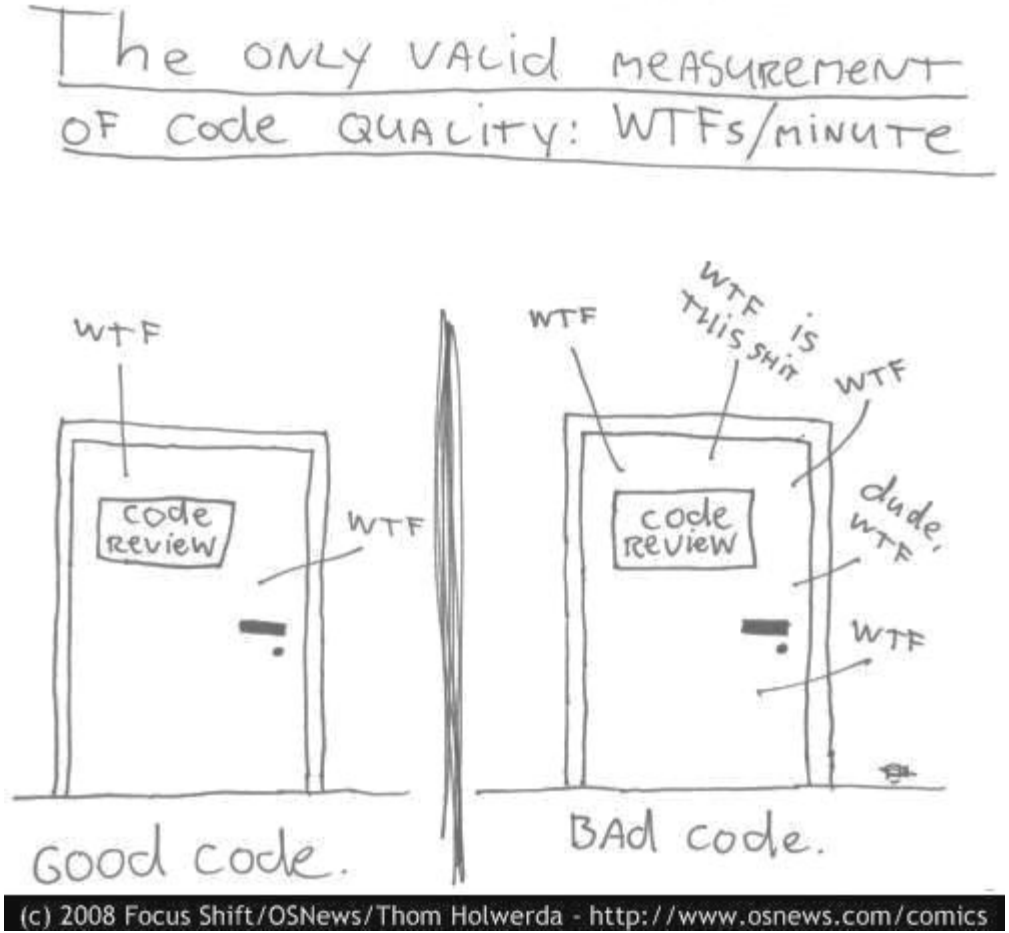
Code Geruch

It's not a Bug, it's a Code Smell



https://twitter.com/GeorgeTakei/status/397647996361031 68

Über mich

- Computervisualistik-Studium an Uni Magdeburg
- Wissenschaftlicher Mitarbeiter an Uni Magdeburg
- Webentwickler
- Software-Ingenieur bei XITASO in MD

Java C++ C# PHP Python JavaScript TypeScript SVN HG Git OpenGL OpenAL Avro Maven Hadoop Cloudera Antlr Bower Grunt Gulp Node.js npm Docker JUnit Jasmine Protractor Mockito MySQL PDO JQuery Konva KineticJs CSS LESS SASS Bootstrap Material Design AngularJS Angular Twig Symfony JMeter Sonar Bamboo Jenkins Confluence Fisheye Sentry Bugzilla Redmine Facebook PHP API Joomla Typo3 Wordpress SallyCMS SSL/TLS Security SEO NetBeans Eclipse Visual Studio IntelliJ IDEA Git Flow Jira Confluence Fisheye

