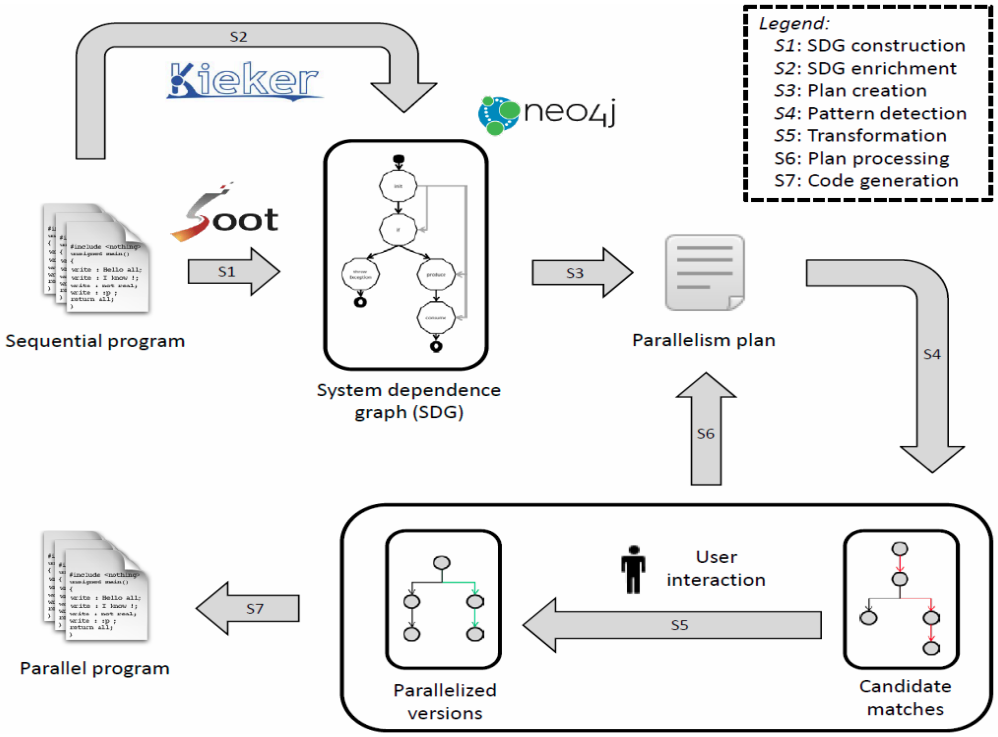
Automatic Refactoring with the Null Object Pattern using PARROT

Tim-Niklas Reck

28.09.17

Motivation

Kontext



Der halbautomatische Ansatz PARROT
(In Anlehung an Wulf [2014])

Motivation

Problem

Listing 3.4. Ausschnitt aus Checkstyles Checker.java.

```
1   public class Checker {
2   private PropertyCacheFile cache;
3
4   private void processFiles(List<File> files) throws CheckstyleException {
5       for (final File file : files) {
6           try {
7               final String fileName = file.getAbsolutePath();
8               final long timestamp = file.lastModified();
9               if (cache != null && cache.isInCache(fileName, timestamp)
10                      || !CommonUtils.matchesFileExtension(file, fileExtensions)
11                      || !acceptFileStarted(fileName)) {
12                  continue;
13              }
14              if (cache != null) {
15                  cache.put(fileName, timestamp);
16              }
17              fireFileStarted(fileName);
18              final SortedSet<LocalizedMessage> fileMessages = processFile(file);
19              fireErrors(fileName, fileMessages);
20              fireFileFinished(fileName);
21          }
22          // -@cs[IllegalCatch] There is no other way to deliver filename that was under
23          // processing. See https://github.com/checkstyle/checkstyle/issues/2285
24          catch (Exception ex) {
25              // We need to catch all exceptions to put a reason failure (file name) in
                    exception
26              throw new CheckstyleException("Exception was thrown while processing "
27                      + file.getPath(), ex);
28          }
29      }
30  }
31  }
```

