ASOS - Antipatterns

Smell Code

- Čo je smell code

  - výsledok zlého/nesprávneho programovania

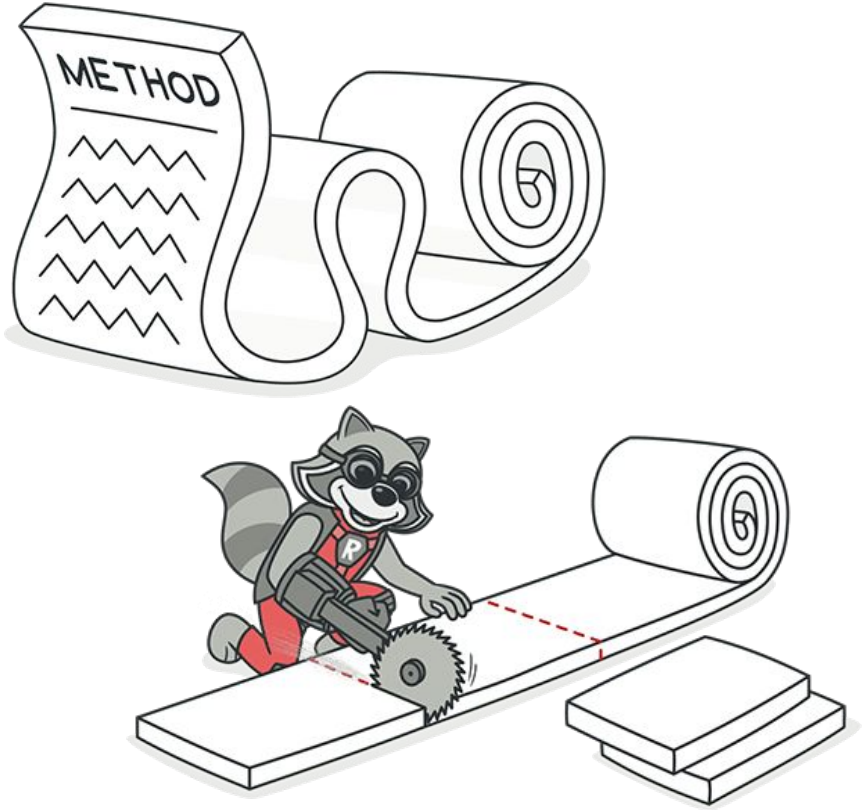
- Vznik
  - neschopnosť napísať kód v súlade so štandardmi, neochota refaktorovať už existujúci kód
- Dôsledky
  - neefektívny, zložitý

— What? How can code "smell"??

— Well it doesn't have a nose... but it definitely can stink!

Kategórie smell kódu

- Bloaters
  - napr. Long Method
- Object-Orientation Abusers
  - napr. Switch Statements
- Change Preventers
  - napr. Divergent Change
- Dispensables
  - napr. Comments
- Couplers
  - napr. Message Chains



Najbežnejšie prípady smell kódu

- duplicate code => Dispensables
- dead code=> Dispensables
- long methods => Bloaters
- comments => Dispensables
- unnecessary primitive variables => Dispensables

Long Method

```
showChapter(chapter, tutorial) {
  if(chapter.orderIndex == 1) {
    this.redirectToRoot()
  }

  let higherChapters = chapter.items.filter(item => item.higher)
  let lowerTutorials = chapter.items.filter(item => !item.higher)

  let title = chapter.title
  let description = chapter.description

  if(chapter.lowerItem) {
    this.nextPath = this.getChapterPath(this.tutorial, chapter.lowerItem)
    this.nextLink = this.nextPath
  } else if(!!tutorial.pathId && !!lowerTutorials) {
    this.nextPath = this.getTutorialPath(lowerTutorials[0])
    this.nextLink = this.nextPath
  } else if (chapter.last && !tutorial.footLinks.length > 0) {
    this.nextPath = tutorial.footLinks.url
  }

  if(higherChapters.length === 0) {
    this.prevLink = this.getChapterPath(tutorial, higherChapters)
  } else if (!!tutorial.pathId && tutorial.higherItem) {
    let higherTutorial = tutorial.higherItem
    lastChapter = higherTutorial.chapters[higherTutorial.chapters.length-1]

    if (lastChapter.orderIndex == 1) {
      this.prevLink = this.getTutorialPath(higherTutorial)
    } else {
      this.prevLink = this.getChapterPath(higherTutorial, higherChapters)
    }
  }

  this.storeProps = {
    checkableType: 'Chapter',
    checkableId: chapter.id,
    checkboxes: this.signedIn ? currentUser.getCheckboxesFor(chapter) : []
  }

  this.setStore()

  this.modal = this.getModalChapter(chapter) || this.getModalTutorial(tutorial)
}
```

