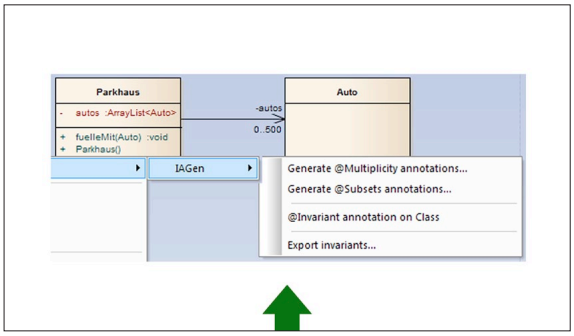
Frederick Martin Egli

Dominik Mengelt

| | |
|---|---|
| Diplomanden | Frederick Martin Egli, Dominik Mengelt |
| Examinator | Thomas Letsch |
| Experte | Dr. Martin Zimmermann, FH Offenburg DE |
| Themengebiet | Software |

# IAGen – Invariants Assertion Generation

## Automatische Zusicherung von Invarianten im Enterprise Architect sowie im Java Source Code




Erkennung der Multiplizitätsinvariante durch das IAGen Add-In

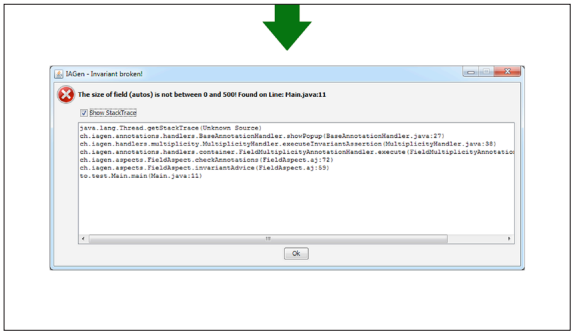
```java
@Invariant
public class Parkhaus {

    @Multiplicity(upper=500, showPopup=true)
    private ArrayList<Auto> autos;

    public Parkhaus() {
        autos = new ArrayList<Auto>();
    }

    public void fuelleMit(Auto a){
        autos.add(a);
    }
}
```

Invariantendefinition als Java Annotation nach der Codegenerierung




Pop-up-Benachrichtigung, da Invariante zur Laufzeit verletzt wurde

Ausgangslage: Modelliert ein Entwickler mit dem CASE Tool Enterprise Architect, können auch Invarianten definiert werden. Beispielsweise kann eine Multiplizität, welche aussagt, wie viele Objekte in einer Beziehung stehen können, auf einer Assoziation angegeben werden. Durch die Verwendung von Code-Synchronisierung (Forward Engineering) im Enterprise Architect gehen diese Informationen allerdings verloren und eine Invariantenüberprüfung des Programmes zur Laufzeit findet nicht statt. Ein Java-Entwickler kann Invariantenüberprüfungen zwar programmatisch vornehmen, dabei gibt es aber negative Effekte:

- Die Logik der Invariantenüberprüfung vergrössert den Code
- Ändert eine Invariantendefinition, kann dies zu erheblichem Aufwand führen
- Die Invariantenüberprüfung kann teilweise umgangen werden (öffentliche Felder)

Vorgehen/Technologien: Das Vorgehensmodell während des Projekts basiert auf dem Unified Process nach Craig Larman. Durch die Erstellung von Anforderungsspezifikationen wussten wir, was der Kunde will und wo die Rahmenbedingungen des Projektes liegen. Es wurde früh mit der Entwicklung eines Prototyps begonnen, da im Vorhinein nicht bekannt war, ob das Projekt im gewünschten Stil realisierbar ist. Dank der Dokumentation von Architektur und Design hatte man eine gute Diskussionsgrundlage und war immer im Bild, wie IAGen aufgebaut wurde. Aufgrund der Aufgabenstellung, der Anforderungsspezifikationen und des Prototyps wurden folgende Technologien vorgegeben bzw. evaluiert:

- Java und Java-Annotationen
- AspectJ
- Enterprise Architect
- C#

Ergebnis: IAGen hilft bei Code-Synchronisierung aus und in den Enterprise Architect, damit keine Invariantendefinitionen verloren gehen. Ausserdem ermöglicht IAGen, gesetzte Invarianten während der Laufzeit einer Java-Applikation zu überprüfen und bei allfälligen Invariantverletzungen entsprechend zu reagieren. Die Benutzung von IAGen ist dabei intuitiv und kann schnell erlernt werden. Unter anderem gibt es folgende Teilergebnisse:

- Java-Source-Code-Annotationen: @Min, @Max, @Range, @Multiplicity, @Subsets, @Notnull und @Const
- Mögliche Reaktionen bei Invariantverletzungen: Exception, Pop-up, Logging
- Annotation Processing von IAGen-Annotationen in der Entwicklungsumgebung
- Behandlung von modellierten IAGen-Invarianten mit einem Enterprise Architect Add-In
- Nachträgliche Annotierung von Java Byte Code