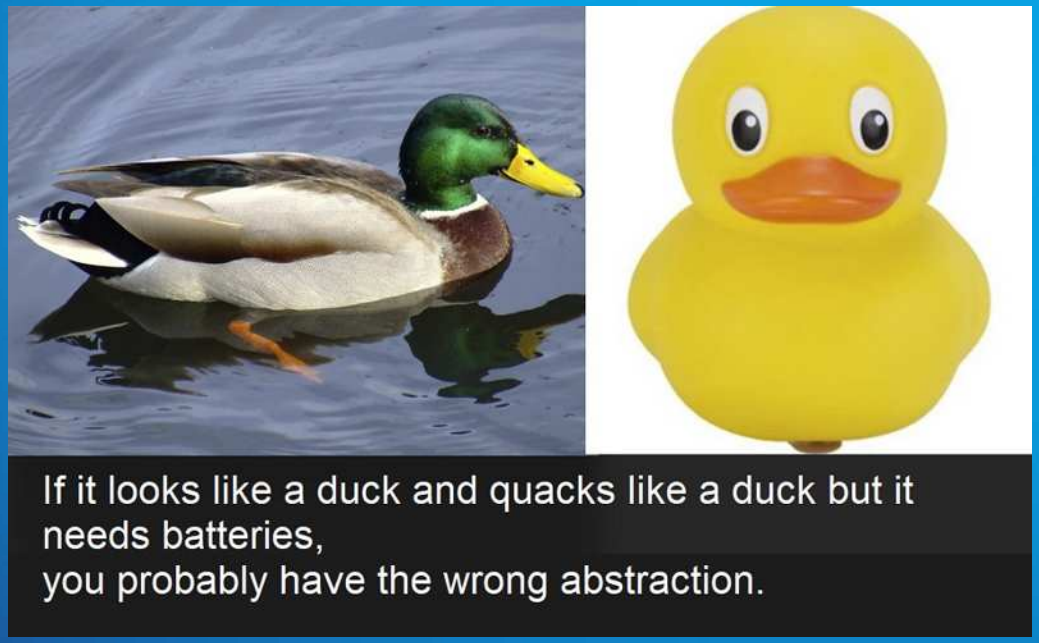
eKogu & the Liskov Substitution Principle

