Hochschule Kempten
University of Applied Sciences

Seminar

Refactoring in der Softwareentwicklung
Sommersemester 2024
bei Prof. Dr. Georg Hagel

five lines of code (Christian Clausen)

Thema: Fogle der Struktur im Code

Name, Vorname	Heiserer Valentin
Matrikelnummer	453990
Studiengang	Bachelor Informatik
Semester	Sommersemester 2024

Kurzzusammenfassung

Dieser Blindtext zeigt den ungefähren Umfang des deutschen Abstract.Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In dieser Arbeit solle genauer untersucht werden, was Refactoring tatsächlich an Code verändert, welche Möglichkeiten es gibt ein bestimmtes Verhalten überhaupt darzustellen. Als Grundlage dieser Arbeit gilt das 11. Kapitel aus dem Buch *„five lines of code“* von Christian Clausen [1]. Dieses wurde im Rahmen des Seminars „Refactoring“, bei Professor Dr. Georg Hagel, untersucht und für diese Arbeit aufgearbeitet. Das Kapitel des Buchs kann zwar eigenständig gelesen werden, aber ein grundlegendes Verständnis von Refactoring ist trotzdem erforderlich.
Neben dem Untersuchen des Verhaltens und der Struktur im Code, sollen einige Maßnahmen vorgestellt werden, mit denen Sicherheit erlangt werden kann, dass Code ordnungsgemäß funktioniert. Gerade nach einem umfassenden Refactoring spielt dies eine große Rolle.

2 Strukturen in der Softwareentwicklung

Bevor man sich mit den Strukturen in der Softwareentwicklung auseinandersetzen kann, ist es wichtig, sich erneut vor Augen zu führen, was Software eigentlich ist.
"*Software modelliert einen Teil der Welt. Die Welt - und unser Verständnis davon - entwickelt sich, und unsere Software muss sich entwickeln, um ein akkurates Modell zu sein.*"[1]
Das heißt also, dass Code verschiedenste Gegebenheiten aus der Realität abbildet. Darunter zählen Informationen, Zusammenhänge und ganze Abläufe. Zusammen ergibt sich dadurch eine Struktur, ein wiedererkennbares Muster, welches sich sowohl in der echten Welt als auch in der Software finden lässt.

2.1 Verschiedene Arten von Strukturen

Es gibt verschiedene Bereiche in der Softwareentwicklung, in denen Struktur eine Rolle spielt. Es ist möglich diese Bereiche an zwei Achsen einzuteilen. Zum einen gibt es einige Faktoren welche den Menschen, also die Softwareentwickler betreffen. Auf der anderen Seite befindet sich der tatsächliche Code. Um die Einteilung sinnvoll zu gestalten, teilt die andere Achsen anhand des Wirkungsbereiches der Strukturen ein. [1] Die folgende Tabelle zeigt das Ergebnis der Einteilung.

	Teamintern	Teamübergreifend
Code	Daten und Funktionen	externe APIs
Menschen	Hierarchie, Prozesse	Verhalten, Domänenexperten

Tabelle 2.1: Strukturkategorien [1] (korrigierte Form)

Melvin E. Conway stellt bereits 1968 Beobachtungen an, dass es eine gewisse Symmetrie zwischen der Arbeitsweise von Entwicklerteams und den Zusammenhängen der tatsächlichen Systemen gibt. [2] Diese Aussage wurde von Harvard Wissenschaftlern bestätigt. [3]

2.2 Weitere Einschränkungen

Auch das Nutzerverhalten kann bestimmte Strukturen vorgeben, da diese ein immer gleiches Verhalten erwarten. Auch wenn einige Abläufe optimiert oder umstrukturiert werden können, ist es nicht immer sinnvoll dies zu tun, da dann ggf. Nutzer neu geschult werden müssen.[1]

3 Arten, wie Code Verhalten spiegelt

Im nächsten Teil wird beleuchtet, wie Verhalten im Code abgebildet werden kann. Dabei gibt es grundlegend drei verschiedene Arten. Diese werden im folgenden anhand eines einfachen Beispiels erläutert. Des Weiteren soll auf die Erzeugung von Endlosschleifen eingegangen werden, wie Clausen feststellt, einen Sonderfall darstellen. [1]

Beispielverhalten

Es soll bis zu einer bestimmten ganzen Zahl abwechselnd „gerade“ und „ungerade“ in der Konsole ausgegeben werden. „0“ wird hierbei als gerade angesehen.
Dieses Verhalten ist am von Clausen genutzten Beispiel (FizzBuzz) nachempfunden und so weit wie möglich vereinfacht, um weiterhin alle nötigen Besonderheiten zu veranschaulichen.[1]

3.1 Verhalten im Kontrollfluss

Die erste und wohl einfachste Möglichkeit, Verhalten im Code abzubilden, ist der Kontrollfluss. Dieser zeichnet sich durch die Verwendung von Kontrolloperatoren, Methodenaufrufen und der Zeilenabfolge aus. [1] In folgender Abbildung werden dafür jeweils einfache Beispiele gezeigt. (3.1)

```
const i = 0;
while (i < 5) {
    foo(i);
    i++;
}
```

(a) Kontrolloperatoren

```
function loop(i: number) {
    if (i < 5) {
        foo(i);
        loop(i + 1);
    }
}
```

(b) Methodenaufrufe

```
foo(0);
foo(1);
foo(2);
foo(3);
foo(4);
```

(c) Zeilenabfolge

Abbildung 3.1: Beispiele für Code im Kontrollfluss [1]

Der Unterschied dieser Unterkategorien wird bei der Betrachtung des Aufrufs „foo(i)“ und dessen Werthereingabe deutlich. Bei 3.1a wird mithilfe des „while“ Operators die Funktion aufgerufen und die Eingabe erhöht.
Das mittlere Beispiel zeigt die Verwendung einer rekursiven Methode, um das Verhalten darzustellen. Der Eingabeparameter dient hier als Wert für den Funktionsaufruf.
Das letzte Beispiel 3.1c zeigt das gleiche Verhalten durch einfache Aufrufe. Hier wird die Funktion mit Wert im Klartext aufgerufen.