Pred

```
showChapter(chapter, tutorial) {
  if(chapter.orderIndex == 1) {
    this.redirectToRoot()
  }

  this.setNextPathLink(chapter, tutorial);
  this.setPrevLink(chapter, tutorial, higherChapters)
  this.setStore(this.getStoreProps(chapter, currentUser))

  this.modal = this.getModal(chapter, tutorial)
}
```

Po

Duplicate code

```
function calculateTax(subtotal, country, state, taxrates) {
  let taxrate;
  if(country === 'US') {
    taxrate = taxrates[state];
  } else {
    taxrate = taxrates[country];
  }
  return subtotal + subtotal * taxrate;
}

function findTimeZone(country, state, zones) {
  let timezone;
  if(country === 'US') {
    timezone = zones[state];
  } else {
    timezone = zones[country];
  }
  return timezone;
}
```

Pred

```
function calculateTax(subtotal, country, state, taxrates) {
  return subtotal +
    subtotal * USvsNonUSLookup(country, state, taxrates);
}

function findTimeZone(country, state, zones) {
  return USvsNonUSLookup(country, state, zones);;
}

function USvsNonUSLookup(country, state, lookup) {
  let foundValue;
  if(country === 'US') {
    foundValue = lookup[state];
  } else {
    foundValue = lookup[country];
  }
  return foundValue;
}
```

Po

Comments

```
// valid email regular expression
var regex = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
if(regex.test(input.value)) {
  user.email = input.value
} else {
  showEmailError()
}
```

Pred

```
if(isEmailAddress(input.value)) {
  user.email = input.value
} else {
  showEmailError()
}


function isEmailAddress(possibleEmail) {
  var regex = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  return regex.test(possibleEmail);
}
```

Po

Unnecessary primitive variables

```
public class CheckingAccount
{
    public int AccountNumber { get; set; }
    public string CustomerName { get; set; }
    public string Email { get; private set; }
    public string Address { get; set; }
    public int ZipCode { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }
    public string SocialSecurityNumber { get; set; }
    public DateTime ActiveDate { get; set; }
    public string GetSSNLast4Digit()...
}
```

Pred

```
public class CheckingAccount
{
    public int AccountNumber { get; set; }
    public string CustomerName { get; set; }
    public string Email { get; private set; }
    public Address Address { get; set; }
    public DateTime ActiveDate { get; set; }
    public SocialSecurity SocialSecurity { get; set; }
}
```

Po

God Object

- Čo je to?
  - objekt, ktorý odkazuje na veľké množstvo odlišných typov
- Prečo je to problém?
  - veľká zodpovednosť/pokrytie, častá aktualizácia, veľké množstvo zmien
- Ako to riešiť?
  - stratégia rozdeľ a panuj

Príklad ako nie

Employee.java

```
public class Employee {

    private Integer ID;
    private Date BirthDate ;
    private String FirstName;
    private String LastName;
    private String SpouseFirstName;
    private String SpouseLastName;
    private String ChildOneFirstName;
    private String ChildOneLastName ;

    /*  Every child is a new variable, not good
    ...
    public String ChildXXXFirstName;
    public String ChildXXXLastName;
    public String ChildYYYFirstName;
    public String ChildYYYLastName;
    ...
    */
```

Príklad ako áno


```java
public class Person {
    private String FirstName;
    private String LastName;
    private Date BirthDate;
```

```java
public class Employee extends Person{
    private Integer ID;
```

```java
public class Child extends Person{
    private Integer EmployeeID;
```

```java
public class Spouse extends Person{
    private Integer EmployeeID;
```

Napevno inicializované hodnoty

Komplikácie:

- databáza
- úložisko
- cloud deployment

```java
public DataSource getDataSource() {
    DataSourceBuilder<?> dataSourceBuilder = DataSourceBuilder.create();
    dataSourceBuilder.url("jdbc:mysql://localhost:3306/asos");
    dataSourceBuilder.username("root");
    dataSourceBuilder.password("root");
    return dataSourceBuilder.build();
}
```

Riešenie:

- application.properties
- .env

```
### Database configuration ###
spring.datasource.url           =   jdbc:mysql://localhost:3306/asos
spring.datasource.username      =   root
spring.datasource.password      =   root

### Cloud configuration ###
spring.datasource.cloud.url      =   ${DB_CONNECTOR}://${DB_URL}:${DB_PORT}/${DB_SCHEMA}
spring.datasource.cloud.username =   ${DB_USERNAME}
spring.datasource.cloud.password =   ${DB_PASSWORD}
```

```
### Deployment configuration ###
PUBLIC_URL          =   /myapp/homepage
GENERATE_SOURCEMAP  =   false
```

Nesprávne catch bloky

```
try {
    Double.parseDouble(price);
} catch (NumberFormatException e) {
}
```

```
try {
    Double.parseDouble(price);
} catch (Exception e) {
    e.printStackTrace();
```

```
try {
    Double.parseDouble(price);
} catch (NumberFormatException e) {
    e.printStackTrace();
}
```

Dôsledky:

- komplikovanie debugovania
- zánik niektorých hodnôt
- neuniformita chýb

Riešenie:

- nenechávať prázdne catch bloky
- odchytávanie špecifických chýb
- zabalovanie hlášok do uniformnej odpovede

```
try {
    return Double.parseDouble(price);
} catch (NumberFormatException e) {
    throw new CustomException("This is my custom message", 404);
}
```

Spaghetti code antipattern

- 70-te roky 20 storočia.
- Nevhodná štruktúra zdrojového kódu -> nezrozumiteľnosť/ťažšia udržateľnosť.
- Nespoľahlivá architektúra projektu.
- nevyriešenie -> plytvaniu zdrojmi.
- GOTO, nedodržaná/zle navrhnutá architektúra, zastarané postupy vývoja, programátorské chyby.
- Prevencia: komentovanie/dokumentácia, pravidelné testovanie.

```
LOOP:do {
   if( a%2==0 && a < 20) {
      a = a + 1;
      cout << "value of a: " << a << endl;
      goto LOOP2;
   }
}
while( a < 20 );

 LOOP2:do{
   if( a%2==1  && a < 20) {
      a = a + 1;
      cout << "value of a: " << a << endl;
      goto LOOP;
   }

}
while( a < 20 );
```

```
do {
   a = a + 1;
   cout << "value of a: " << a << endl;
}
while( a < 20 );
```

Fear of adding classes

- Viac tried == komplikovaný návrh/dizajn
- Jednoduché/zrozumiteľné > veľké/zložité

```java
public class Reservation {

    private int numberOfRooms;
    private int numberOfBathrooms;
    private boolean pool;
    private boolean wellness;
    private boolean privateChef;
    public boolean playGround;
    public String houseType;

    public String printPriceInfo() {
        if (houseType.equals("Apartment")) {
            double pool = isPool() ? 0.65 * 50 : 0;
            double price = numberOfBathrooms + numberOfRooms;
            return "Your house type is Apartment and price is " + price;
        } else if (houseType.equals("Cottage")) {
            double pool = isPool() ? 0.65 * 50 : 0;
            double price = numberOfBathrooms + numberOfRooms * 2.8;
            return "Your house type is Cottage and price is " + price;
        } else {
            double pool = isPool() ? 0.65 * 50 : 0;
            double price = numberOfBathrooms + numberOfRooms * 3.9;
            return "Your house type is House and price is " + price;
        }
    }
```

```java
public abstract class HouseType {

    private int numberOfRooms;
    private int numberOfBathrooms;
    private boolean pool;

    public String printPriceInfo() { return null; }
```

```java
public class Apartment extends HouseType{

    private boolean wellness;
    private boolean privateChef;

    @Override
    public String printPriceInfo(){
        double pool = hasPool() ? 0.65 * 50 : 0;
        double price = getNumberOfBathrooms() + getNumberOfRooms() * 3.9;
        return "Your house type is Apartment and price is " + price;
    }
```