An example of not to do it, and an approach to fix it …



CSS Versicherung

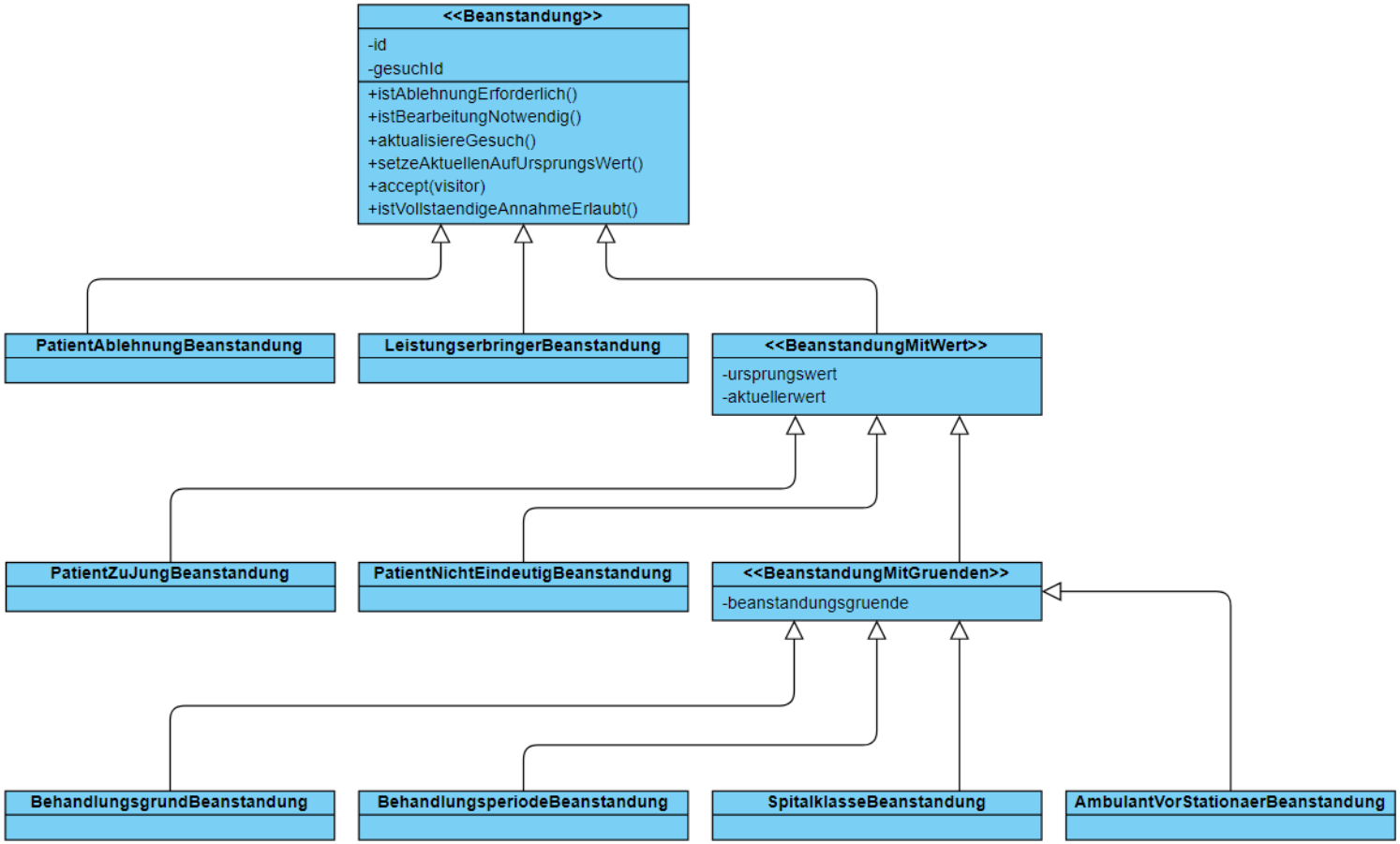
Timetable

1. Implementation
2. Analysis
3. Problem
4. Redesign
5. Result
6. Conclusion

CSS
Versicherung

Implementation I

- We have different complaint types



Implementation II

- `BeanstandungMitWert`

```
public abstract class BeanstandungMitWert<T> extends Beanstandung {

    private final T ursprungswert;
    private T aktuellerwert;

    public BeanstandungMitWert(BeanstandungMitWertBuilder<? extends BeanstandungMitWertBuilder<?, T>, T> builder) {
        super(builder);
        this.ursprungswert = builder.ursprungswert;
        this.aktuellerwert = builder.aktuellerwert;
    }

    @Override
    public final void setzeAktuellenAufUrsprungsWert() {
        aktuellerwert = ursprungswert;
    }
```

Implementation III

- `BeanstandungMitGruenden`

```
public abstract class BeanstandungMitGruenden<T> extends BeanstandungMitWert<T> {

   private final List<Beanstandungsgrund> beanstandungsgruende;

   public BeanstandungMitGruenden(BeanstandungMitGruendenBuilder<? extends BeanstandungMitGruendenBuilder<?, T>, T> builder) {
      super(builder);
      this.beanstandungsgruende = ImmutableList.copyOf(builder.beanstandungsgruende);
   }
```

