HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Musterlösungen in Graja als Mechanismus für Regressionstests

Kolloquium zur Bachelorarbeit

Marc Herschel, 16.09.2020

Inhaltsverzeichnis

Problem- und Zielstellung der Arbeit

http://graja.hs-hannover.de

- Autobewerter ***Graja*** stellt ein komplexes Softwaresystem dar
- Zustand vor der Arbeit: **~60k** Zeilen Java-Quellcode (Metrik: *SLOC*) und **31** Gradle-Untermodule
- Modultests mittels JUnit 4 für nur **3** dieser **31** Module
- Integrationstests bezüglich der Untermodule: **keine**

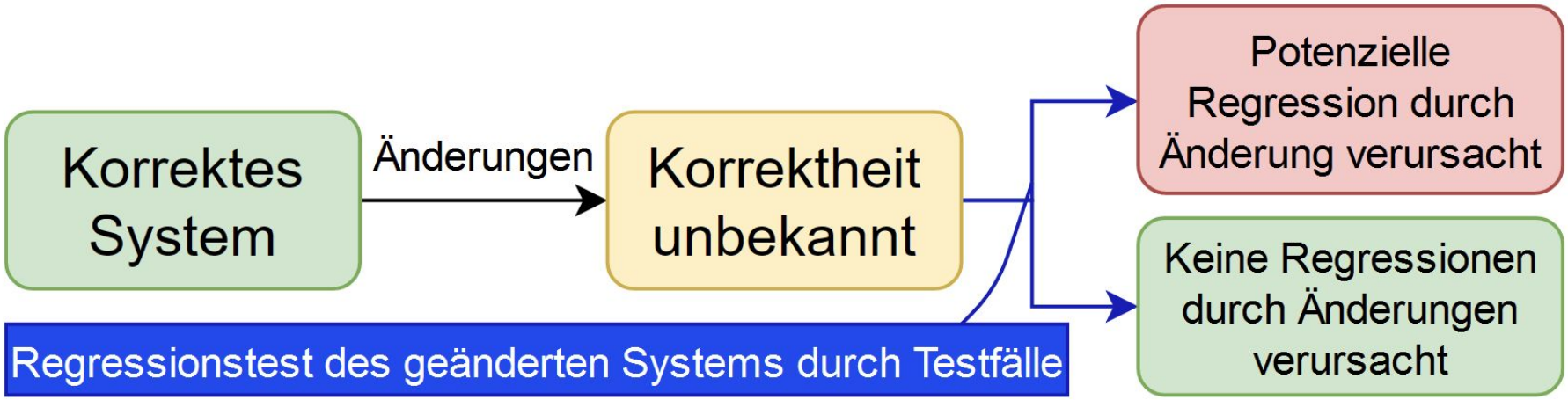
Problem: *Wie kann eine fehlerfreie Weiterentwicklung Grajas durch Softwaretests, in diesem Falle Regressionstests, garantiert werden?*

Ziel: *Konzept für einen Regressionstestmechanismus unter der Verwendung von Graja-Musterlösungen entwickeln.*

Regressionstests

„Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.“

IEEE Standard Glossary of Software Engineering Terminology, S. 61



- Regression stellt hier einen solchen unerwünschten Nebeneffekt dar
- Falls keine Regressionen verursacht → Übernehme Änderung in System

Regressionstests im Kontext von Graja

Im Falle von Graja: Das Bewertungsergebnis einer beliebigen Graja-Aufgabe sollte nach Änderungen innerhalb eines Graja-Untermoduls, bei gleichbleibenden Aufgaben-Verhalten reproduziert werden.

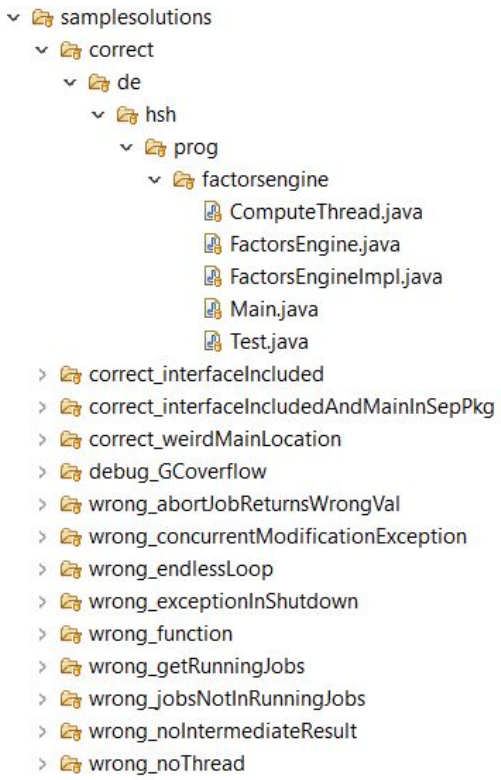
Ausnahme: Änderung soll Testfall explizit durch vorher unbekanntes Verhalten *„brechen“* → neuer Testfall, der Verhalten der Änderung abbildet notwendig.

Problem: *Woher für Graja-Regressionstests die Testfälle nehmen?*

Die bereits bestehenden Aufgaben in Kombination mit den bereits bestehenden Musterlösungen als Testfälle verwenden!