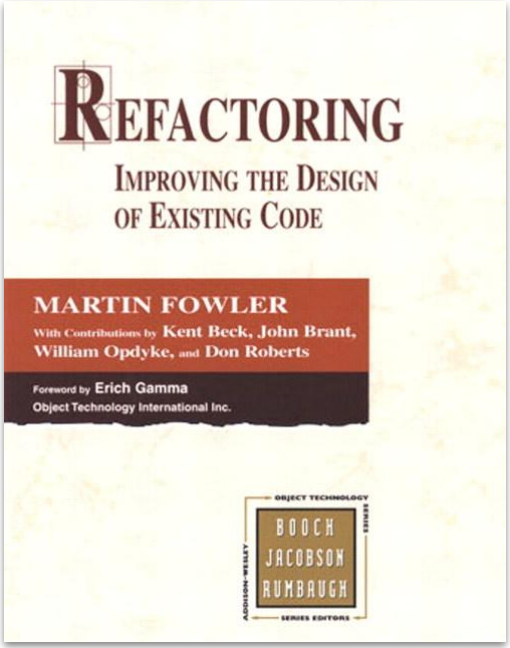
Themen

1. Hintergrund, Begriff, Relevanz
2. Code Smells Überblick, Beispiele
3. Ursachen, Beseitigung, Vorbeugung

Hintergrund, Begriff, Relevanz

‘Code Smell’ Hintergrund

- **Begriffsprägung** durch **Kent Beck** im WardsWiki Ende der 90er
- **Verbreitung** durch **Martin Fowler**s Buch ‘*Refactoring: Improving the Design of Existing Code*’ (1999)



‘Code Smell’ Begriff

“*A code smell is **a hint** that something has **gone wrong** somewhere in your code. Use the smell to track down the problem.*” [Kent Beck]

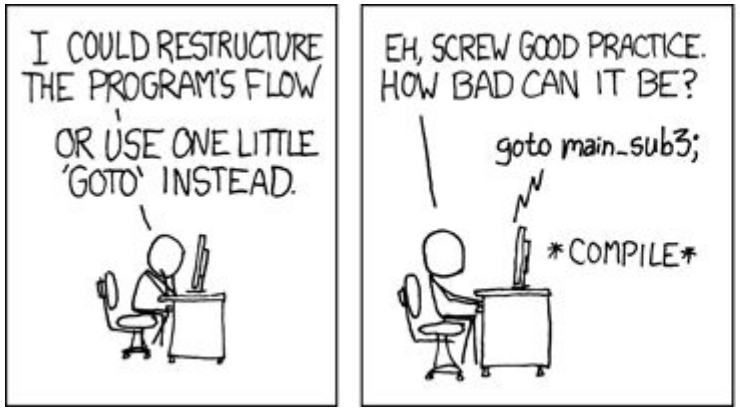
“*A code smell is a **surface indication** that usually corresponds to a **deeper problem** in the system.*” [Martin Fowler]

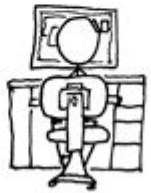
Code Smell?

- **Code Smell != Bug**
  Code Smell == Warning Sign
- subjektiv!
  abhängig von Sprache, Methodik, Programmierer/Team...

Relevanz

- Entwicklungsgeschwindigkeit (Wartbarkeit, Erweiterbarkeit…)
- Bug-Risiko
- Coding Joy :-)
- Ehre?
- Raptoren…





Quelle: https://xkcd.com/292/

Code Smells Überblick

Kategorisierung

I Method-level Code Smells

Initialisierung, Aufgabe, Komplexität, Kontext

II Class-level Code Smells

Initialisierung, Modellierung, Aufgabe, Komplexität, Kontext

III Generelle Code Smells

Duplizierungen, Divergenz, Kapselung, Naming, Kommentare

I METHOD-level Code Smells

I Method-Level – Initialisierung

- **Lost Intent**
  Methodennamen beschreibt nicht den Zweck der Methode

```
# LOST INTENT #

public void onConfirmButtonClick() {
  this.service.print();
}

---

public void execute() {
  // ...
}
```

I Method-Level – Initialisierung

- *Lost Intent*
  Methodennamen beschreibt nicht Zweck der Methode
- **Long Param List**
  zu viele Parameter übergeben

```
# LONG PARAM LIST #

public void saveUser(
  string email,
  string userName,
  string country,
  string city,
  string street,
  string birthday,
  string phone) {
  //...
}
```

I Method-Level – Initialisierung

- *Lost Intent*
  Methodennamen beschreibt nicht Zweck der Methode
- *Long Param List*
  zu viele Parameter übergeben
- **Dead Code**
  ungenutzte Methode

Quelle: refactoring.guru

I Method-Level – Aufgabe

- **God Method**
  Methode tut zuviel

*It should XXX **and/or** XXX.*

`fetchPosts()` → check login status **and** fetch posts **and** parse JSON data into post objects.