3.1.1 Eigenes Beispiel

Folgende Abbildung zeigt das Beispiel-Verhalten im Kontrollfluss.

```
function istGerade(n: number) {
    for(let i = 0; i <= n; i++) {
        if(i % 2 == 0) {
            console.log("Gerade");
        } else {
            console.log("Ungerade");
        }
    }
}
```

Abbildung 3.2: Beispiel im Kontrollfluss

Um die Unterschiede der verschiedenen Darstellungsformen zu erkennen, ist es sinnvoll die Aufrufe von *„console.log()“* zu betrachten. Diese stellen bei unserem Beispiel die tatsächlich durchzuführende Aktion dar. In Abbildung 3.4 lässt sich erkennen, dass diese Aufrufe durch die Kontrolloperatoren *for* und *if* gesteuert werden.

3.1.2 Vor und Nachteile

Da das Programmieren im Kontrollfluss schnell und einfach funktioniert, eignet sich diese Art gut, um neues Vehalten initial abzubilden.

3.2 Verhalten in der Struktur der Daten

3.2.1 Eigenes Beispiel

3.3 Verhalten in den Daten

3.3.1 Eigenes Beispiel

```
interface Zahl{
    istGerade(): void;
}

class GeradeZahl implements Zahl{
    constructor(private count: number) {}
    istGerade() {
        console.log("Gerade");
        if(this.count != 0) {
            new UngeradeZahl(this.count - 1).istGerade();
        }
    }
}

class UngeradeZahl implements Zahl{
    constructor(private count: number) {}
    istGerade() {
        console.log("Ungerade");
        if(this.count != 0) {
            new GeradeZahl(this.count - 1).istGerade();
        }
    }
}
```

Abbildung 3.3: Beispiel in einer Datenstruktur

```
const daten: (() => void)[] = [
    () => console.log("Gerade"),
    () => console.log("Ungerade")
];

function istGerade(n: number) {
    for(let i = 0; i <= n; i++) {
        daten[i % daten.length]();
    }
}
```

Abbildung 3.4: Beispiel in einer Datenstruktur

4 Ist Refactoring immer sinnvoll?

4.1 Beobachten statt vorhersagen

5 Sicherheit gewinnen, ohne den Code zu verstehen

5.1 Sicherheit durch Tests

Die wohl einfachste Möglichkeit die Korrektheit von Software zu überprüfen, ist das Testen. In der Softwareentwicklung gibt es verschiedene Arten, wie richtiges Verhalten, einzelne Codeabschnitte oder ganze Produkte getestet werden können. Einige dieser Arten werden im folgenden aufgelistet.

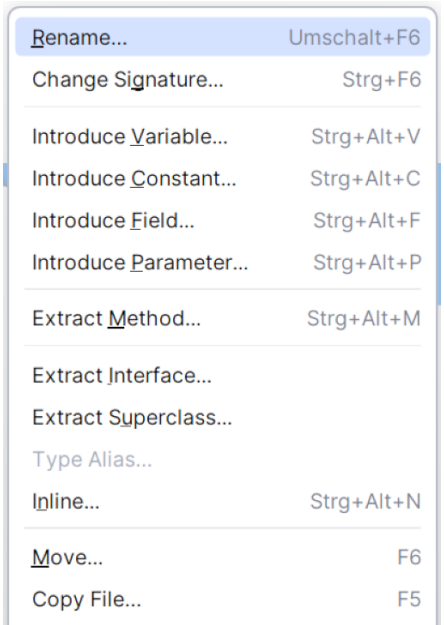
- **Unit Tests:** Testen von bla
- **Whiteboxtests:** Testen von bla

5.2 Sicherheit durch Handwerkskunst

5.3 Sicherheit durch Werkzeuge

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pel-



lentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

5.4 Sicherheit durch formale Verifikation

5.5 Sicherheit durch Fehlertoleranz

Literatur

[1] Christian Clausen. *five lines of code - Clean Code durch gezieltes Refactoring*. Bonn: Rheinwerk Verlag, 2023, S. 311–333. ISBN: 9783836292245.

[2] Melvin E Conway. „How do committees invent“. In: *Datamation* 14.4 (1968), S. 28–31.

[3] Alan MacCormack, Carliss Baldwin und John Rusnak. „Exploring the duality between product and organizational architectures: A test of the “mirroring” hypothesis“. In: *Research policy* 41.8 (2012), S. 1309–1324.

[4] JetBrains s.r.o. *WebStorm, Version 232.10203.14*. Screenshot. Verfügbar unter: `https://www.jetbrains.com/de-de/webstorm/`. 2024.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

Füssen, den 27. Juni 2023 .....

Unterschrift des Verfassers