Musterlösungen in Graja

Verpflichtender Teil des durch Graja unterstützten ProFormA-Aufgaben-formats (*https://github.com/ProFormA/proformaxml*)



Verwendungszweck: z.B. manuelles Testen des *Grader-Codes* einer Aufgabe durch *Graja-GUI*.

Musterlösungen simulieren studentische Einreichungen und werden in Verbindung mit der jeweiligen Aufgabe durch Graja bewertet.

Können sowohl positiv als auch negativ ausfallen, als auch *„verbotene“* Operationen, die durch Graja abgefangen werden durchführen.

Für **57** Aufgaben existieren bereits **437** Musterlösungen!

Daher guter Kandidat für Testfälle! Große Vielfalt an Aufgaben und Musterlösungen können direkt übernommen werden und decken den gesamten Einsatzbereich ab!

Regressionstests durch Verhaltensabgleich

Realisiere Regressionstests durch Verhaltensaufzeichnung und Verhaltensabgleich des Bewertungsverhaltens einer Musterlösung:

Ist-Verhalten:

- Durch die geänderte Graja-Version aufgezeichnetes Bewertungsverhalten.
- *Sonderfall*: Existiert kein Soll-Verhalten wird die Erstaufzeichnung des Ist-Verhaltens zu neuem Soll-Verhalten hochgestuft.

Soll-Verhalten:

- Neben den Musterlösungen im Quellcode der Aufgabe persistiert.
- Durch Ist-Verhalten einer geänderten Version zu reproduzierendes Verhalten.
- Differenzen dieses Abgleichs gelten als potenzielle Regressionen.
- *Sonderfall*: Neuaufzeichnung möglich, falls Änderungen zu potenziellen Regressionen führen, diese aber explizit erwünscht sind (z.B. neues Feature).

Frage: *Welche Graja-Daten eignen sich als Bewertungsverhalten?*

Eignung der durch Graja generierten Daten

Generierte Feedback-Dokumente:

- In beiden Fällen (*Plain text* / HTML) ungeeignet, da zu unstrukturiert.
- Bei HTML müsste auf *Screen Scraping* gesetzt werden
- → fragile, gegenüber Änderungen in der Struktur der Feedback-Dokumente vulnerable Methode der Datenextraktion.

```
<a id='anchor_7b32...'></a>
<span style='...'>
Correct. Should use indentation consistently
</span>
<span style='...'>. Score: 0.15&#47;0.15</span><br>
<div>
Checks correct indentation of Java code. The expected basic
    indentation offset is 4 spaces.
</div><p>
<span>
Check Indentation (aspect ID: @@!!ProFormA-GradingHints:
   testRef=checkstyle:ProFormA-GradingHints:subRef=
   Indentation:ProFormA-GradingHints:cnt=45!!@@ ) is clean.
</span></p>
```

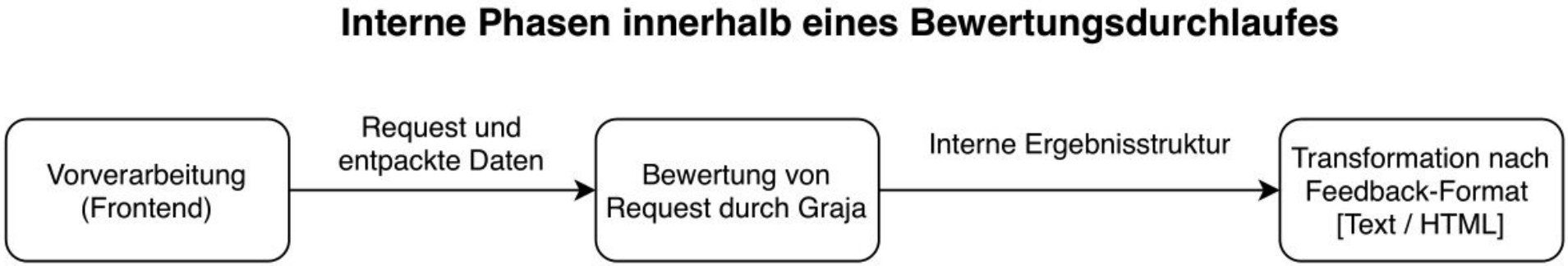
Extrahieren von Punkten?

Identifikation vom generierenden Graja-Modul?

Kommentare immer in ?

Interesse bezüglich der Regressionstests auch eher am korrekten Bewertungsverhalten, nicht dem abschließenden Feedback-Dokument!