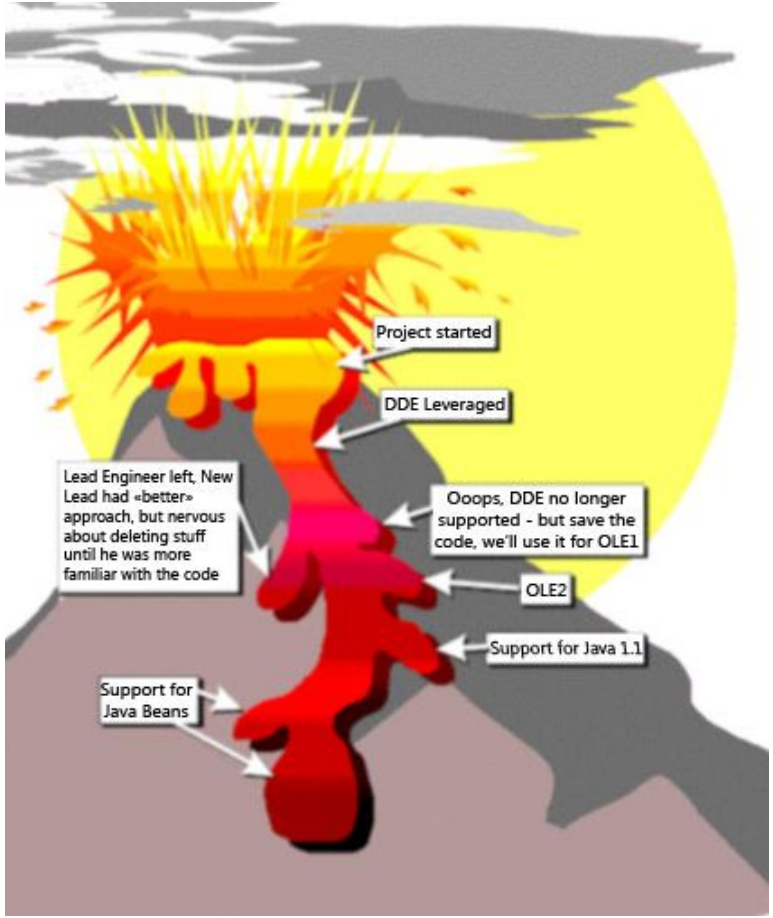
```java
public class Cottage extends HouseType{

    public boolean playGround;

    @Override
    public String printPriceInfo(){
        double pool = hasPool() ? 0.65 * 50 : 0;
        double price = getNumberOfBathrooms() + getNumberOfRooms() * 2.8;
        return "Your house type is Cottage and price is " + price;
    }
```

Lava Flow Antipattern

- Výskum -> produkcia
- “Lava-like” prúdy
- Viac prístupov k riešeniu
- Prestriedanie developerov
- Neriadi sa architektúrou, urýchlený vývoj
- Časté menenie zámeru
- Bloky ktoré nie sú súčasťou kódu
- Zakomentované bloky kódu
- Komplikované/dôležité
- Zaberá zdroje



```java
public class House extends HouseType {

    /*  Nobody knows what is this doing. Matus(left the company long time ago)
        wrote it and Balo(left company half year ago) edited it.
        Don't delete it, but I don't think it is used anywhere*/
    public House(int numberOfBathrooms) {
        super(numberOfBathrooms);
        darkMagic();
    }
```

```java
public abstract class HouseType {

    private int numberOfRooms;
    private int numberOfBathrooms;
    private boolean pool;

    public HouseType(int numberOfRooms, int numberOfBathrooms, boolean pool) {...}

    public HouseType(int numberOfBathrooms) { this.numberOfBathrooms = numberOfBathrooms; }

    /*
    public int bathroomCost() {
        if (getNumberOfBathrooms() == 0) {
            int a = 14;int b = 12;
            someMagicHere();
            somethingDoingHere();
            return getNumberOfBathrooms() * 12 + 4 + a - b;
        } else if (getNumberOfBathrooms() == 1) {
            somethingDoingThere();
            someMagicThere();
            int c = 2646;int a = 48;
            return getNumberOfBathrooms() * 12 + 4 * c / a;
        } else if (getNumberOfBathrooms() == 2) {
            somethingDoingHereAndThere();
            someMagicHereAndThere();
            int p = 1;int s = 16;
            return getNumberOfBathrooms() * 12 + 4 + p / s;
        } else if (getNumberOfBathrooms() == 3) {
            return getNumberOfBathrooms() * 12 + 4;
        }
        return 65;
    }*/
```

Poltergeist Class Antipattern

"I'm not exactly sure what this class does, but it sure is important!"

- zbytočná trieda
- pridáva na zložitosti kódu (neprehľadný kód)
- zaberá zdroje na vývoj, beh, testovanie a udržiavanie softvéru

Charakteristika

- obmedzená životnosť "here now then suddenly vanished"
- single-operation triedy
- limited responsibilites and roles
- opak GodObjectu a anti-patternu Fear of Adding Classes

Ako vznikajú?