Motivation

Idee

Listing 3.2. Klasse mit nicht initialisiertem Feld.

```
1  public class RandomClass {
2      private Cache cache;
3
4      public RandomClass(){
5      }
6
7      public void createCache(){
8          this.cache = new Cache();
9      }
10
11     public void cacheFile(File filename)
           {
12         if (cache != null) {
13             cache.put(filename);
14         }
15     }
16
17     public void clearCache(){
18             if (cache != null) {
19                 cache.reset();
20         }
21     }
22 }
```

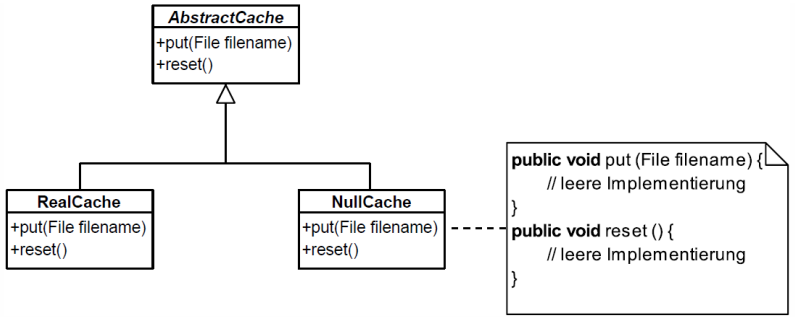


Abbildung 3.3. Umsetzung der Klasse *Cache* aus Listing 3.2 nach dem Nullobjekt-Muster.

Gliederung

- Motivation
- Ziele
- Grundlagen und Technologien
- Ansatz
  - Erkennung
  - Transformation
- Evaluation
- Fazit