Eignung der durch Graja generierten Daten

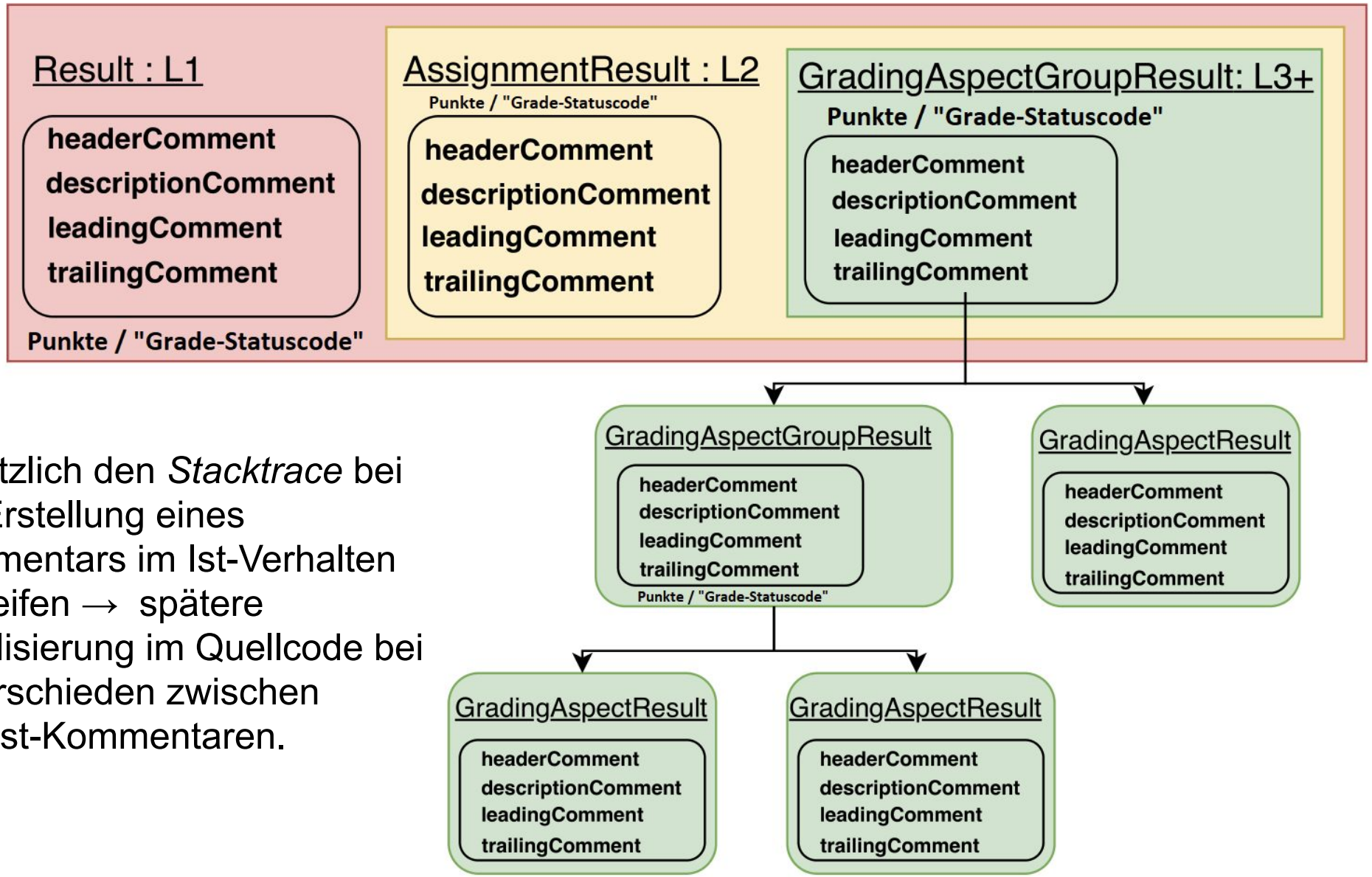


Interne Graja-Ergebnis-Datenstruktur:

- Guter Kandidat, da kein Informationsverlust durch Umwandlung.
- Zugriff auf ausgefülltes, baumartiges Bewertungsschema (Bewertungsaspekte und Aspektgruppen + Punkte/Kommentare).
- Zugriff auf die baumartige Zwischenpräsentation der Graja-Kommentar-Bibliothek und verschiedene Kommentar-Typen (Text, Bild, Diff, Inline HTML, Listen, …).

