Don't repeat yourself

Inhaltsverzeichnis

1 Definition

Das Programmierprinzip Don't repeat yourself (DRY, Wiederhole dich nicht) besagt, dass Code nicht dupliziert und anschließend nur marginal modifiziert werden sollte.[1]

2 Begründung

Der Grund ist folgender: Wenn eine Folge von Anweisungen mehrfach in sehr ähnlicher Form verwendet wird, liegt diesen Anweisungen ein allgemeineres Prinzip zu Grunde, das explizit ins zugehörige Programm eingearbeitet werden sollte.

Code ohne Duplikate und Quasis-Duplikate von Anweisungen wird häufig als **DRY** bezeichnet. Mit **“to DRY code”** wird der Vorgang bezeichnet, Anweisungs-Duplikate aus dem entsprechenden Code zu entfernen.[2]

Code der DRY ist bietet folgende Vorteile:

⇒ Wartbarkeit (da Änderungen nicht an mehreren, fast identischen Anweisungen vorgenommen werden müssen)

⇒ Robustheit (bei einer Änderung, die an mehreren Stellen vorgenommen werden muss, kommt es häufig zu Inkonsitenzen)

⇒ Lesbarkeit (da weniger Code existiert)

3 Beispiele

3.1 C#-Code, der nicht DRY ist (Autor: Anonym)

Folgendes Beispiel stammt aus einem C#-Tool, das Daten für unterschiedliche Gruppen von Personen verarbeitet. Als das Tool entwickelt wurde, gab es genau zwei Gruppen.

```
// Filtere hier nach Gruppe 1 oder 2
if (this.selectedGroup == 1)
  persons = persons.Where(p => p.IsActive && p.Group == 1);

else if (this.selectedGroup == 2)
  persons = persons.Where(p => p.IsActive && p.Group == 2);
```

Wie man sieht, kann der Code problemlos für 3, 4 oder auch 27 Gruppen erweitert werden.

Wenn man diesen Code allerdings DRY macht, wird er nicht nur kürzer, sondern funktioniert auch noch ohne Änderungen, wenn irgendwann einmal die Anzahl der Gruppen erhöht wird.

```
persons = persons.Where(p => p.IsActive && p.Group ==
this.selectedGroup);
```

3.2 JavaScript-Code, der nicht DRY ist (Autor: Anonym)

```
v_counter1.text = ++v_score_player1;
v_counter1.text = ++v_score_player1;
v_counter1.text = ++v_score_player1;
```

Glücklicher weise soll der Score nur um 3 und nicht um 50 Punkte erhöht werden. DRY wäre folgender Code:

```
v_score_player1 += 3;
v_counter1.text = v_score_player1;
```

3.3 JavaScript-Code, der nicht DRY ist (Autoren: Anyom)

Folgender Code stammt aus einen einfachen JavaScript-Spiel mit 15 Spielsteinen („Coins“). Für jeden einzelnen Stein wurde der zugehörige Code separat ausformuliert:

```
var btnSrc1 = new Image();
var btn1;

var btnSrc2 = new Image();
var btn2;

var btnSrc3 = new Image();
var btn3;
...

function Main()
{□□
...
  btnSrc1.src = 'button1.png';
  btnSrc1.name = 'coin1';
  btnSrc1.onload = loadGfx;
  □
  btnSrc2.src = 'button2.png';
  btnSrc2.name = 'coin2';
  btnSrc2.onload = loadGfx;
  □
  btnSrc3.src = 'button3.png';
  btnSrc3.name = 'coin3';
  btnSrc3.onload = loadGfx;
...
}

function loadGfx(e)
{□□
...
  if(e.target.name = 'coin1'){btn1 = new Bitmap(btnSrc1);}
  if(e.target.name = 'coin2'){btn2 = new Bitmap(btnSrc2);}
  if(e.target.name = 'coin3'){btn3 = new Bitmap(btnSrc3);}
...
}

// etc. pp.
```

Ein Array, ein zusätzliches Attribut `btn` für die `btnSrc`-Objekte und die Verwendung einer Schleife machen diesen Code DRY, d.h. kompakter, lesbarer und änderungsfreundlicher. Zum Beispiel kann die Anzahl der Spielsteine jederzeit problemlos geändert werden, indem der Wert von `NR_OF_COINS`

geändert wird.

```
var bs_array = [], btnSrc;

for (var i = 0; i < NR_OF_COINS)
{ btnSrc = new Image();

  btnSrc.src    = 'button'+i+'.png';
  btnSrc.name   = 'coin'+i;
  btnSrc.onload = loadGfx;

  bs_array[i] = btnSrc;
}

function loadGfx()
{ e.target.btn = new Bitmap(e.target.src); }
```

Anmerkung: Hier sind noch weitere Verbesserungen denkbar, wie z.B. eine bessere Wahl der Objekt- und Attributnamen. Dies hat aber nichts mehr mit dem Prinzip “Don't repeat yourself” zu tun.

3.4 Der Klassiker

Der Non-DRY-Klassiker dürfte wohl folgender sein:

```
if (<Bedingung>)
  return <Ein langer Ausdruck>;
else
  return -<Derselbe lange Ausdruck>;
```

Um diesen Code DRY zu bekommen, führt man entweder eine Hilfsvariable ein:

```
var a = <Ein langer Ausdruck>;
if (<Bedingung>)
  return a;
else
  return -a;
```

Oder – eleganter – man verwendet die If-Then-Else-Funktion `... ? ... : ...` an Stelle der If-Then-Else-Anweisung:

```
return (<Bedingung> ? 1 : -1) * <Ein langer Ausdruck>;
```

Von diesem Klassiker gibt es zahlreiche Varianten, wie z.B. folgende Anweisung, bei der klar wird,

dass der Autor (oder die Autorin) Boolesche Ausdrücke für etwas hält, was nur zum Testen, aber nicht zum Rechnen verwendet werden kann:

```
if (<Boolscher Ausdruck>)
  return true;
else
  return false;
```

Hier schreibt man natürlich:

```
return <Boolscher Ausdruck>;
```

Noch ein Beispiel aus der Praxis. Im zuvor erwähnten C#-Tool findet sich folgender Code:

```
var deactivate =
    (t.State == "gesperrt" || t.State == "inaktiv") ? true : false;

if (deactivate)
    DataModel.Team.Deactivate(id);
```

Hier ist nicht nur die Abbildung von `true` auf `true` und von `false` auf `false` überflüssig, sondern auch noch die Einführung einer Hilfsvariablen. Folgender Code ist kürzer und damit einfacher zu verstehen:

```
if (t.State == "gesperrt" || t.State == "inaktiv")
    DataModel.Team.Deactivate(id);
```

4 Quellen

1. **Hunt, Thomas (2003)**: Andrew Hunt und David Thomas; Der Pragmatische Programmierer; Verlag: Fachbuchverlag Leipzig im Carl Hanser Verlag; ISBN: 3446223096, 978-3446223097; 2003; Quellengüte: 5 (Buch)
2. Quelle fehlt
3. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

5 Siehe auch

1. Wikipedia (EN): DRY (Web)

Kategorien:

Programmierprinzip
HowTo