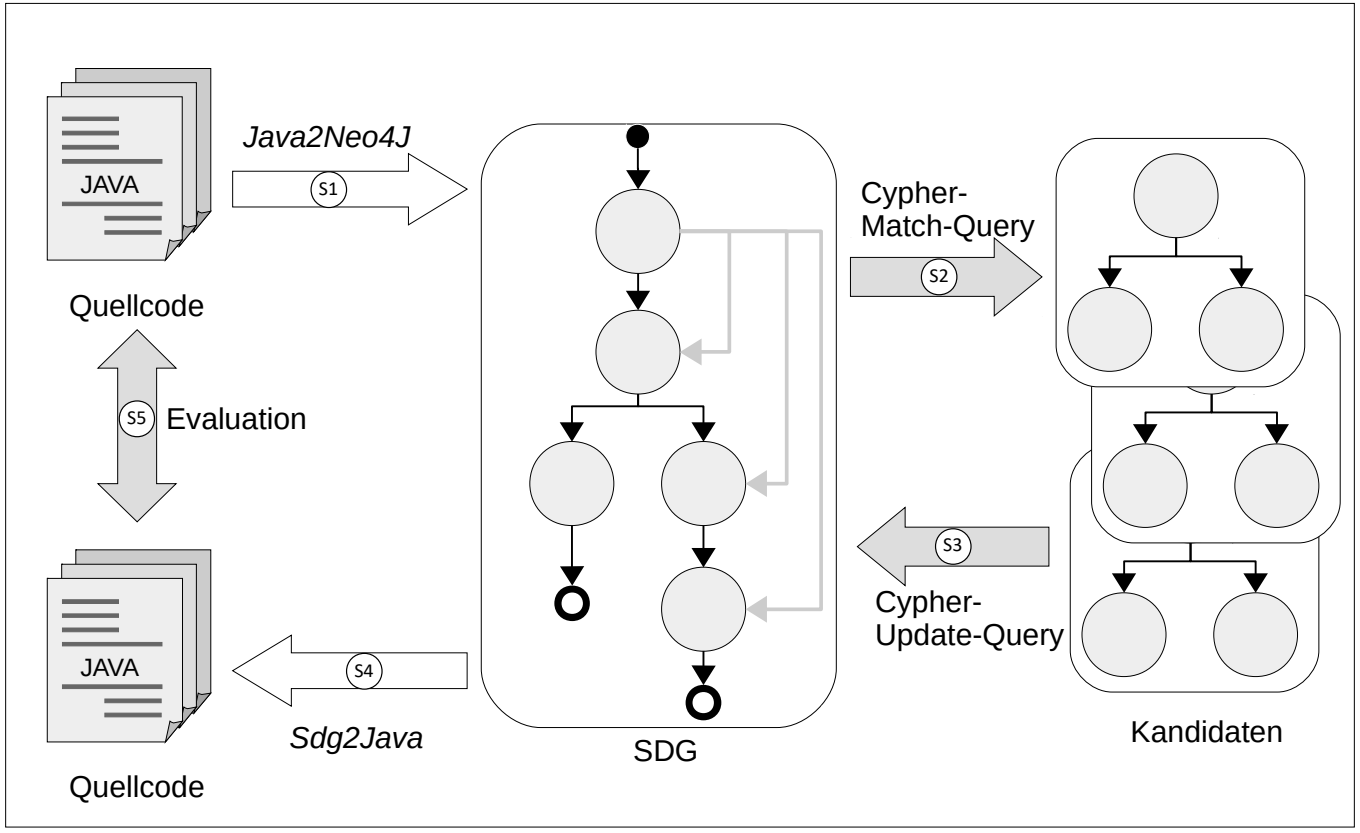
Ziele

- Z1: Erkennung von Nullobjekt-Kandidaten
- Z2: Transformation von Nullobjekt-Kandidaten
- Z3: Evaluation der Ergebnisse

Grundlagen und Technologien

- Graphentheorie
  - Allgemeine Graphen
  - JsysDG
- Die Graphdatenbank Neo4J
- Das Nullobjekt-Muster
- Die Tools Java2Neo4J und Sdg2Java

Ansatz


JAVA
Quellcode
Java2Neo4J
S1
SDG
Cypher-
Match-Query
S2
Kandidaten
S3
Cypher-
Update-Query
S5
Evaluation
JAVA
Quellcode
S4
Sdg2Java

Erkennung

Listing 4.3. Kandidatenklasse Cache.java

```
1  public class Cache {
2
3      public void put(String filename, int
           size) {
4          size();
5      }
6
7      public void reset() {
8          size();
9      }
10
11     private int size(){
12         return 0;
13     }
14 }
```

Listing 1. Die Aufruferklasse MainClass.java

```
1  public class MainClass {
2
3      private Cache cache;
4      ...
5
6      public MainClass(){
7          subtitle = new Text();
8      }
9
10     public void createCache(){
11         if (cache == null) {
12             this.cache = new Cache();
13         }
14     }
15
16     public void clearCache(){
17         if (cache != null) {
18             cache.reset();
19         }
20     }
21
22     ...
23 }
```