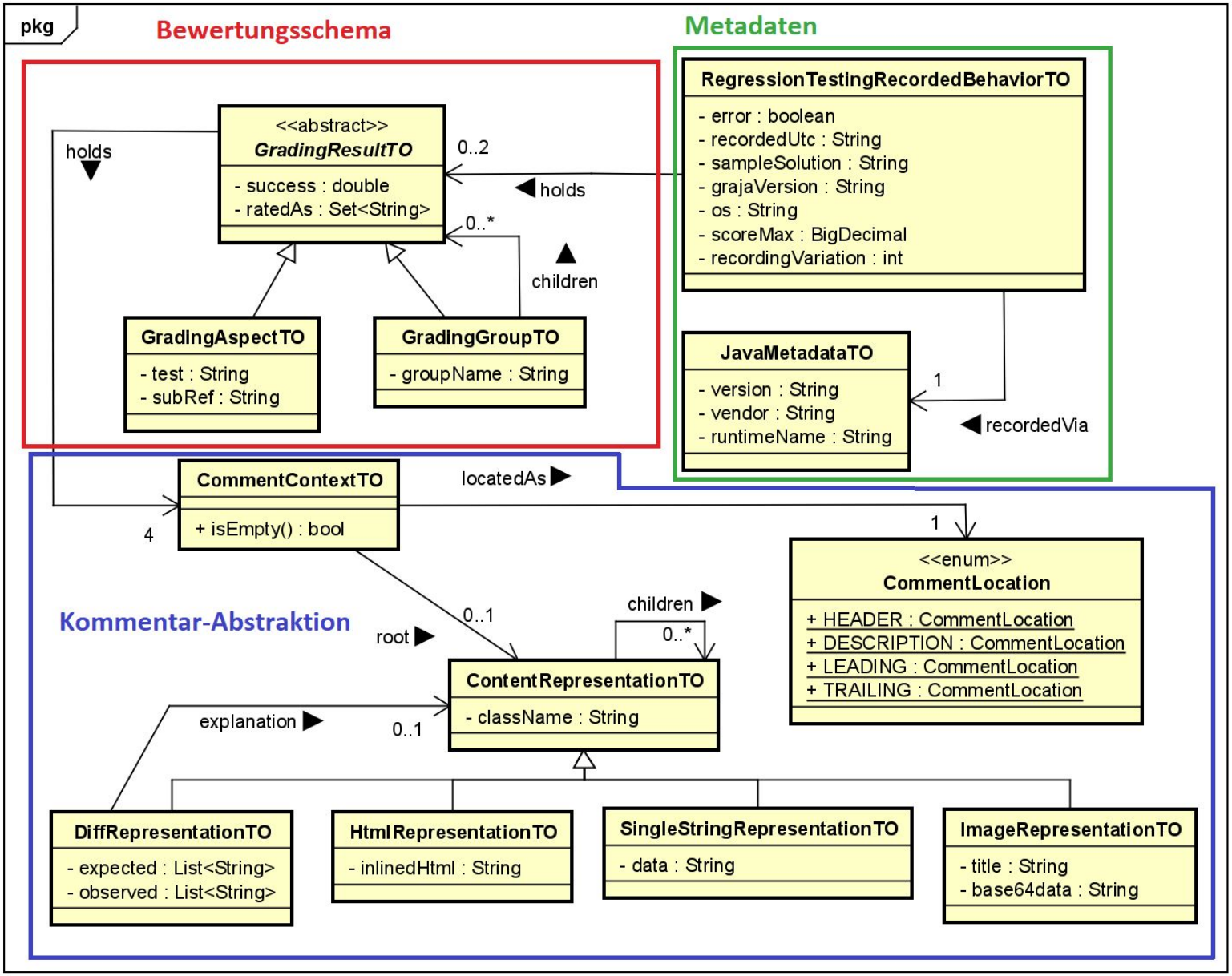
Zu ausführlich für den Vortrag, in der Arbeit unter Kapitel 5.2.2 und 5.2.3 behandelt.

Interne Graja-Ergebnis-Datenstruktur



Zusätzlich den *Stacktrace* bei der Erstellung eines Kommentars im Ist-Verhalten abgreifen → spätere Lokalisierung im Quellcode bei Unterschieden zwischen Soll-Ist-Kommentaren.

Datenmodell für aufgezeichnetes Verhalten



Behandlung von Fehlalarmen und Rauschen

Rauschen in Kommentaren:

- Identifiziert durch Datenanalyse → z.B. durch Dateipfade, Zeitstempel, ...
- Neutralisierung durch Ersetzung mithilfe regulärer Ausdrücke

```
PATH_WINDOWS_SLASH_REVERSED
    // Raw: ([A-Z]{1}:\/)([^:\*\?\"\>\<\|]+\/)(([^\'\]\,:\;\/\*\?\"\>\<\|\\\
    ("([A-Z]{1}:\\/)([^:\\*\\?\\\"\\>\\<\\|]+\\/)(([^\\'\\]\\,:\\;\\/\\*\\?\
            ReplaceAlias.PATH),
DATE_SIMPLETIMEFORMAT_DEFAULT
    // Raw: \d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(.(\d{3}|\d{2}|\d{1}))*
    ("\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}(.(\\d{3}|\\d{2}|\\d{1}))*",
            ReplaceAlias.TIMESTAMP),
TIME_VALUE_IN_SECONDS
    // Raw: [0-9]+s
    ("[0-9]+s",
            ReplaceAlias.TIMESTAMP),
```

Problem: Manchmal nicht ausreichend bei z.B. platformabhänigen Trennzeichen oder Tippfehlern.

```
<< SOURCE
Compiler options: [-encoding, UTF-8, -classpath, _PATH_:_PATH_]
>> SOURCE

<< TARGET
Compiler options: [-encoding, UTF-8, -classpath, _PATH_;_PATH_]
>> TARGET
```

Daher zusätzlich: *„fuzzy string matching“* mithilfe der *Levenshtein-Distanz*.

Levenshtein-Distanz: Ermittlung der Einfüge/Ersetzungs/Löschungs-Operationen zwischen zwei Zeichenketten, um die erste in die zweite Umzuwandeln.

„fuzzy string matching“ durch Schwellwertoperation realisieren → z.B. 3 als globaler Schwellwert in der Referenzimplementierung gewählt (Grund: Trennzeichen / *„Typos“*).

Behandlung von Fehlalarmen und Rauschen

Unvorhersehbare Abarbeitungsreihenfolgen:

- Auflistung der Regeln des PMD-Moduls in nichtdeterministischer Reihenfolge → lexikografisch vor Ausgabe sortieren
- Systemabhängige, unterschiedliche Bearbeitung der eingereichten Dateien durch das Checkstyle-Modul → lexikografisch nach Dateiname sortieren

Musterlösungen mit mehreren validen Ausführungssträngen:

- Einige Musterlösungen weisen mehrere korrekte Bewertungsergebnisse auf.
- z.B. bei Aufgaben, die Parallelisierung beinhalten (*factorsengine: wrong_concurrentModificationException*).
- daher *„multi sample mode“* um mehrere Ausführungsstränge aufzuzeichnen.

