEDINGUNGEN VEREINFACHEN

- Verschachtelte Bedingungen durch Wächter ersetzen
- Bedingung zerlegen: Auslagerung einer komplexen Bedingung in eine Methode
- Bedingung durch Polymorphismus zerlegen: sinnvoll bei Fallunterscheidungen, aber: Übersicht über alle Fälle geht verloren

BEDINGUNGEN VEREINFACHEN II

- Einführung eines Null-Objekts: statt null wird ein Objekt verwendet
  - Schutz vor NullPointerExceptions
  - sinnigerweise als Singleton implementiert
- Zusicherung einführen: explizite Angabe von Vorbedingungen

LESBARKEIT VERBESSERN

- Methode oder Variable umbenennen
- Parameterklasse einführen
- Konstruktor durch Factory-Methode ersetzen
- Fehlercode durch Ausnahme ersetzen
  - Fehlercodes müssen bei „tiefen“ Aufrufen durch die gesamte Hierarchie gereicht werden

LESBARKEIT VERBESSERN II

- Ausnahme durch Vorbedingung ersetzen
  - falls Exception als Ausnahmesituation nicht gerechtfertigt ist, kann der reguläre Fall durch eine explizite (und als aufrufbare Methode implementierte) Vorbedingung notiert werden
- Ausnahme durch Test ersetzen
  - wie oben, nur ohne assert

DATEN ORGANISIEREN

- Feld kapseln: Zugriffsfunktionen statt public-Attribut
- Collections kapseln: kein direkter Zugriff von außen auf private Collections

DATEN ORGANISIEREN II

- Attributwert durch Objekt ersetzen: eigene Typen für Attribute
- Wert durch Referenz ersetzen: Wert- oder Referenzobjekt je nach Anwendungsfall
- Klasse extrahieren: Zusammenfassung der beiden vorhergehenden Refactorings

DATEN ORGANISIEREN III

- Unidirektionale in bidirektionale Verknüpfung ändern
  - aufwändigere Aktualisierung der Verknüpfungen

GENERALISIERUNG EINSETZEN

- Superklasse extrahieren: zu zwei ähnlichen, prinzipiell substituierbaren Klassen
- Feld oder Methode nach oben verschieben
  - setzt syntaktische und semantische Gleichheit voraus
  - weitere Subklassen können so einen neues Feld / eine neue Methode unbeabsichtigt erben
  - der Typ von this ändert sich in der Methode

GENERALISIERUNG EINSETZEN II

- Feld oder Methode nach unten verschieben
- Subklasse extrahieren: wenn Superklasse Felder/Methoden enthält, die nicht für alle Instanzen gelten
- Interface extrahieren: meist werden kontextspezifische Interfaces erstellt
  - benötigte Methoden ggf. nicht sofort offensichtlich, wenn z.B. das entsprechende Objekt als Parameter weitergereicht wird

GENERALISIERUNG EINSETZEN III

- Interface berechnen: Erstellung von Typeconstraints anhand Nachverfolgung
  - das gesuchte Interface ist das kleinste, das alle Typeconstraints erfüllt
- Subklasse durch Felder ersetzen: Gegenteil zu Attributwert durch Objekt ersetzen
  - bei begrenzter Anzahl von Subklassen, die alle ähnliches Verhalten zeigen

GENERALISIERUNG EINSETZEN IV

- Vererbung durch Delegation ersetzen
- genaugenommen Forwarding
- ggf. muss ehemalige Subklasse Interfaces implementieren oder eine bestimmte Superklasse haben

➡ Erfüllung der Zuweisungskompatibilität

METHODEN ORGANISIEREN

- Methode extrahieren: hin zu kurzen, prägnanten Methodennamen
  - Finden der benötigten formalen Parameter kann aufwändig sein
- Methode in Methodenobjekt auslagern
  - Methodenobjekt leitet sich dann aber nicht mehr aus der Problemdomäne ab

METHODEN ORGANISIEREN II

- Feld oder Methode verlagern: in eine andere Klasse, wenn deren Zustand gehäuft abgefragt und anschließend geändert wird

- Delegat verbergen: neue Methode, die den Zugriff auf einen Delegat kapselt (s. Law of Demeter)

- Mittelsmann entfernen: Umkehrung des vorhergehenden

METHODEN ORGANISIEREN III

- Klassenfremde Methode einführen: kein Zugriff, um eine Methode am beabsichtigten Ort zu definieren
  - dann als statische Methode in der aktuellen Klasse
- Lokale Erweiterung einführen: kein Zugriff, um eine Methode am beabsichtigten Ort zu definieren
  - dann als Methode in einer Subklasse der problematischen Klasse (ohne Überschreiben von Methoden)

REFACTORING TOOLS

- Forderungen, um Implementierung von Refactoring-Tools zu vereinfachen
  - ein einheitlicher AST für alle IDEs und Sprachen
  - formale Sprache, mit der sich Refactorings als Verkettungen auf diesen AST ausdrücken lassen