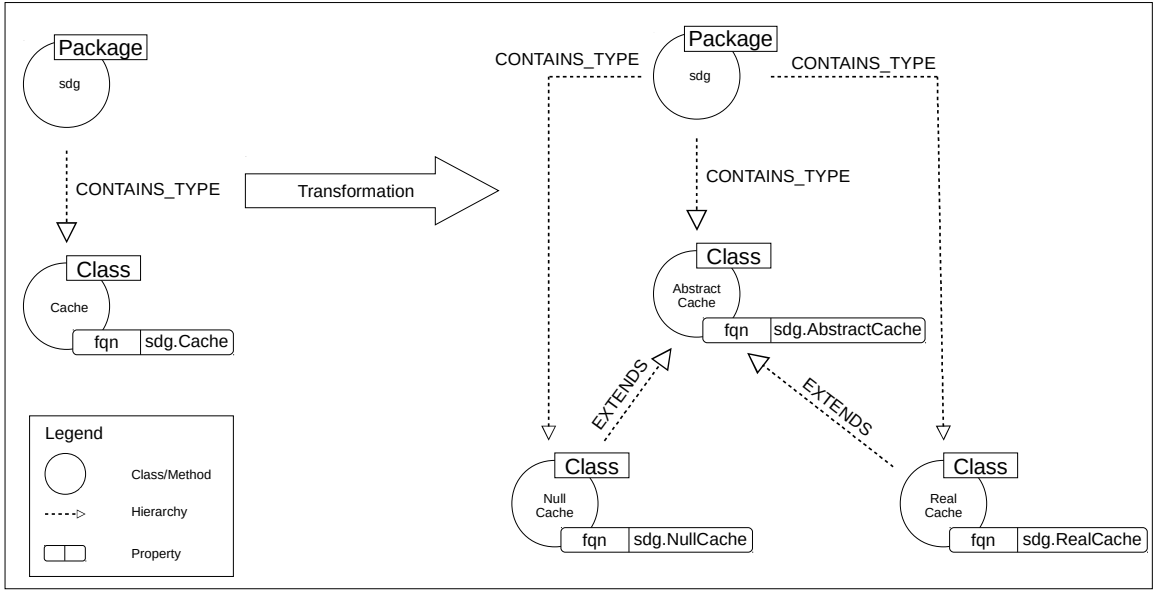
Erkennung

Listing 3.2. MATCH-Abfrage

```
1 MATCH (mainClass:Class)-[:CONTAINS_FIELD]->(candidateField:Field {isfinal:false})<-[:
      AGGREGATED_FIELD_READ]-(method:Method)
2    USING INDEX candidateField:Field(isfinal)
3 MATCH (candidateField)-[:DATA_FLOW]->(condVariable:Assignment)-[:DATA_FLOW]->(condition:
      Condition {operation:"!="})
4    WHERE condition.operand1 = "null" OR condition.operand2 = "null"
5 MATCH (condVariable)<-[:CONTROL_FLOW]-(ifStmt:NopStmt)
6    WHERE ifStmt.nopkind = "IF_COND" OR ifStmt.nopkind = "IF_COND_X" OR (ifStmt) <-[:
         CONTROL_FLOW]- (:Condition)
7 MATCH (candidate:Class)
8    WHERE candidate.fqn = candidateField.vartype AND candidate.isabstract = false
9 RETURN DISTINCT candidateField, condVariable, candidate
```

