

Seminar

Refactoring in der Softwareentwicklung

Sommersemester 2024

bei Prof. Dr. Georg Hagel

five lines of code (Christian Clausen)

Thema: Folge 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, consectetur 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

Inhaltsverzeichnis	iii
Abbildungsverzeichnis	iv
Tabellenverzeichnis	v
1 Strukturen in der Softwareentwicklung	1
1.1 Verschiedene Arten von Strukturen	1
1.2 Weitere Einschränkungen	1
2 Arten, wie Code Verhalten spiegelt	2
2.1 Verhalten im Kontrollfluss	2
2.1.1 Eigenes Beispiel	3
2.2 Verhalten in der Struktur der Daten	3
2.2.1 Eigenes Beispiel	3
2.3 Verhalten in den Daten	3
2.3.1 Eigenes Beispiel	3
3 Sicherheit gewinnen, ohne den Code zu verstehen	5
3.1 Sicherheit durch Tests	5
3.2 Sicherheit durch Handwerkskunst	5
3.3 Sicherheit durch Werkzeuge	5
3.4 Sicherheit durch formale Verifikation	6
3.5 Sicherheit durch Fehlertoleranz	6
Literatur	7

Abbildungsverzeichnis

2.1	Beispiele für Code im Kontrollfluss [1]	2
2.2	Beispiel im Kontrollfluss	3
2.3	Beispiel in einer Datenstruktur	3
2.4	Beispiel in einer Datenstruktur	4
3.1	Screenshot WebStorm Refactoring-Tools [4]	5

Tabellenverzeichnis

1.1	Strukturkategorien [1] (korrigierte Form)	1
-----	---	---

1 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.

1.1 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 1.1: 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]

1.2 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]

2 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]

2.1 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 kann man bla sehen (2.1)

```
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 2.1: Beispiele für Code im Kontrollfluss [1]

Der Unterschied dieser Unterkategorien wird bei der Betrachtung des Aufrufs „foo(i)“ und dessen Werthereingabe deutlich. Bei 2.1a 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 2.1c zeigt das gleiche Verhalten durch einfache Aufrufe. Hier wird die Funktion mit Wert im Klartext aufgerufen.

2.1.1 Eigenes Beispiel

```
1 function istGerade(n: number) {  
2   for(let i = 0; i <= n; i++) {  
3     if(i % 2 == 0) {  
4       console.log("Gerade");  
5     } else {  
6       console.log("Ungerade");  
7     }  
8   }  
9 }
```

Abbildung 2.2: Beispiel im Kontrollfluss

2.2 Verhalten in der Struktur der Daten

2.2.1 Eigenes Beispiel

```
1 interface Zahl{  
2   istGerade(): void;  
3 }  
4  
5 class GeradeZahl implements Zahl{  
6   constructor(private count: number) {}  
7   istGerade() {  
8     console.log("Gerade");  
9     if(this.count != 0) {  
10      new UngeradeZahl(this.count - 1).istGerade();  
11    }  
12  }  
13 }  
14  
15 class UngeradeZahl implements Zahl{  
16   constructor(private count: number) {}  
17   istGerade() {  
18     console.log("Ungerade");  
19     if(this.count != 0) {  
20      new GeradeZahl(this.count - 1).istGerade();  
21    }  
22  }  
23 }
```

Abbildung 2.3: Beispiel in einer Datenstruktur

2.3 Verhalten in den Daten

2.3.1 Eigenes Beispiel

```
1  const daten: (() => void)[] = [  
2      () => console.log("Gerade"),  
3      () => console.log("Ungerade")  
4  ];  
5  
6  function istGerade(n: number) {  
7      for(let i = 0; i <= n; i++) {  
8          daten[i % daten.length]();  
9      }  
10 }
```

Abbildung 2.4: Beispiel in einer Datenstruktur

3 Sicherheit gewinnen, ohne den Code zu verstehen

3.1 Sicherheit durch Tests

3.2 Sicherheit durch Handwerkskunst

3.3 Sicherheit durch Werkzeuge

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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. Pellentesque 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,

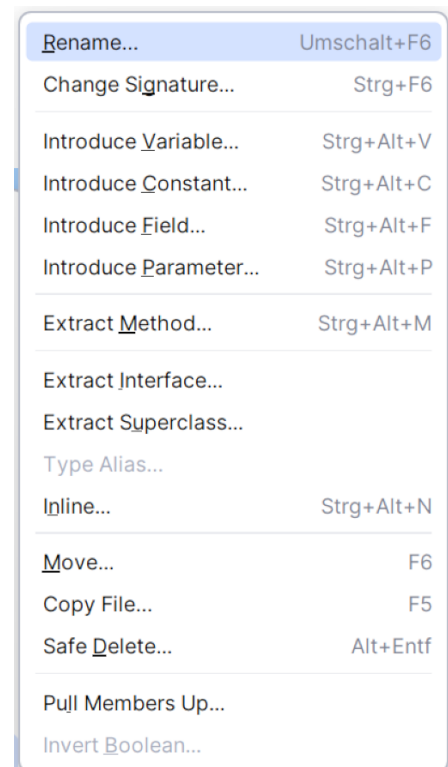


Abbildung 3.1: Screenshot WebStorm Refactoring-Tools [4]

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, consectetur 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.

3.4 Sicherheit durch formale Verifikation

3.5 Sicherheit durch Fehlertoleranz

aa

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