Behandlung von Fehlalarmen und Rauschen

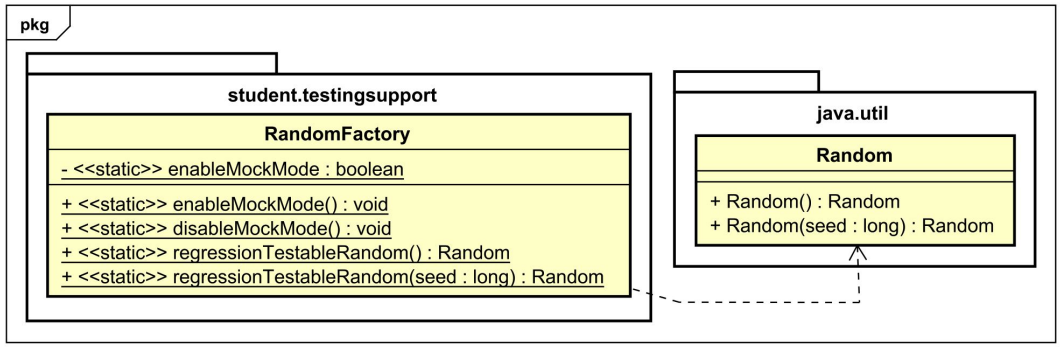
Verwendung des Pseudozufallsgenerators der Klasse `Random`:

- Einige Aufgaben verwenden Pseudozufallsgenerator für Testwerte

Problem: `Random` bezieht initialen *„seed“* standardmäßig aus der Systemzeit.

→ bei ungleicher gleicher Systemzeit können unterschiedliche Ausgaben in den durch Graja generierten Kommentaren stehen (z.B. zufallsgenerierte Zeichenketten, Zahlen, …)

Lösung: Über *„Factory“* `Random`-Objekte mit gleichbleibenden Seed für Tests generieren.



→ Nutzung von `Random`-Objekten in der stud. Einreichung nicht berücksichtigt.

→ Anpassung des Quellcodes einiger Aufgaben notwendig.

```
//Vorher
  static {
     Random random= new Random();
  }

  //Nachher
  static {
     Random random= RandomFactory.regressionTestableRandom();
  }
```

```
// Vorher
    private static Random rnd= new Random(new Date().getTime());

// Nachher
    private static Random rnd;
    @Before void initRandom() {
       rnd = RandomFactory.regressionTestableRandom(new Date().getTime());
    }
```

Durchführung des Verhaltensabgleichs

Vergleiche Soll-Verhalten mit Ist-Verhalten. Gefundene Differenzen sind als potenzielle Regressionen zu behandeln → Testbericht.

Bewertungsabbrüche:

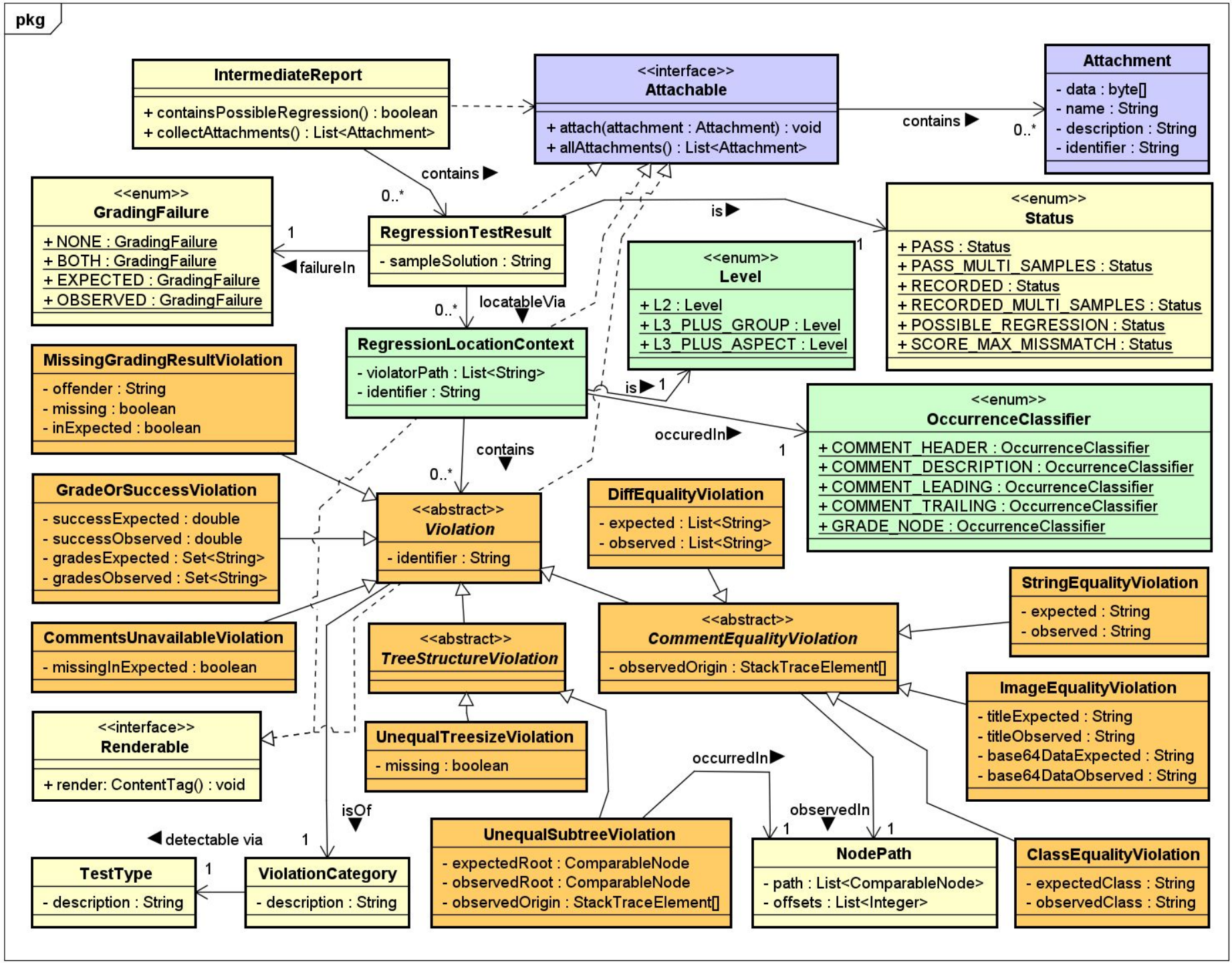
- binärer Mechanismus → Überprüfe ob beide Verhalten bzgl. eines Bewertungsabbruches denselben Zustand aufweisen.