***If** XXX, it should do XXX.*

`fetchPosts()` → fetch posts, **if** the user is logged in. **If** it fetches posts, parse JSON data into posts objects.

I Method-Level – Aufgabe

- *God Method*
  Methode tut zuviel
- **Lazy Method**
  Methode tut zu wenig

```
# LAZY METHOD #

function convertToLowerCase(name) {
  return name.toLowerCase();
}
```

I Method-Level – Aufgabe

- *God Method*
  Methode tut zuviel
- *Lazy Method*
  Methode tut zu wenig
- **Middle Man**
  Methode delegiert nur Aufgaben

I Method-Level – Komplexität

- **Contrived Complexity**
  zu komplexe Implementierung

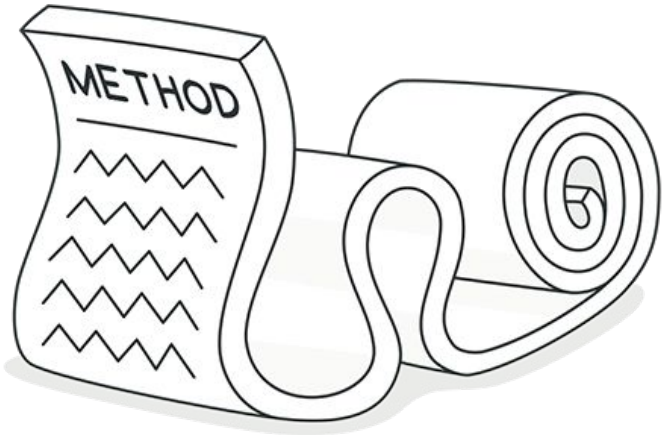
I Method-Level – Komplexität

- *Contrived Complexity*
  zu komplexe Implementierung
- **Cyclomatic Complexity**
  zu verzweigt u/o loopy

```
# CYCLOMATIC COMPLEXITY #

function putAnswers() {
  let answers = [];

  if (isMultiple) {
     …
  } else {
     if (isTrueFalse) {
        …
     } else {
        …
     }
  }

  return answers;
}
```

I Method-Level – Komplexität

- *Contrived Complexity*
  zu komplexe Implementierung
- *Cyclomatic Complexity*
  zu verzweigt u/o loopy
- **Long Method**
  zu viele Zeilen



Quelle: refactoring.guru

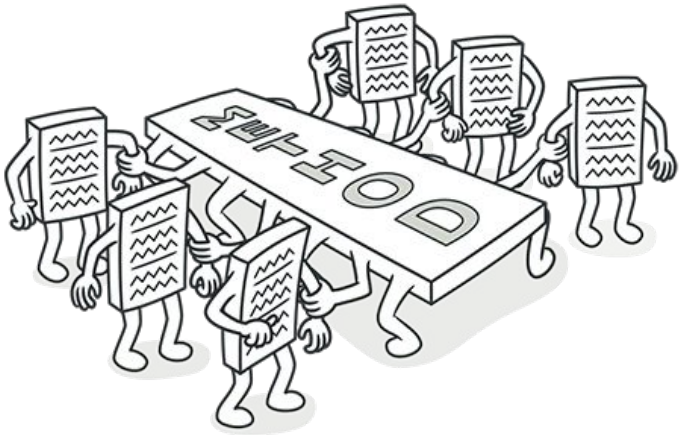
I Method-Level – Kontext

- **Feature Envy**
  greift mehr auf fremde Daten zu als auf eigene

```
# FEATURE ENVY #

class User {
  private contactInfo = new ContactInfo();

  public getAddress() {
    return
      contactInfo.getStreetName() . ' ' .
      contactInfo.getStreetNr() . ', ' .
      contactInfo.getZip() . ', ' .
      contactInfo.getCity();
  }
}
```