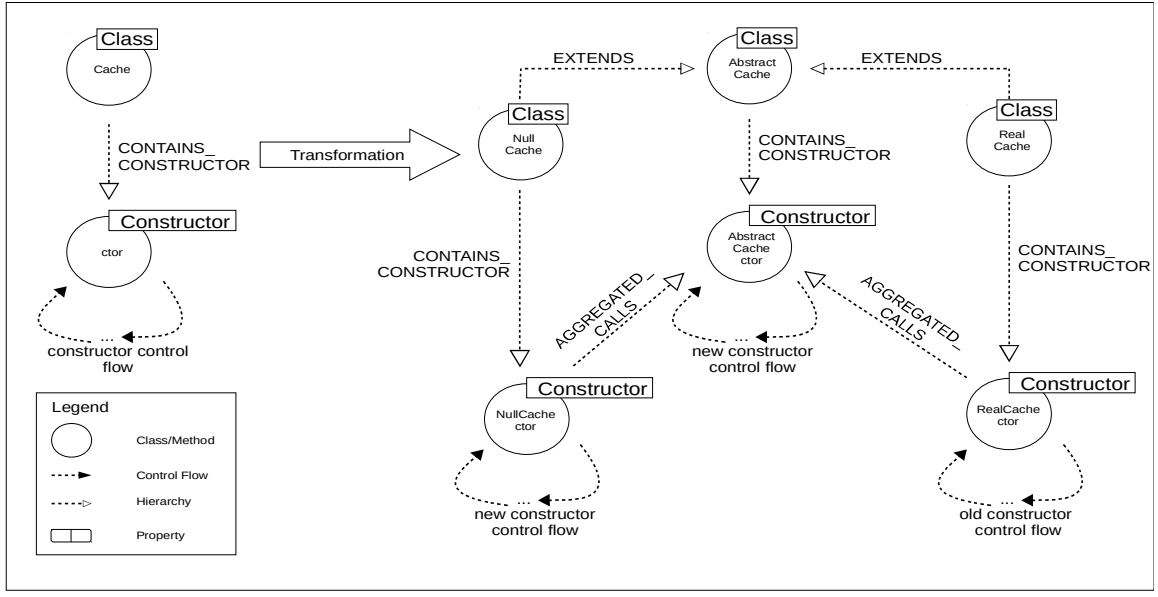
Transformation der Kandidaten

- Erstellung neuer Klassen



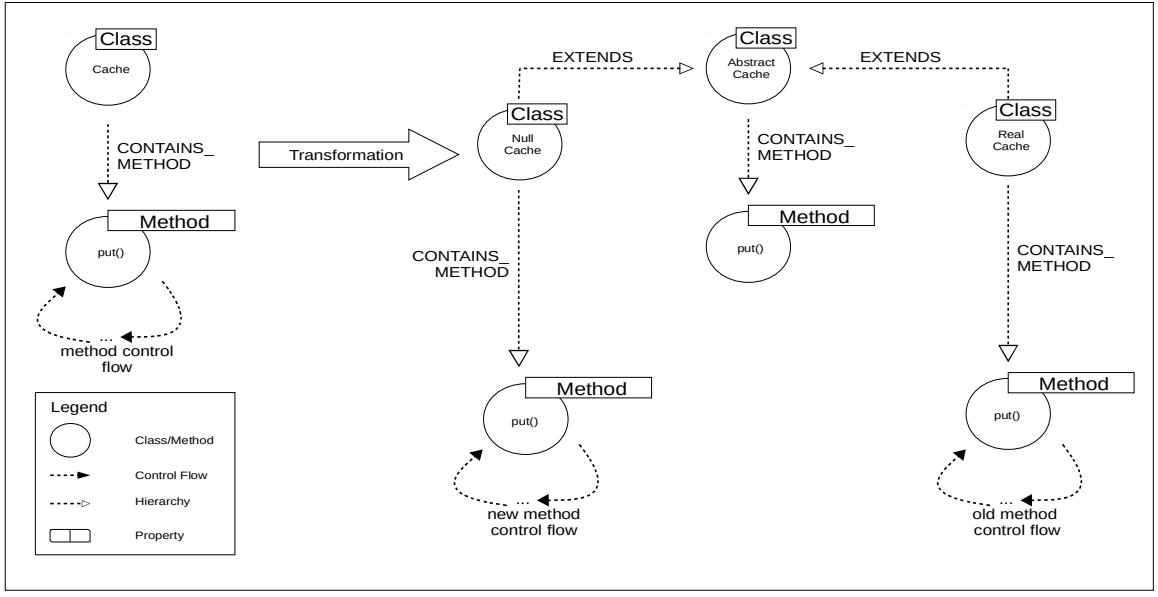
Transformation der Kandidaten

- Erstellung neuer Klassen
- Anpassung der Konstruktoren



Transformation der Kandidaten

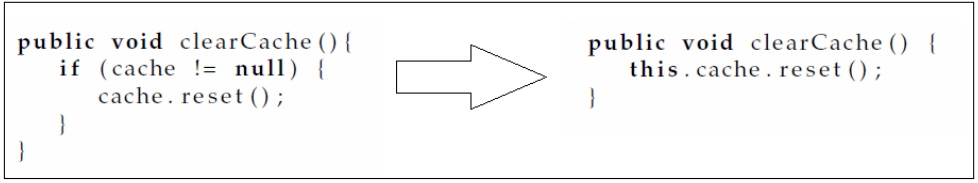
- Erstellung neuer Klassen
- Anpassung der Konstruktoren
- Anpassung der Methoden