Bewertungsschema:

- Überprüfe ob Bewertungsschema beider Verhalten die gleichen ProFormA-Tests mit identischen Bewertungsergebnissen aufweisen.

Kommentarbäume:

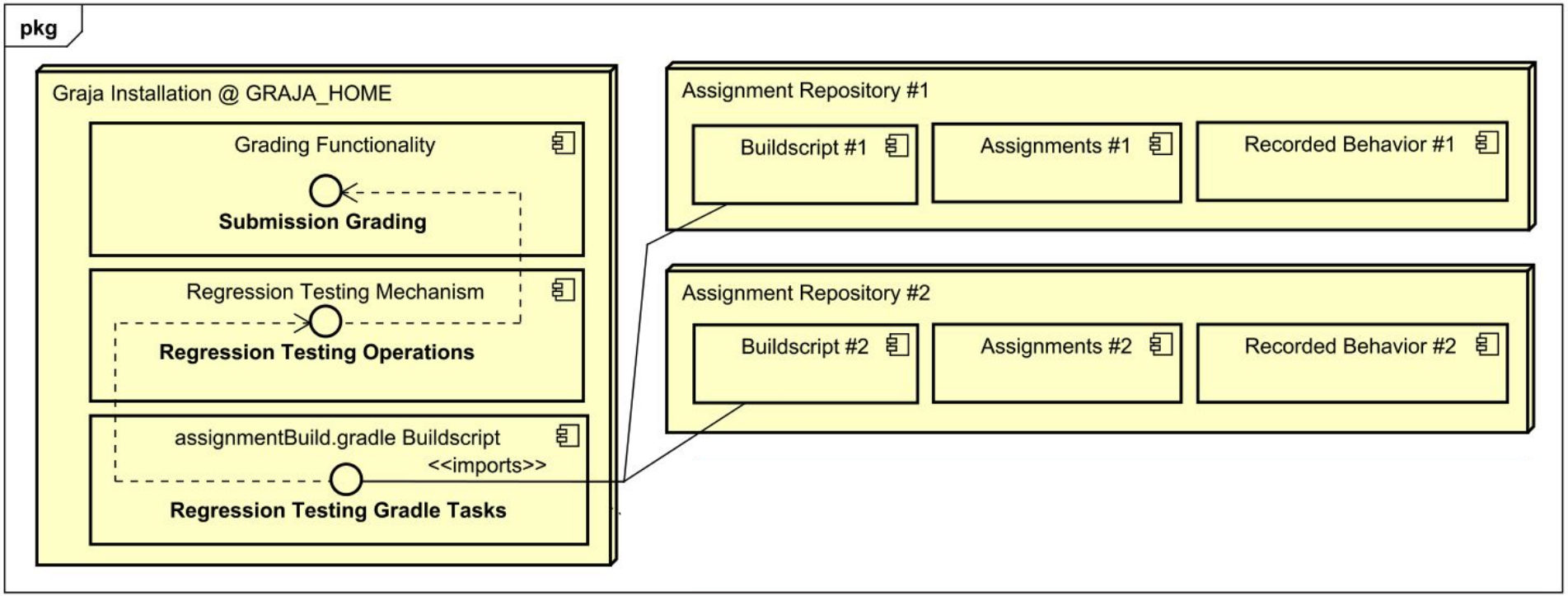
- Überprüfe Struktur der Kommentarbäume d.h. selbe Anzahl d. Knoten in gleicher Anordnung, gebe gefundene Differenzen aus.
- Überprüfe Inhalt der Knoten, die in beiden Kommentarbäumen existieren. Gebe Unterschiede aus, bei ungleichem Text die Lev. Distanz anwenden.