I Method-Level – Kontext

- *Feature Envy*
  greift mehr auf fremde Daten zu als auf eigene
- **Inappropriate Intimacy**
  zuviel Abhängigkeit von anderer Methode



Quelle: refactoring.guru

I Method-Level – Kontext

- *Feature Envy*
  greift mehr auf fremde Daten zu als auf eigene
- *Inappropriate Intimacy*
  zuviel Abhängigkeit von anderer Methode
- **Black Sheep**
  Methode deutlich anders als die anderen Klassenmethoden

```
# BLACK SHEEP #

class StringUtil {
  public String pascalCase(String str) {
    ...
  }
  public String camelCase(String str) {
    ...
  }
  public String replaceUmlauts(String str) {
    ...
  }
  public String log (String str) {
    ...
  }
}
```

II CLASS-level Code Smells

II Class-Level – Initialisierung

- **Lost Intent**
  Klassenname beschreibt nicht Bedeutung der Klasse

```
# LOST INTENT #

class AbcObject

class DefManager

class YourMotherHandler

class GhiController

class JklData
```

II Class-Level – Initialisierung

- *Lost Intent*
  Klassenname beschreibt nicht Bedeutung der Klasse
- **Temporary field**
  Class member nur in einer Methode in Verwendung

```
# TEMPORARY FIELD #

class User {
  private id;
  private privileges;
  private name;

  public delete() {
    doDelete(this.id);
  }
  public update() {
    doUpdate(this.id, this.privileges)
  }
  public notify() {
    message = this.name . '...';
    notifyService->sendSms(message);
  }
```

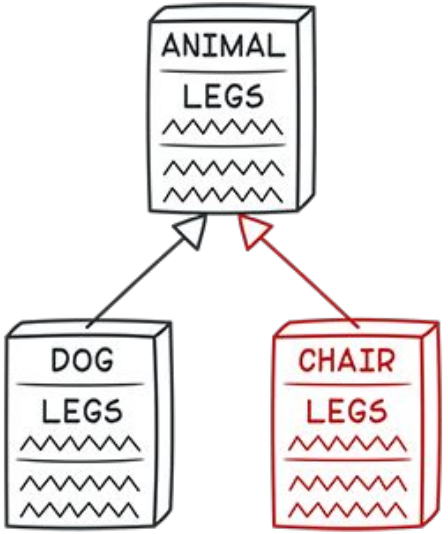
II Class-Level – Modellierung

- **Primitive Obsession**
  Modellierung mit primitiven Datentypen statt Klasse

```
# PRIMITIVE OBSESSION #

public addressUsa() {
  address = new Array();
  address['streetNo'] = 2074;
  address['streetName'] = 'JFK street';
  address['zipCode'] = '507874';

  return address['streetName']. ' '.
         address['streetNo']. ' '.
         address['zipCode'];
}
```

II Class-Level – Modellierung

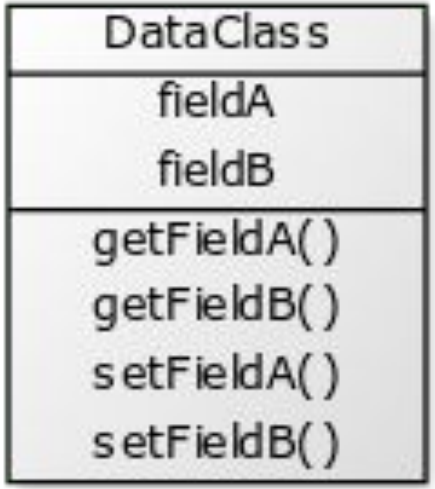
- *Primitive Obsession*
  Modellierung mit primitiven Datentypen statt Klasse
- **Refused Bequest**
  geringes Nutzen der Klasse, von der geerbt wird



Quelle: refactoring.guru

II Class-Level – Modellierung

- *Primitive Obsession*
  Modellierung mit primitiven Datentypen statt Klasse
- *Refused Bequest*
  geringes Nutzen der Klasse, von der geerbt wird
- **Data Class**
  Klasse enthält nur Daten, keine Methoden