Ansatz

Transformation der Aufruferklassen

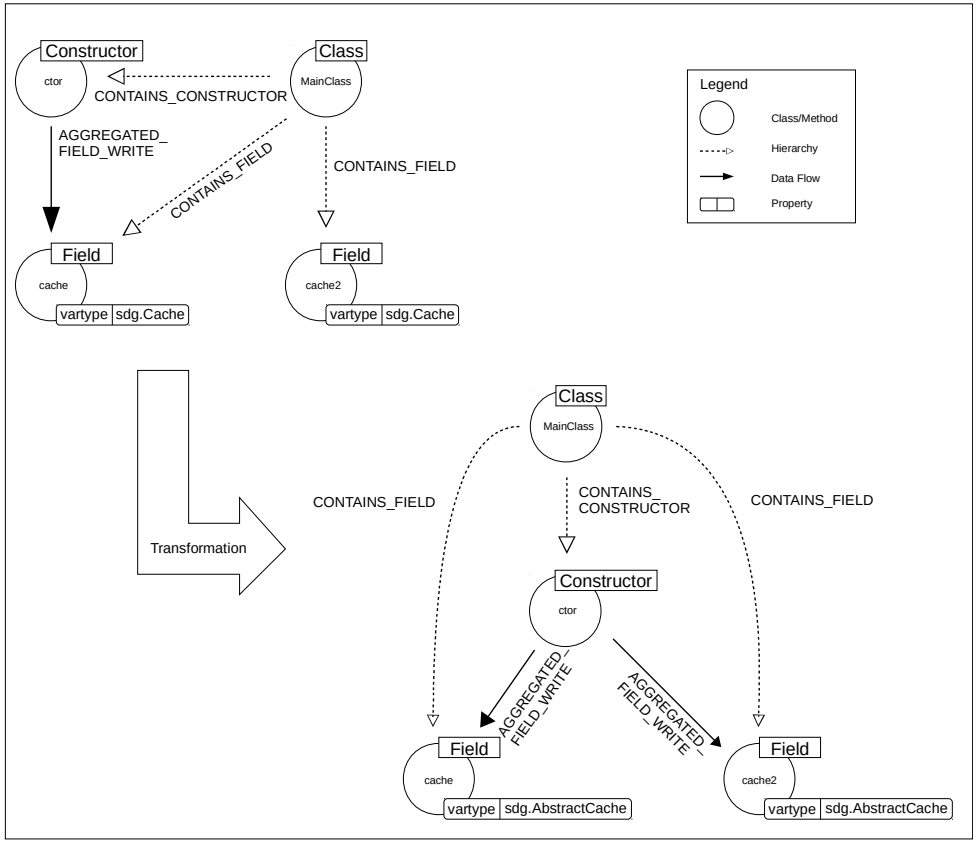
- Anpassen der if-Anweisungen



Ansatz

Transformation der Aufruferklassen

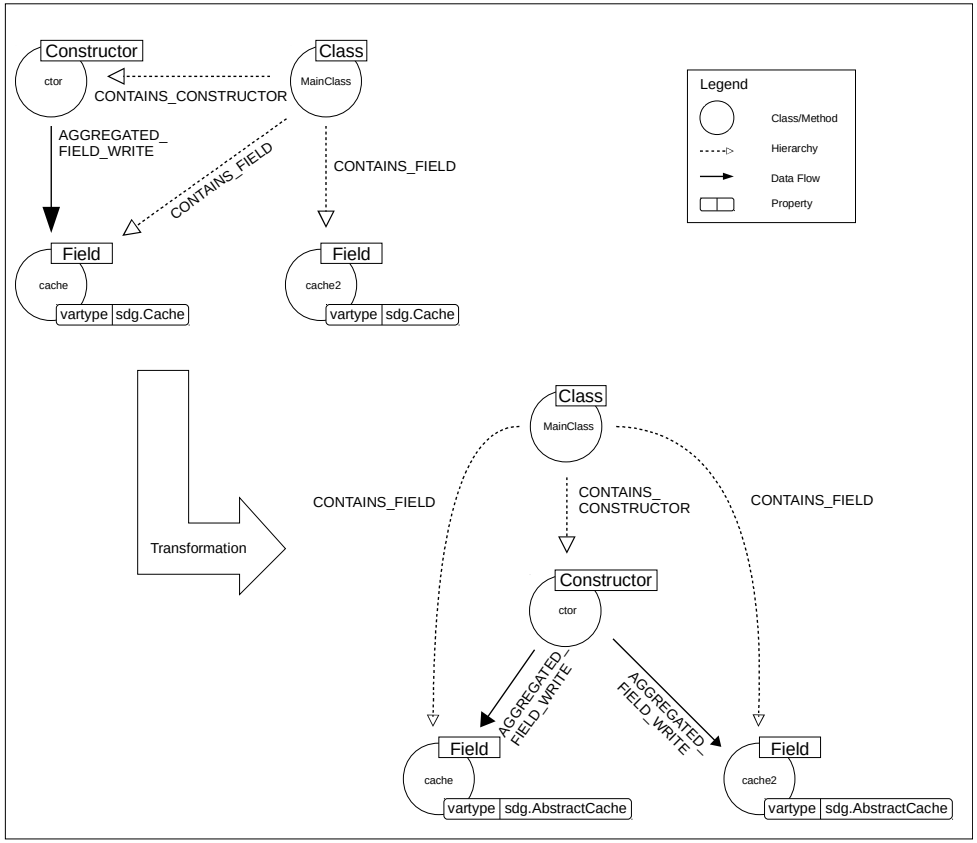
- Anpassen der if-Anweisungen
- Anpassung der Felder an neue Typen