Implementation IV

- We have two visitor interfaces

```
public interface BeanstandungVisitor {

    void visitLeistungserbringerBeanstandung(LeistungserbringerBeanstandung leistungserbringerBeanstandung);
    void visitPatientAblehnungBeanstandung(PatientAblehnungBeanstandung patientAblehnungBeanstandung);
    void visitPatientZuJungBeanstandung(PatientZuJungBeanstandung patientZuJungBeanstandung);
    void visitPatientNichtEindeutigBeanstandung(PatientNichtEindeutigBeanstandung patientNichtEindeutigBeanstandung);
    void visitSpitalklasseBeanstandung(SpitalklasseBeanstandung spitalklasseBeanstandung);
    void visitBehandlungsgrundBeanstandung(BehandlungsgrundBeanstandung behandlungsgrundBeanstandung);
    void visitBehandlungsperiodeBeanstandung(BehandlungsperiodeBeanstandung behandlungsperiodeBeanstandung);
    void visitAmbulantVorStationaerBeanstandung(AmbulantVorStationaerBeanstandung ambulantVorStationaerBeanstandung);
}
public interface BeanstandungEntityVisitor {

    void visitLeistungserbringerBeanstandung(LeistungserbringerBeanstandungEntity leistungserbringerBeanstandung);
    void visitPatientAblehnungBeanstandung(PatientAblehnungBeanstandungEntity patientAblehnungBeanstandung);
    void visitPatientZuJungBeanstandung(PatientZuJungBeanstandungEntity patientZuJungBeanstandungEntity);
    void visitPatientNichtEindeutigBeanstandung(PatientNichtEindeutigBeanstandungEntity patientNichtEindeutigBeanstandungEntity);
    void visitSpitalklasseBeanstandung(SpitalklasseBeanstandungEntity spitalklasseBeanstandungEntity);
    void visitBehandlungsperiodeBeanstandung(BehandlungsperiodeBeanstandungEntity entity);
    void visitBehandlungsgrundBeanstandung(BehandlungsgrundBeanstandungEntity behandlungsgrundBeanstandungEntity);
    void visitAmbulantVorStationaerBeanstandung(AmbulantVorStationaerBeanstandungEntity ambulantVorStationaerBeanstandungEntity);
}
```

Analysis I

- All needed operations placed on top

```
@NotNull
private final BeanstandungId id;

@NotNull
private final GesuchId gesuchId;

protected Beanstandung(BeanstandungBuilder<? extends BeanstandungBuilder<?>> builder) {
    this.id = builder.id;
    this.gesuchId = builder.gesuchId;
}

public abstract boolean istAblehnungErforderlich();

public abstract boolean istBearbeitungNotwendig();

public abstract <T extends BeanstandungVisitor> T accept(T visitor);

public abstract void aktualisiereGesuch(Gesuch gesuch);

public abstract void setzeAktuellenAufUrsprungsWert();

public boolean istVollstaendigeAnnahmeErlaubt() {
    return false;
}
```

Analysis II

- `istAblehnungErforderlich()` is always false except for `PatientAblehnungBeanstandung` & `LeistungserbringerBeanstandung`
- `istBearbeitungNotwendig()` is always false except for `PatientNichtEindeutigBeanstandung` & `PatientZuJungBeanstandung`
- `istVollstaendigeAnnahmeErlaubt()` is always false except for `PatientNichtEindeutigBeanstandung` & `PatientZuJungBeanstandung`

Analysis III

- `PatientAblehnungBeanstandung` & `LeistungserbringerBeanstandung` throw an Exception whenever `aktualisiereGesuch()` or `setzeAktuellenAufUrsprungsWert()` is called

- `PatientAblehnungBeanstandung` & `LeistungserbringerBeanstandung` each have a method `getAblehnungsgrund()` which is not in the super class

- `AmbulantVorStationaer` has no values, it represents a state whether or not the planned stationary therapy should be ambulant

Analysis IV

- `BeanstandungMitWert` adds only two fields, but no additional logic

- `BeanstandungMitGruenden` adds only a list, but no additional logic

Analysis V

- One database table to store all complaints
- One DTO known to the UI
- But: eight different types of complaints!

- Trying to avoid instanceof by using the visitor pattern
  - Is there a better way to do it?

Problem

- Trying to unite all complaints under one roof
- But we have different types:
  - Invalid patient and/or care provider
  - Patient identification
  - Wrong information within the request
- Not all complaints need and support everything
  - LSP violation
  - Is it an Open/Close Principle violation as well?