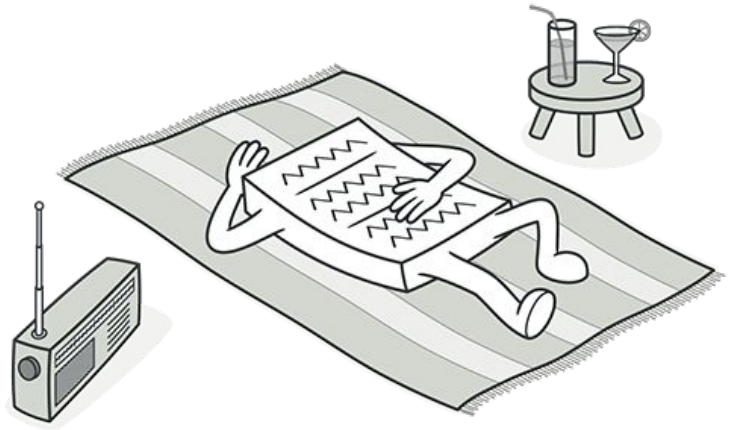
II Class-Level – Aufgabe

- **God Object**
  Klasse tut zuviel (SRP!)

```
# GOD OBJECT #

  class Book {
      function getTitle() {
          return "A Great Book";
      }

      function getAuthor() {
          return "John Doe";
      }

      function turnPage() {
          // pointer to next page
      }

      function printCurrentPage() { ... }
  }
```

II Class-Level – Aufgabe

- *God Object*
  Klasse tut zuviel (SRP!)
- **Lazy Object**
  Klasse tut zu wenig



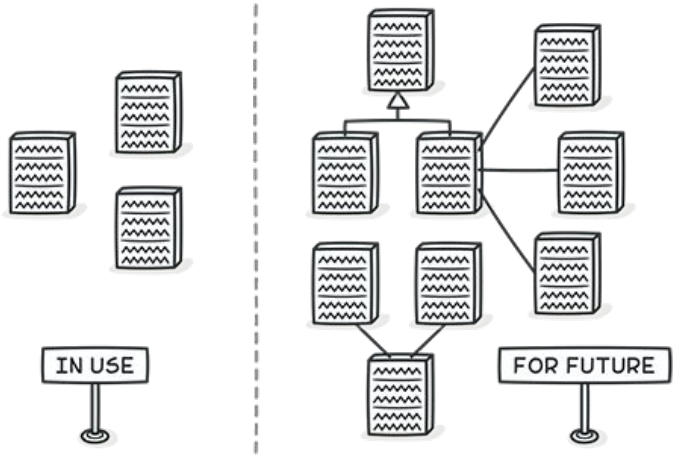
Quelle: refactoring.guru

II Class-Level – Aufgabe

- *God Object*
  Klasse tut zuviel (SRP!)
- *Lazy Object*
  Klasse tut zu wenig
- **Middle Man**
  Klasse delegiert nur Aufgaben

II Class-Level – Komplexität

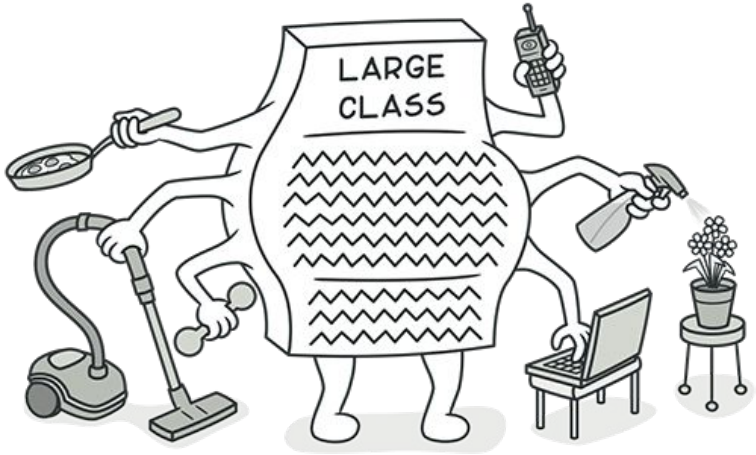
- **Speculative Generality**
  zu komplex entworfen für mögliche, zukünftige Features



Quelle: refactoring.guru

II Class-Level – Komplexität

- *Speculative Generality*
  zu komplex entworfen für mögliche, zukünftige Features
- **Large Class**
  zu viele Zeilen