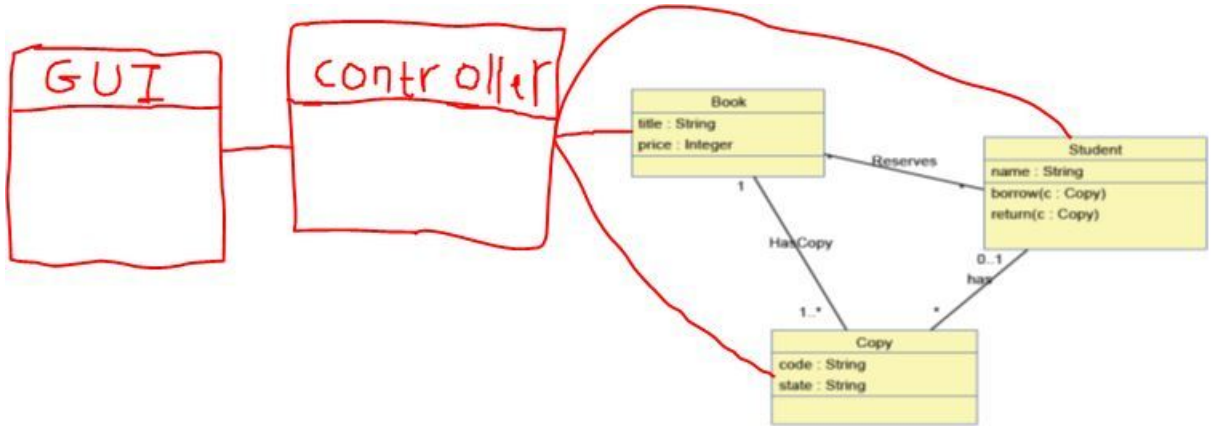
-zlý návrh softvéru

- využitie predlohy pre komplexnejší softvér

-nedostatočné skúsenosti s OOP

Ako ich identifikovať

- ”controller”, “handler”, “manager”,“start_process_whatever”
- čo to vlastne robí?
- zbytočné navigation paths
- single-operation triedy



Veľmi ilustračný príklad

```
class Rectangle:
    def __init__(self, height, width):
        self.height = height
        self.width = width

    def area(self):
        return self.height * self.width
```

vs

```
def area(height, width):
    return height * width
```

Ilustračný príklad

```
// Poltergeist class, with no real value to the code
// as it invokes methods from other classess

public class UserController {
    private UserService service;

    public UserController(UserService service) {
        this.service = service;
    }

    User createUser(User user){
        return service.create(user);
    }
    int findAllUsers(){
        return service.findAll().size();
    }
}
```

```
public class UserService {
    private List users;
    private User user;

    .
    .
    .
    .

    public List findAll(){
        return users;
    }
    public UserService(List users, User user) {
        this.users = users;
        this.user = user;
    }
```

Riešenie

- “ghostbusting”
- refraktorovanie kódu (odstránenie poltergeist triedy)
- nahradenie stratenej funkcionality po odstránení poltergeist triedy

```
public class UserService_NoPoltergeist {
    private List users;
    private User user;

    .
    .
    .
    .
    // Wihtout poltergeist class, we added a new method to
    // UserService  class,
    // that fills the functionality void
    // left after poltergeist class
    public int findAllUsers(){
        return users.size();
    }
    public UserService_NoPoltergeist(List users, User user) {
        this.users = users;
        this.user = user;
    }

}
```

Prevencia

- patrí to do kódu?
- viem to spraviť aj bez toho?
- dôkladná príprava projektu

Otázka č.1

Čo je to godobject(vlastnými slovami):

Otázka č. 2

Pri ktorom z príkladov sa môžu antipatterny prejaviť v kóde:

a) Nevhodne pomenované premenné
b) Bloky nepoužitého kódu
c) Rozdelovanie tried
d) Nedostatočná dokumententácia
e) Pravidelné unit testovanie

Otázka č. 2

Pri ktorom z príkladov sa môžu antipatterny prejaviť v kóde:

a) Nevhodne pomenované premenné
b) Bloky nepoužitého kódu
c) Rozdelovanie tried
d) Nedostatočná dokumententácia
e) Pravidelné unit testovanie