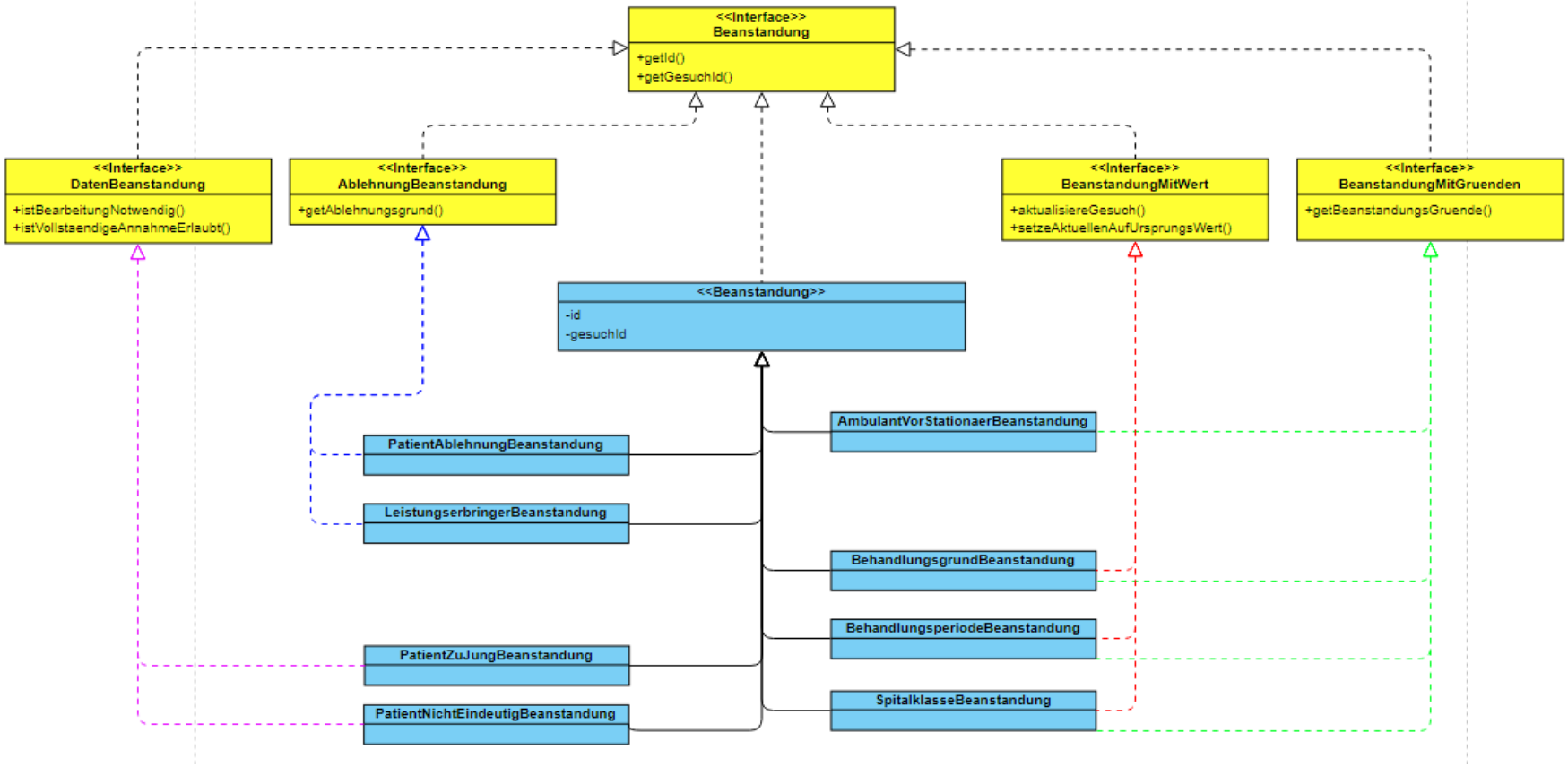
Redesign

Find the «right» abstraction…

- Introduce interfaces to build groups
- Flatten hierarchy
- Inherit only what you need and support

Result

- It looks promising, but it’s only the beginning…



Conclusion

- Solution is not «finalized» since it’s only theoretical
- Getting rid of unnecessary extensions
- Getting rid of the visitor pattern by having interfaces
- Helper classes should not be in the hierarchy

Thank you!

- References:
  - «Agile Technical Practices Distilled» by Pedro Moreira Santos, Marco Consolaro & Alessandro Di Gioia