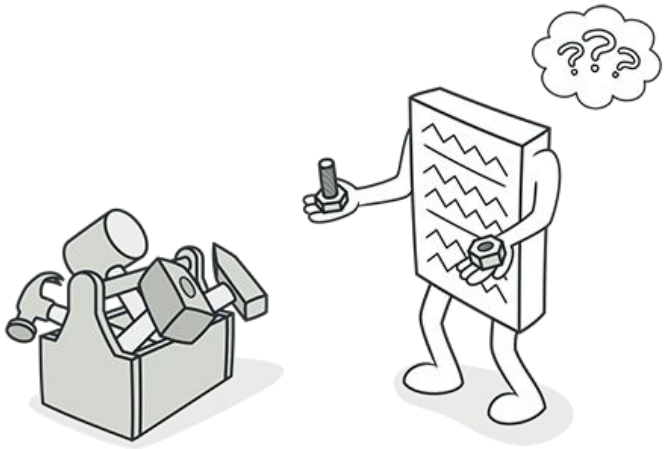
Quelle: refactoring.guru

II Class-Level – Kontext

- **Indecent Exposure**
  interne Klassendaten public
  statt private

II Class-Level – Kontext

- *Indecent Exposure*
  interne Klassendaten public statt private
- **Incomplete Library Class**
  Methode nicht in der passenden Library



Quelle: refactoring.guru

III GENERELLE
Code Smells

III Generell – Duplizierungen

- **Duplicated Code**
  fast identischer Code in verschiedenen Klassen

```
# DUPLICATE CODE #

void printInformation() {
  printTitle();
  System.out.println("name: " + name);
  System.out.println(
    "address: " + getAddress());
}

void printAdressBook() {
  for(int i = 1; i <= book.length; i++) {
    System.out.println(
      "name: " + book[i].name);
    System.out.println(
      "address: " + book[i].getAddress());
  }
}
```

III Generell – Duplizierungen

- *Duplicated Code*
  fast identischer Code in verschiedenen Klassen
- **Combinatorial Explosion**
  Duplikate mit verschiedenen Parameterkombinationen

III Generell – Duplizierungen

- *Duplicated Code*
  fast identischer Code in verschiedenen Klassen
- *Combinatorial Explosion*
  Code, der fast das gleiche macht
- **Oddball Solution**
  dasselbe Problem auf verschiedene Arten gelöst

Quelle: www.xkcd.com/1474/

III Generell – Divergenz

- **Solution Sprawl**
  eine Funktionalität über
  multiple Klassen / Methoden
  verstreut

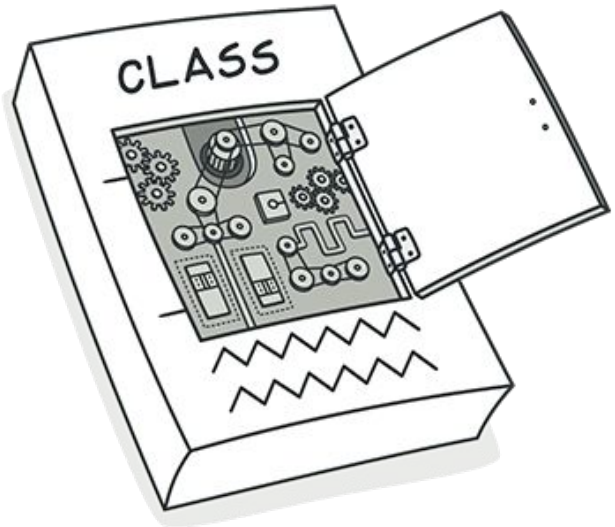
III Generell – Divergenz

- *Solution Sprawl*
  eine Funktionalität über multiple Klassen / Methoden verstreut
- **Data Clumps**
  Variablengruppen, die wiederholt zusammen übergeben werden

```
# DATA CLUMPS #

public static void main(args[]) {
  String firstName = args[0];
  String lastName = args[1];
  Integer age = new Integer(args[2]);
  String gender = args[3];
  String occupation = args[4];
  String city = args[5];

  welcomeNew(
    firstName, lastName, age, gender, occupation, city);
}

public void welcomeNew(
  firstName, lastName, age, gender, occupation, city) {
  System.out.printf("Hi %s %s, a %d-year-old %s from %s
    who works as a%s\n",
    firstName, lastName, age, gender, city, occupation);
}
```