TESTBERICHT

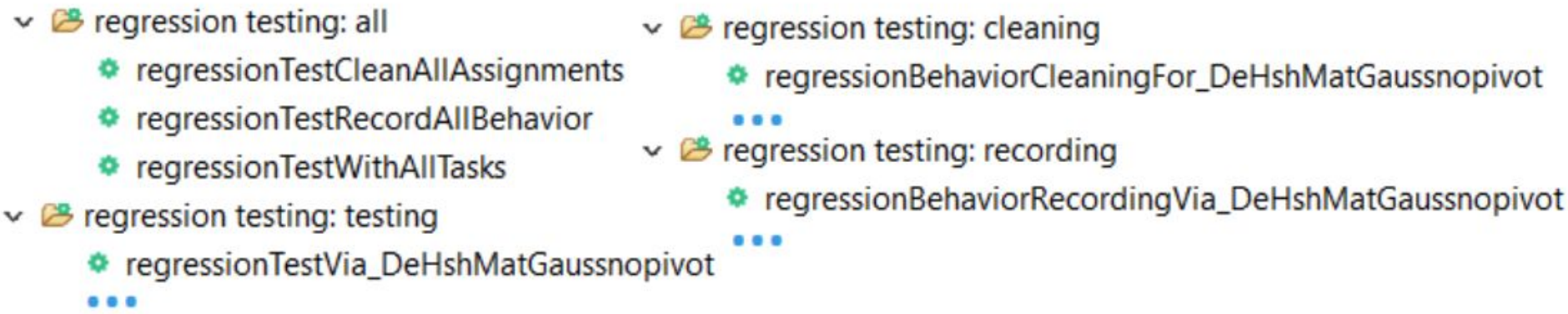
Aufbau einer Testinfrastruktur um Graja

- Operationen: `Test <Task>`, `Test All`, `Record <Task>`, `Record All` und `Clean`
- Integration mithilfe des verwendeten Build-Tools *Gradle* → Mechanismus ist als eigenes Modul fester Bestandteil von Graja
- Steuerungsparameter: Ignorieren von Musterlösungen / Aufgaben, Lev. Distanz (Musterlösungen- und Aufgaben-Lokal), *„multi sample mode“* über **`assignment.properties`**-Datei