Ansatz

Transformation der Aufruferklassen

- Anpassen der if-Anweisungen
- Anpassung der Felder an neue Typen
- Initialisierung bisher nicht initialisierter Felder



Evaluation

Erkennung

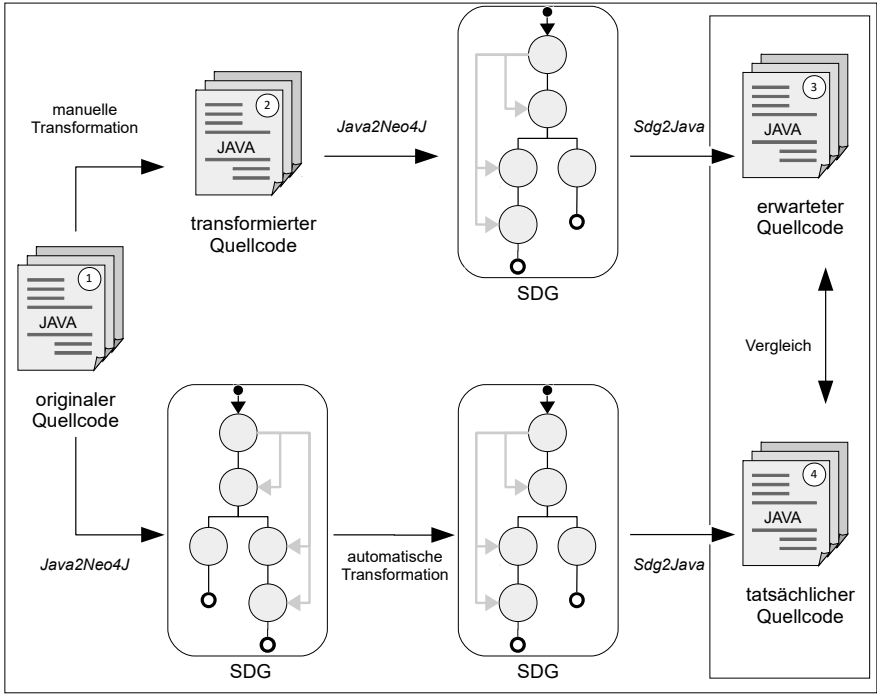
- Überprüfung mit:
  - eigenen Klassen
  - Klassen in Apache Ant
- Vergleich der textuellen Suche nach Kandidaten in Ant und der Abfrage

Evaluation anhand unseres eigenen Quellcodes

Klasse	Feldname	Übereinstimmung erwartet	Korrektes Matching	
			des Feldes	der Methoden
Cache	cache	ja	✓	✓
	cache2	nein	✓	-
Text	text	ja	✓	✓
	title	nein	✓	-
	subtitle	nein	✓	-
AbstractObject	abstractObject	nein	✓	-

Evaluation

Transformation



Evaluation

Transformation

Die erwartete Klasse RealCache

```
1  package de.tnr.sdg.example.
       transformedCache;
2
3  public class RealCache extends de.tnr.sdg.
       example.transformedCache.
       AbstractCache {
4
5      public RealCache() {
6          super();
7      }
8
9      private final int size() {
10         return 0;
11     }
12
13     public void reset() {}
14
15     public void put(java.lang.String
           filename, int size) {}
16 }
```

Die tatsächliche Klasse RealCache

```
1  package de.tnr.sdg.example.cache;
2
3  public class RealCache extends de.tnr.sdg.
       example.cache.AbstractCache {
4
5      private final int size() {
6          return 0;
7      }
8
9      public RealCache() {
10         super();
11     }
12
13     public void put(java.lang.String
           filename, int size) {}
14
15     public void reset() {}
16 }
```

Fazit

- Cypher-Abfragen sind
  - effizient bei der Identifizierung von Mustern
  - sehr unübersichtlich bei größeren Transformationen
- Transformation und Erkennung sind korrekt durchführbar
- Future Work:
  - Transformation abstrakter Klassen
  - Java-Klassen als Kandidaten
  - Abfragen, ob ein Feld null ist, überdenken.