III Generell – Kapselung

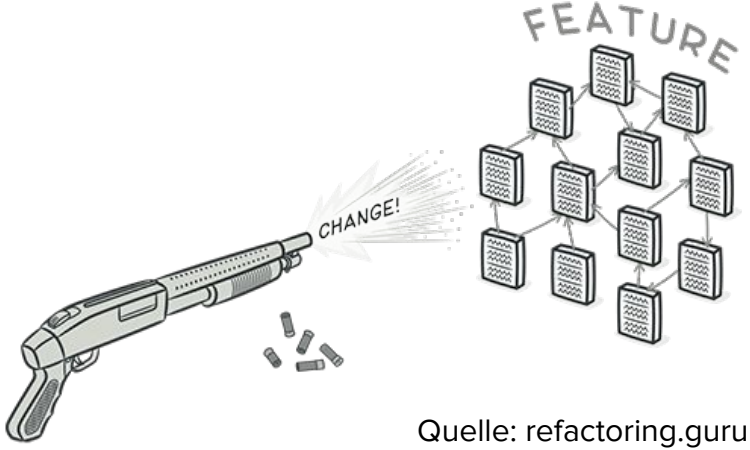
- **Divergent Change**
  Änderung an Klasse erfordert
  Anpassung vieler Methoden



Quelle: refactoring.guru

III Generell – Kapselung

- *Divergent Change*
  Änderung an Klasse erfordert Anpassung vieler Methoden
- **Shotgun Surgery**
  zu viele Klassen anzupassen für eine Funktionalität



Quelle: refactoring.guru

III Generell – Aussagekraft 1

- **Zweck unklar**
  Name nicht zweck-beschreibend

```
# ZWECK UNKLAR #

int d; // abgelaufene Zeit in Tagen

public List<int[]> getThem() { ... }

public void copyChars(
  char a1[], char a2[]) { ... }

for (int j=0; j<34; j++) {
  s += (t[j]*4)/5;
}
```

III Generell – Aussagekraft 1

- *Zweck unklar*
  Name nicht zweck-beschreibend
- **Irreführung**
  zweideutige Namen

```
# IRREFÜHRUNG #

int a = l;
if (0 == l) a = 01;
else l = 01;
```

III Generell – Aussagekraft 2

- **Codierungen in Namen**
  Typ in Name, Member-Präfixe, Interface-Dekorationen

```
# CODIERUNGEN IN NAMEN #

PhoneNumber phoneString;

private String m_dsc; // Beschreibg

interface IShapeFactory { ... }
```

III Generell – Aussagekraft 2

- *Codierungen in Namen*
  Typ in Name, Member-Präfixe, Interface-Dekorationen
- **Unaussprechlichkeit**
  durch Abkürzungen

```
# CODIERUNGEN IN NAMEN #

class DtaRcrd102 {
  private Date genymddhms;
  private final String pszqint;
  ...
}
```

III Generell – Aussagekraft 3

- *keine Naming Conventions*
  irreguläre Bezeichnungen für Methoden, Variablen

- Klasse kein Substantiv

- Methode kein Verb (+ evtl. Substantiv)

- Abweichung von is/has, get/set

- Groß/Kleinschreibung

III Generell – Aussagekraft 3

- *keine Naming Conventions*
  irreguläre Bezeichnungen für Methoden, Variablen
- **Inkonsistenz**
  unterschiedliche Wörter für selbes Konzept

```
# INKONSISTENZ #

retrieveAbc() { … }

fetchHij() { … }

getOpq() { … }
```

III Generell – Aussagekraft 3

- *keine Naming Conventions*
  irreguläre Bezeichnungen für Methoden, Variablen
- *Inkonsistenz*
  unterschiedliche Wörter für selbes Konzept
- **Wortspiele**
  gleiche Wörter für unterschiedliche Konzepte

```
# WORTSPIELE #

add() statt insert(), append()
```

III Generell – Kommentare

- Offensichtlichkeit
- Unverständlicher Text
- Denglisch
- Chaotisch
  (Parameter, Returns…)
- Veraltet
- Auskommentierter Code

```
/**
 * Replaces with spaces
 * the braces in cases
 * where braces in places
 * cause stasis.
 */

$str = str_replace(
  array("\{","\}")," ",$str
);
```

“The proper use of comments is to compensate for our failure to express our self in code”

[„Clean Code: A Handbook of Agile Software Craftsmanship” Robert C. Martin]