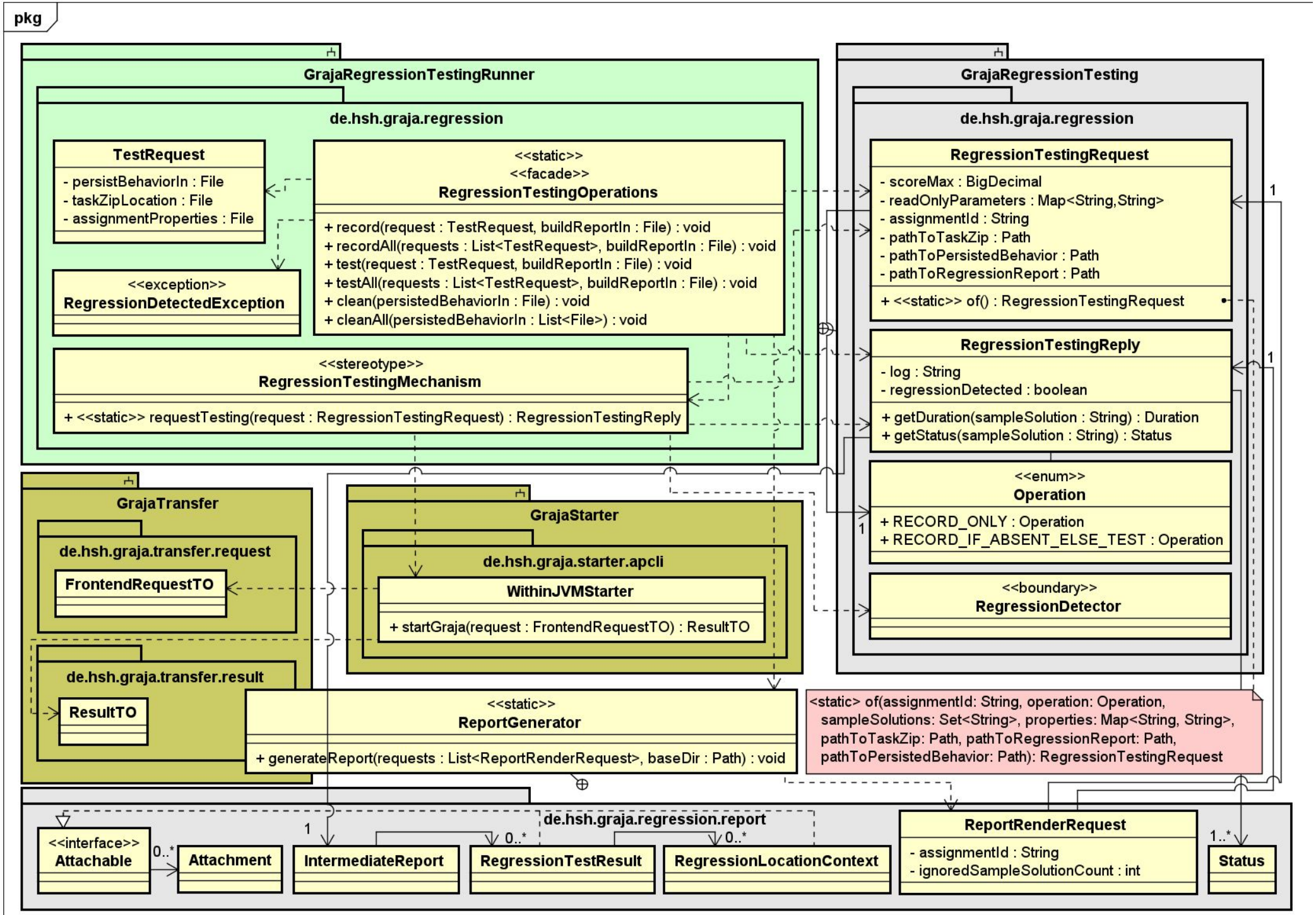


Referenzimplementierung

- Fester Teil von Graja durch zwei neue Module: **`GrajaRegressionTesting`** und **`GrajaRegressionTestingRunner`**
- Schnittstellen für **Java** + **Gradle**, Testbericht generiert im HTML-Format (angelehnt an durch Gradle generierte JUnit 4-Testberichte).



Java-Schnittstelle: Request/Reply-basiert. Sende Test-Request → erhalte Test-Reply. Dieser kann anschließend in einen HTML-Bericht transformiert werden.
Gradle-Schnittstelle: Request-basiert. Sende vereinfachten Test-Request (*task.zip*, *assignment.properties*, *Verhaltenspfad*) + *Pfad für Testbericht*. Ruft dann die Java-Schnittstelle auf und generiert anschließend einen HTML-Testbericht.


pkg
GrajaRegressionTestingRunner
de.hsh.graja.regression
TestRequest
- persistBehaviorIn : File
- taskZipLocation : File
- assignmentProperties : File
<<static>>
<<facade>>
RegressionTestingOperations
+ record(request : TestRequest, buildReportIn : File) : void
+ recordAll(requests : List<TestRequest>, buildReportIn : File) : void
+ test(request : TestRequest, buildReportIn : File) : void
+ testAll(requests : List<TestRequest>, buildReportIn : File) : void
+ clean(persistedBehaviorIn : File) : void
+ cleanAll(persistedBehaviorIn : List<File>) : void
<<exception>>
RegressionDetectedException
<<stereotype>>
RegressionTestingMechanism
+ <<static>> requestTesting(request : RegressionTestingRequest) : RegressionTestingReply
GrajaRegressionTesting
de.hsh.graja.regression
RegressionTestingRequest
- scoreMax : BigDecimal
- readOnlyParameters : Map<String,String>
- assignmentId : String
- pathToTaskZip : Path
- pathToPersistedBehavior : Path
- pathToRegressionReport : Path
+ <<static>> of() : RegressionTestingRequest
RegressionTestingReply
- log : String
- regressionDetected : boolean
+ getDuration(sampleSolution : String) : Duration
+ getStatus(sampleSolution : String) : Status
<<enum>>
Operation
+ RECORD_ONLY : Operation
+ RECORD_IF_ABSENT_ELSE_TEST : Operation
<<boundary>>
RegressionDetector
GrajaTransfer
de.hsh.graja.transfer.request
FrontendRequestTO
de.hsh.graja.transfer.result
ResultTO
GrajaStarter
de.hsh.graja.starter.apcli
WithinJVMStarter
+ startGraja(request : FrontendRequestTO) : ResultTO
<<static>>
ReportGenerator
+ generateReport(requests : List<ReportRenderRequest>, baseDir : Path) : void
<static> of(assignmentId: String, operation: Operation, sampleSolutions: Set<String>, properties: Map<String, String>, pathToTaskZip: Path, pathToRegressionReport: Path, pathToPersistedBehavior: Path): RegressionTestingRequest
de.hsh.graja.regression.report
<<interface>>
Attachable
Attachment
IntermediateReport
RegressionTestResult
RegressionLocationContext
ReportRenderRequest
- assignmentId : String
- ignoredSampleSolutionCount : int
Status
0..*
1
1..*

Quelle: https://camius.com/camius_best_suvreillance_ip_cameras_with_advanced_centralized_video_management_software/start-live-demo/

Fazit

- Regressionstestmechanismus mittels Musterlösungen in Graja als erstes Konzept umsetzbar, aber noch ausbaufähig.

- **Langsam** → Sequentielle Abarbeitung, Overhead durch Graja- Frontend, da zweite JVM über externen Prozess gestartet wird.
- **Nutzen in der Praxis?** **Klärt sich erst durch Einsatz in der Entwicklung!**
- **Übertragbarkeit auf andere Grader?** → Müsste pro Grader-Basis geprüft werden. Grobes Konzept wahrscheinlich, Graja spezifisches eher nicht.
- **Lokalisierung / Verständlichkeit d. Regressionen** → Momentan noch Verbesserungswürdig. Bisher noch nicht möglich Regressionen eindeutig in allen Fällen auf Stelle im Quellcode zurückzuführen.
- **Fehlalarme** → Gute Steuerungsparameter, besser mögl. mit Container.

Diskussion

Vielen Dank für Ihre Aufmerksamkeit!

Fragen können nun gestellt und beantwortet werden.