Entstehung, Beseitigung, Vorbeugung

Wie Code Smells entstehen

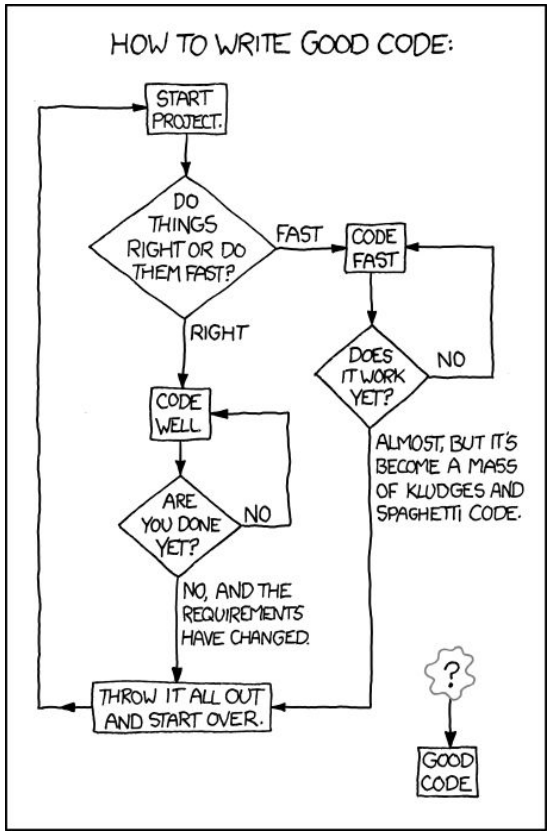
- Zeitdruck
- fehlendes Wissen um Coding Conventions / Design Patterns
- fehlendes Team Commitment
- Solo Coding
- MacGyver Programmierer [t3n]

Code Smells beseitigen

- Commitment im Team vereinbaren
- Linter Tools, statische Code Analyse
- methodisches Refactoring
- “Boy Scouting” - Pfadfinder-Regel
- “Smell of the Week”
- Tatortreiniger [t3n] einstellen ;-)

Code Smells verhindern

- Editor Config
- Linter Tools / Build Barriers
- Code Reviews / Pull Requests
- Technical Debt Tracking
- Refactoring Coding Dojos



Quelle: https://xkcd.com/844/

Mythen

1. Code Smells werden durch **Wartung** und **Evolution** eingeschleppt.

2. **Refactoring** beseitigt Code Smells.

3. Nur **Anfänger** verursachen Code Smells.

[When and Why Your Code Starts to Smell Bad 2015]

“AnY fOol cAn wRitE cOde thAt a **cOmPutEr** cAn uNDersTanD. GoOd prOgrAmMErs wrITe coDe tHat **hUMans** cAn UndeRstaNd.”

[Martin Fowler, Refactoring: Improving the Design of Existing Code, 1999]

Jana Sieber

twitter.com/ja_sie | www.jasie.de/codegeruch/

Code-Beispiele

- Wikipedia: **Common code smells**
  en.wikipedia.org/wiki/Code_smell#Common_code_smells
- Mohamed Aladdin: **Write clean code and get rid of code smells with real life examples**, 2018
  codeburst.io/write-clean-code-and-get-rid-of-code-smells-aea271f30318
- Refactoring Guru: **Code Smells**
  refactoring.guru/refactoring/smells
- Codemanship: **Code Smell Of The Week**
  www.youtube.com/user/parlezuml/search?query=code+smell

Code Smell Causes and Treatments:
https://refactoring.guru/refactoring/smells

Quellen

- Martin Fowler: **Refactoring: Improving the Design of Existing Code**, 1999
  https://martinfowler.com/books/refactoring.html
- Robert C Martin: **Clean Code - Refactoring, Patterns und Techniken** [...], 2009
  https://www.mitp.de/IT-WEB/Programmierung/Clean-Code.html

- Jeff Atwood: **Code Smells**, 2006
  blog.codinghorror.com/code-smells/
- Dino Esposito: **Sharpen your sense of (code) smell**, 2018
  blog.jetbrains.com/dotnet/2018/06/18/sharpen-sense-code-smell/

- Industrial Logic, Inc.: **Smells to Refactorings Quick Reference Guide**, 2005
  industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf
- Tufano et al: **When and Why Your Code Starts to Smell Bad**, 2015
  http://www.cs.wm.edu/~denys/pubs/ICSE'15-BadSmells-CRC.pdf